INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or “target” for pages apparently lacking from the document photographed is “Missing Page(s)”. If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of “sectioning” the material has been followed. It is customary to begin filming at the upper left hand comer of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.
Exact generalized inverses and solution to linear least squares problems using multiple modulus residue arithmetic

by

Sallie Ann Keller McNulty

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Major: Statistics

Approved:

In Charge of Major Work

For the Major Département

For the Graduate College

Iowa State University
Ames, Iowa

1983
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2.</td>
<td>RESIDUE ARITHMETIC AND NONSINGULAR MATRIX INVERSION USING MULTIPLE MODULUS RESIDUE ARITHMETIC</td>
<td>6</td>
</tr>
<tr>
<td>2.1</td>
<td>Residue Arithmetic for Integers</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Residue Arithmetic for Matrices</td>
<td>22</td>
</tr>
<tr>
<td>2.3</td>
<td>Error-Free Nonsingular Matrix Inversion</td>
<td>33</td>
</tr>
<tr>
<td>2.4</td>
<td>Alternative Solution to the Error-Free Nonsingular Matrix Inversion Problem</td>
<td>38</td>
</tr>
<tr>
<td>3.</td>
<td>GENERALIZED INVERSES USING MULTIPLE MODULUS RESIDUE ARITHMETIC</td>
<td>45</td>
</tr>
<tr>
<td>3.1</td>
<td>Existence of Generalized Inverses Over Various Fields and Rings</td>
<td>46</td>
</tr>
<tr>
<td>3.2</td>
<td>Multiple Modulus Residue Arithmetic Generalized Matrix Inversion</td>
<td>61</td>
</tr>
<tr>
<td>3.3</td>
<td>The Multiple Modulus Base and Corresponding Product Modulus</td>
<td>74</td>
</tr>
<tr>
<td>3.4</td>
<td>Algorithms for Finding Generalized Inverses Using Multiple Modulus Residue Arithmetic</td>
<td>84</td>
</tr>
<tr>
<td>4.</td>
<td>MOORE–PENROSE INVERSE USING MULTIPLE MODULUS RESIDUE ARITHMETIC</td>
<td>105</td>
</tr>
<tr>
<td>4.1</td>
<td>Existence of the Moore–Penrose Inverse Over Various Fields and Rings</td>
<td>106</td>
</tr>
<tr>
<td>4.2</td>
<td>Multiple Modulus Residue Arithmetic Moore–Penrose Inverse</td>
<td>110</td>
</tr>
<tr>
<td>5.</td>
<td>LINEAR LEAST SQUARES SOLUTIONS USING MULTIPLE MODULUS RESIDUE ARITHMETIC</td>
<td>123</td>
</tr>
<tr>
<td>5.1</td>
<td>The Multiple Modulus Residue Arithmetic Least Squares Solution $\mathbf{b}_1 = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$</td>
<td>126</td>
</tr>
<tr>
<td>5.2</td>
<td>The Multiple Modulus Residue Arithmetic Least Squares Solution $\mathbf{b}_2 = \mathbf{X}^+\mathbf{y}$</td>
<td>131</td>
</tr>
<tr>
<td>6. RATIONAL MATRICES</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>6.1 Scaling a Rational Matrix to be Used for Multiple Modulus Residue Arithmetic</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>6.2 Scaling Rational Matrices on the Computer</td>
<td>141</td>
<td></td>
</tr>
<tr>
<td>7. COMPUTER IMPLEMENTATION</td>
<td>149</td>
<td></td>
</tr>
<tr>
<td>7.1 Product Modulus Bound and the Multiple Modulus Base</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>7.2 Implementing Multiple Modulus Residue Arithmetic and the Symmetric Mixed Radix Representation</td>
<td>154</td>
<td></td>
</tr>
<tr>
<td>7.3 Examples</td>
<td>166</td>
<td></td>
</tr>
<tr>
<td>8. BIBLIOGRAPHY</td>
<td>178</td>
<td></td>
</tr>
<tr>
<td>9. ACKNOWLEDGMENTS</td>
<td>183</td>
<td></td>
</tr>
<tr>
<td>10. APPENDIX</td>
<td>184</td>
<td></td>
</tr>
<tr>
<td>10.1 The Gaussian Elimination Method FORTRAN Program</td>
<td>184</td>
<td></td>
</tr>
<tr>
<td>10.2 The Bordering Method FORTRAN Program</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>10.3 The Moore-Penrose Inversion Method FORTRAN Program</td>
<td>216</td>
<td></td>
</tr>
<tr>
<td>10.4 The Minimum Euclidean Norm Least Squares Method FORTRAN Program</td>
<td>227</td>
<td></td>
</tr>
</tbody>
</table>
LIST OF NOTATIONS

Z - set of all integers
Z⁺ - set of positive integers
G(p) - Galois field generated by a prime number p
R(M) - commutative ring generated by a composite number M
|x|ₘ - residue of the scalar x modulo m
/x/ₘ - symmetric residue of the scalar x modulo m
|A|ₘ - residue of the matrix A modulo m
/A/ₘ - symmetric residue of the matrix A modulo m
x⁻¹(m) - inverse of the scalar x modulo m or the inverse of x over R(m)
x⁻¹(p) - inverse of the scalar x modulo p, p-prime, or the inverse of x over G(p)
A⁻¹(m) - inverse of the matrix A modulo m or the inverse of |A|ₘ over R(m)
A⁻¹(p) - inverse of the matrix A modulo p, p-prime, or the inverse of |A|ₚ over G(p)
Aₚ adj(p) - adjoint of the matrix |A|ₚ, p-prime, over G(p)
A - generalized inverse of the matrix A over the field of real or rational numbers (specified in context.)
A⁻¹(m) - generalized inverse of the matrix |A|ₘ over R(m)
A⁻¹(p) - generalized inverse of the matrix |A|ₚ, p-prime, over G(p)
$A^+$ - Moore-Penrose inverse of the matrix $A$ over the field of real or rational numbers (specified in context.)

$A^+(m)$ - Moore-Penrose inverse of the matrix $|A|^m$ over $R(m)$

$A^+(p)$ - Moore-Penrose inverse of the matrix $|A|_p$, $p$-prime, over $G(p)$

$\text{rank}(A)$ - maximum number of linearly independent columns in the matrix $A$

$\text{rank}(|A|_p)$ - maximum number of linearly independent columns in the matrix $|A|_p$, linearly independent over $G(p)$

$\{p_1, p_2, \ldots, p_s\}$ - multiple modulus base, $p_i$-prime, $i = 1, 2, \ldots, s$

$M = \prod_{i=1}^{s} p_i$ - product modulus for the multiple modulus base $\{p_1, p_2, \ldots, p_s\}$
1. INTRODUCTION

A problem which is frequently present in statistical computing is generalized matrix inversion and the use of these generalized inverses in the solution to linear least square problems. Many different numerical methods have been proposed and used for obtaining generalized inverses and computing linear least squares parameter estimates. Most of these methods require the aid of the computer. Since floating-point arithmetic is normally used in computer solutions, the accuracy of solutions obtained for differing data conditions and algorithm stability has been extensively studied. The result of this effort is that there are several good algorithms available which provide reasonably accurate generalized inverses for most matrices, but none is entirely satisfactory. In fact, disastrously inaccurate results are sometimes obtained from all of the algorithms which utilize the standard floating-point arithmetic provided by the computer hardware. This is because the numerical theory behind these algorithms is based on the real number system but the computer's floating-point number system does not accurately model the real number system.

This research will develop generalized matrix inversion methods and linear least squares solutions, based on the rational number system, which can be implemented on the computer to generate error-free solutions. Implementation of these methods in floating-point arithmetic will once again cause inaccurate results because the floating-point number system cannot accurately model the rational number system either. A multiple
modulus number system, however, is capable of exactly modeling the rational number system. Therefore, with only the assumption that the problem has rational entries, using a multiple modulus number system (implemented in fixed-point arithmetic) an exact solution can be sought. This assumption poses no difficulty in problems requiring computer generated solutions because computers can only deal with rational numbers.

Available in the published literature, there are methods for obtaining exact solutions (\(\infty\) - precision) to full-rank systems of linear equations which use congruence techniques (i.e., residue arithmetic). The reader is referred to Borosh and Fraenkel (1966), Newman (1967), Bariess (1968 and 1972), Howell and Gregory (1969a, 1969b, and 1970), Fraenkel and Loewenthal (1971), Cabay (1971), Cabay and Lam (1977a), and Gregory (1980) for discussion of the theory and application of such methods. The works of Newman, Cabay and Lam, and Howell and Gregory will be discussed in detail in Chapter 2. There has also been some research done by Stallings and Bouillon (1972), Rao et al. (1976), Adegbeyeni and Krishnamurthy (1977) and Sen and Shamin (1978) on finding error-free generalized inverses of rational matrices using a single modulus number system. Stalling and Boullion's work on computing an exact Moore-Penrose inverse will be discussed in depth in Chapter 4.

An undesirable feature in some of the previous work done on the exact generalized matrix inversion problem is that special information like the rank or condition of the matrix must be known at the onset of
the problem. Another drawback with some of these methods is they require working with complicated forms like $A'(AA'AA')^{-1}AA'$, $A$ an $n \times s$ integral matrix, which are not practical from a computer implementation point of view. The methods developed in this research do not require any a-priori information about the matrix or the condition of the data, and the forms of the error-free generalized inverse and the exact linear least squares parameter estimates are kept as simple as possible.

A computer implemented multiple modulus residue arithmetic number system can be used to generate exact solutions, whereas the computer's fixed-point arithmetic system cannot. This is because once a rational problem is scaled to integers, computation done in fixed-point arithmetic on the computer will quickly overflow the permissible fixed-point number range. Using a large enough multiple modulus residue arithmetic number system is equivalent to doing fixed-point arithmetic on the computer for a fixed-point number system with an unlimited (theoretically) fixed-point number range. This idea was recognized by Kinoshita et al. (1974), O'Keefe (1975), Asai (1976), Gregory and Matula (1977), Aberth (1978), Barsi and Maestrini (1978 and 1981), and Okeke (1979). Their research involved looking at ways to work with arithmetic codes in a residue number system on the computer, converting from one multiple modulus number system to another, and performing residue arithmetic in the computer floating-point arithmetic mode.

As can be imagined, implementing a multiple modulus residue arithmetic method requires more computer storage and more computing
time than straight floating-point or fixed-point arithmetic methods. The cost of computer time and storage is very small compared to the cost only a few years ago. Thus, it seems reasonable to look for algorithms which guarantee highly accurate solutions even though some increase in solving time will surely be realized using such algorithms.

An alternative to using residue arithmetic methods for finding error-free solutions is to use a finite segment $p$-adic number system. This problem has been approached by Krishnamurthy et al. (1975a and 1975b), Krishnamurthy (1977), Gregory (1978 and 1980), and Horspool et al. (1978).

Some other areas of interest tangentially related to this research are computing exact inverses of matrices with polynomial entries, error-free computation of characteristic polynomials, and exact polynomial factorization. For discussions of these topics the reader is referred to Brown (1971), McClellan (1973 and 1977a), Horwitz and Sahni (1975), Musser (1975 and 1978), and Rao (1978).

Some will view this research as an interesting exercise in number theory, others will view it as an exercise in computer programming. Rice (1981) points out that developing mathematical software incorporates both of these things. One cannot simply sit down at the computer and program software to solve complex problems without having developed the theory behind the process first. The emphasis of this research has
been to lay the groundwork and develop the theory necessary to find exact generalized inverses of rational matrices and solutions to linear least squares problems with the aid of the computer.
2. RESIDUE ARITHMETIC AND NONSINGULAR MATRIX INVERSION
USING MULTIPLE MODULUS RESIDUE ARITHMETIC

Given an \( n \times n \) nonsingular integral matrix \( A \) it is possible to compute an error-free inverse of \( A \) using multiple modulus residue arithmetic. Howell and Gregory (1969a,b) and Cabay and Lam (1977a,b) have proposed solutions to this problem. Both of these solutions share a common pitfall that will be discussed in Section 2.3. An additional multiple modulus matrix rank determination result which eliminates this problem is given in Section 2.4. But first it is necessary to review some of the definitions and theorems on residue arithmetic for integers and matrices. The results in Section 2.1 on integers can be found in Szabó and Tanaka (1967). The results in Section 2.2 on matrices can be found in Howell and Gregory (1969a,b). The theorems found in these sources will be stated without proof.

2.1 Residue Arithmetic for Integers

Let \( \mathbb{Z} \) represent the set of all integers and let \( \mathbb{Z}^+ \) represent the set of positive integers. If \( a, b \neq 0, n \in \mathbb{Z} \) and \( a = nb \) then \( b \) divides \( a \). This will be denoted as \( b \mid a \). If there does not exist any \( n \in \mathbb{Z} \) such that \( a = nb \), then we say \( b \) does not divide \( a \) and denote it by \( b \notmid a \).

**Definition 2.1.1**  Given \( a, b \) and \( m \in \mathbb{Z}, m \neq 0 \), if \( m \mid a-b \), we write

\[
a \equiv b \pmod{m}
\]
and say $a$ is congruent to $b$ modulo $m$. Obviously, if $m \mid a-b$ then $-m \mid a-b$, hence, without loss of generality, we will assume $m \in \mathbb{Z}^+$. 

**Definition 2.1.2** Given $x, r \in \mathbb{Z}, m \in \mathbb{Z}^+$ if

$$r \equiv x \pmod{m},$$

and if $0 < r \leq m$, then we write

$$r = \left|x\right|_m$$

and say $r$ is a residue of $x$ modulo $m$.

The definition of a residue modulo $m$ will be used extensively throughout this work. The next theorem answers the question of uniqueness of a residue modulo $m$.

**Theorem 2.1.1** Given $x \in \mathbb{Z}$ and $m \in \mathbb{Z}^+$, $\left|x\right|_m$ is unique.

Theorem 2.1.1 states that $\left|x\right|_m$ is unique, however, there are many integers that have the same residue modulo $m$.

**Example 2.1.1** Let $m = 7$ then

$$2 = \left|9\right|_7 = \left|-5\right|_7 = \left|16\right|_7 = \left|-12\right|_7$$

and the list could go on indefinitely.
Numbers which have the same residue modulo m belong to the same residue class. Table 2.1.1 exhibits the residue classes modulo 7.

<table>
<thead>
<tr>
<th>Table 2.1.1</th>
<th>Residue Classes Modulo 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>21 22 23 24 25 26 27</td>
<td></td>
</tr>
<tr>
<td>14 15 16 17 18 19 20</td>
<td></td>
</tr>
<tr>
<td>7 8 9 10 11 12 13</td>
<td></td>
</tr>
<tr>
<td>0 1 2 3 4 5 6</td>
<td></td>
</tr>
<tr>
<td>-7 -6 -5 -4 -3 -2 -1</td>
<td></td>
</tr>
<tr>
<td>-14 -13 -12 -11 -10 -9 -8</td>
<td></td>
</tr>
<tr>
<td>-21 -20 -19 -18 -17 -16 -15</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Numbers in the same column belong to the same residue class.

The idea of \( r = |x|_m \) refers to the least nonnegative remainder of \( x \) after division by \( m \). The possible values for \( |x|_m \) lie in the interval \([0, m-1]\). The next definition defines a residue number system isomorphic to that in Definition 2.1.1.
Definition 2.1.3  
Given \( x, r \in \mathbb{Z} \) and \( m \in \mathbb{Z}^+ \), if
\[
r \equiv x \pmod{m}
\]
and if
\[
-\frac{m}{2} < r < \frac{m}{2}
\]
then we write
\[
r = /x/_m
\]
and say \( r \) is the symmetric residue of \( x \) modulo \( m \).

There is a one-to-one mapping between \( |x|_m \) and \( /x/_m \). If
\[
0 \leq |x|_m < \frac{m}{2}
\]
then \( /x/_m = |x|_m \),
otherwise, if
\[
|x|_m > \frac{m}{2}
\]
then \( /x/_m = |x|_m - m \).

With this mapping, if \( m \) is odd the nonnegative interval \([0, m-1]\) of integers is mapped onto the symmetric interval of integers \( \left[ -\left(\frac{m-1}{2}\right), \frac{(m-1)}{2} \right] \).

Example 2.1.2  
Let \( m = 5 \), then
\[
|x|_5 \in \{0, 1, 2, 3, 4\}
\]
\[
/x/_5 \in \{-2, -1, 0, 1, 2\}.
\]

Just as \( |x|_m \) is unique, \( /x/_m \) is unique.
Given an $s$-tuple of moduli $\{m_1, m_2, \ldots, m_s\}$, this can be used as the base of a multiple modulus residue number system.

**Definition 2.1.4** Given $x \in \mathbb{Z}$ and $m_i \in \mathbb{Z}^+$, $i = 1, 2, \ldots, s$, then the multiple modulus residue representation of $x$, for the base $\{m_1, m_2, \ldots, m_s\}$ is the $s$-tuple

$$x \sim \{ |x|_{m_1}, |x|_{m_2}, \ldots, |x|_{m_s} \}.$$ 

The analogous definition which pertains to symmetric residues is as follows.

**Definition 2.1.5** Given $x \in \mathbb{Z}$ and $m_i \in \mathbb{Z}^+$, $i = 1, 2, \ldots, s$, then the symmetric multiple modulus residue representation of $x$, for the base $\{m_1, m_2, \ldots, m_s\}$ is the $s$-tuple

$$x \sim \{ x/m_1, x/m_2, \ldots, x/m_s \}.$$ 

Single modulus residue number systems are simply a special case of the multiple modulus residue number systems with $s = 1$. Consequently, one would expect the uniqueness property to carry over.

**Theorem 2.1.2** For a given base, the multiple modulus residue representation of each integer is unique.
As in the single modulus case, it is possible for $x$ and $y$ ($x \neq y$) to have the same multiple modulus residue representation for a given base.

**Example 2.1.3** Let the base be $\{2, 3, 4\}$, then

- $10 \sim \{0, 1, 2\}$
- $22 \sim \{0, 1, 2\}$.

The next theorem clearly points out that there is not a one-to-one correspondence between the integers and their multiple modulus residue representations for a given base.

**Theorem 2.1.3** Two integers $x$ and $y$ have the same multiple modulus residue representation for the base $\{m_1, m_2, \ldots, m_s\}$ if and only if

$$x \equiv y \pmod{M}$$

where $M$ is the least common multiple (lcm) of the moduli in the base.

In Example 2.1.3 $M = \text{lcm}(2, 3, 4) = 12$ and $10 \equiv 22 \pmod{12}$. All the members of the residue class modulo $M$ are mapped onto the same multiple modulus residue representation. The next corollary gives a one-to-one correspondence between a set of integers and their multiple modulus residue representations.
Corollary 2.1.3.1. For a given base \( \{ m_1, m_2, \ldots, m_s \} \), the multiple modulus residue representations of the integers \( x \) in the range
\[ 0 \leq x \leq M-1 \]
where \( M = \text{lcm} \{ m_1, m_2, \ldots, m_s \} \) are distinct. Also, the symmetric multiple modulus residue representations of the integers \( x^* \) in the range
\[ -\frac{M}{2} < x^* \leq \frac{M}{2} \]
are distinct.

A convenient base to choose is one that consists of prime moduli, say \( \{ p_1, p_2, \ldots, p_s \} \), because in this case
\[ M = \text{lcm} \{ p_1, p_2, \ldots, p_s \} = \prod_{i=1}^{s} p_i . \]

Example 2.1.4 Let the base be \( \{3, 5\} \). Then,
\[ M = 3 \cdot 5 = 15 . \]
The multiple modulus residue representations of \( x \in [0, 14] \) are
\[
\begin{align*}
0 &\sim \{0, 0\} & 7 &\sim \{1, 2\} \\
1 &\sim \{1, 1\} & 8 &\sim \{2, 3\} \\
2 &\sim \{2, 2\} & 9 &\sim \{0, 4\} \\
3 &\sim \{0, 3\} & 10 &\sim \{1, 0\} \\
4 &\sim \{1, 4\} & 11 &\sim \{2, 1\} \\
5 &\sim \{2, 0\} & 12 &\sim \{0, 2\} \\
6 &\sim \{0, 1\} & 13 &\sim \{1, 3\} \\
14 &\sim \{2, 4\} &
\end{align*}
\]
and these are all distinct. So, given the base \{3, 5\} and \(x \sim \{2, 1\}\) the only integer in the interval \([0, 14]\) that \(x\) could represent is 11.

The next theorem gives the basic addition, subtraction and product rules for residue arithmetic. From this point on it will be assumed that for a given base \(\{m_1, m_2, \ldots, m_s\}\) each pair of moduli in the base are relatively prime, this is denoted by \((m_i, m_j) = 1\) for \(i \neq j\).

Theorem 2.1.4 Let \(\{m_1, m_2, \ldots, m_s\}\) be the base for a residue number system, where \((m_i, m_j) = 1\) for \(i \neq j\), and

\[
M = \prod_{i=1}^{s} m_i^s.
\]

Then the multiple modulus residue representation of \(|x \pm y|_M\) is given by

\[
x \sim \{|x|_{m_1}, |x|_{m_2}, \ldots, |x|_{m_s}\}
\]

\[
y \sim \{|y|_{m_1}, |y|_{m_2}, \ldots, |y|_{m_s}\}
\]

\[
|x \pm y|_M \sim \{|z_1|_{m_1}, |z_2|_{m_2}, \ldots, |z_s|_{m_s}\}
\]

where

\[
|z_1|_{m_1} = \left|\frac{x_{m_1} \pm y_{m_1}}{m_1}\right|
\]

\[
= \left|\frac{x \pm y}{m_1}\right|
\]

\[
= \left|\frac{x}{m_1} \pm \frac{y}{m_1}\right|
\]

\[
= \left|\frac{x \pm y}{m_1}\right|
\]
The multiple modulus residue representation of $|x^*y|_M$ is given by

$$x \sim \{ |x|_{m_1}, |x|_{m_2}, \ldots, |x|_{m_s} \}$$

$$y \sim \{ |y|_{m_1}, |y|_{m_2}, \ldots, |y|_{m_s} \}$$

$$|x^*y|_M \sim \{ |w_1|_{m_1}, |w_2|_{m_2}, \ldots, |w_s|_{m_s} \}$$

where

$$|w_i|_{m_i} = |x|_{m_i} \cdot |y|_{m_i}$$

$$= |x|_{m_i} \cdot |y|_{m_i}$$

$$= |x|_{m_i} \cdot y$$

$$= |x \cdot y|_{m_i}$$

As the theorem indicates, for a given modulus $m_i$, when invoking these three arithmetic operations the results can be reduced modulo $m_i$ at any intermediate step of computation.

**Example 2.1.5** Let $m = 5$,

$$|12 \cdot 6 + 9 - 107|_5 = \left| \left| 12 |_{5} \cdot |6|_5 \right|_5 - |9|_5 - |107|_5 \right|_5$$

$$= \left| \left| 2 \cdot 1 |_5 + 4 \right|_5 - 2 \right|_5$$
The next basic arithmetic operation that needs to be defined is division. First, it is necessary to discuss the multiplicative inverse modulo $m$.

**Definition 2.1.6** If $x, y \in \mathbb{Z}$ and $m \in \mathbb{Z}^+$, and if

1. $0 < y < m$

2. $|x^y|_m = |y^x|_m = 1$

then we write

$$y = x^{-1}(m)$$

and say $y$ is the multiplicative inverse of $x$ modulo $m$.

**Theorem 2.1.5** If $x \in \mathbb{Z}$, $m \in \mathbb{Z}^+$, then $x^{-1}(m)$ exists if and only if

1. $|x|_m \neq 0$

2. $(x, m) = 1$

**Theorem 2.1.6** If $x^{-1}(m)$ exists, it is unique.
Division can be performed to a limited extent, that is, if the quotient of two integers is again an integer, then the residue of this quotient can be computed.

**Theorem 2.1.7** If \( x, y \in \mathbb{Z}, m \in \mathbb{Z}^+ \), and if

i) \( x \mid y \)

ii) \( x^{-1}(m) \) exists,

then \( \frac{y}{x} \equiv \frac{y \cdot x^{-1}(m)}{m} \).

Given \( x \) and \( m \) to satisfy Theorem 2.1.6, the Euclidean Algorithm may be used to find \( x^{-1}(m) \). This is discussed in detail in Chapter 7.

**Example 2.1.6** Let \( m = 7 \) find \( \frac{24}{12} \), \( |12|_7 = 5 \neq 0 \), \( (12, 7) = 1 \)

and \( 5 \cdot 3 \equiv 1 \pmod{7} \), so, \( 12^{-1}(7) = 3 \) and since \( 12 \mid 24 \) we have

\[
\frac{24}{12} \equiv \frac{|24|_7 \cdot 12^{-1}(7)}{7} \\
= \frac{|24|_7 \cdot 5^{-1}(7)}{7} \\
= |3 \cdot 3|_7 \\
= |9|_7 \\
= 2
\]
Now is perhaps a good time to point out some properties of specific sets of integers. The set of integers \( \{0, 1, 2, \ldots, p-1\} \) where \( p \) is a prime number forms a Galois field with the operations + and \( \cdot \) as defined in Theorem 2.1.4. We call this the Galois field generated by \( p \) and denote it as \( G(p) \). The set of integers \( \{0, 1, 2, \ldots, M-1\} \) where \( M \) is a composite number forms a commutative ring with the operations + and \( \cdot \) as defined in Theorem 2.1.4. We call this the ring generated by \( M \) and denote it as \( R(M) \). The multiplicative inverse of \( x \) modulo \( p \), \( x^{-1}(p) \), can be thought of as the multiplicative inverse of \( x \) over \( G(p) \). According to Theorem 2.1.5, there exists a multiplicative inverse over \( G(p) \) for every nonzero element in \( G(p) \). There does not exist a multiplicative inverse over \( R(M) \) for every nonzero element in \( R(M) \). This is one of the main differences between \( R(M) \) and \( G(p) \).

**Example 2.1.7** Let \( G(5) = \{0, 1, 2, 3, 4\} \) and \( R(6) = \{0, 1, 2, 3, 4, 5\} \). Now, \( 3^{-1}(5) = 2 \) over \( G(5) \) since \( (3,5) = 1 \). but \( 3^{-1}(6) \) does not exist over \( R(6) \) because \( (3,6) = 3 \).

The field (or ring) operations of + and \( \cdot \) are defined as in Theorem 2.1.4 so they are identical to the residue arithmetic operations of + and \( \cdot \). There is a subtle difference between the residue arithmetic operation of
division, as defined in Theorem 2.1.7, and the field (or ring) operation of division. By Theorem 2.1.7, the residue modulo M of a quotient \( \frac{y}{x} \) is only defined when \( x | y \). If \( x \not| y \) the quotient \( \frac{y}{x} \) is not an integer, consequently \( |\frac{y}{x}|_M \) is not defined. Now consider the operation of division over the ring generated by M. If \( x^{-1}(M) \) exists over \( R(M) \) then the quotient \( \frac{y}{x} \) over \( R(M) \) is defined as

\[
\left[ \frac{y}{x} \right](M) = |y \cdot x^{-1}(m)|_M.
\]

In this case \( x \) need not divide \( y \). The quantity \( \left[ \frac{y}{x} \right](M) \) which denotes the quotient \( \frac{y}{x} \) over \( R(M) \) must be interpreted very carefully and should not be confused with \( |\frac{y}{x}|_M \). It is true that if \( x | y \) and \( x^{-1}(M) \) exists then \( |\frac{y}{x}|_M = \left[ \frac{y}{x} \right](M) \). The following example helps to clarify this difference that has just been discussed.

**Example 2.1.8** For \( p = 11 \), find the residue of \( \left[ \frac{3 \cdot 6}{9} \right] \) modulo 11. First, we will solve this by working over \( G(11) \).

\[
\left[ \frac{3 \cdot 6}{9} \right](11) = \left| \frac{3}{9} \right|(11) \cdot 6_{11} \quad \left( = \left| \frac{3 \cdot 6}{9} \right|(11) \right)_{11}
\]

\[
= \left| 3 \cdot 9^{-1}(11) \right| \cdot 6_{11} \quad \left( = \left| 3 \cdot 9^{-1}(11) \right|_{11} \right)
\]

\[
= \left| 3 \cdot 5 \right|_{11} \cdot 6_{11} \quad \left( = \left| 3 \cdot 5 \right|_{11} \right)
\]

\[
= \left| 15 \right|_{11} \cdot 6_{11} \quad \left( = \left| 30 \right|_{11} \right)
\]
Second, we will find the residue of \( \frac{3 \cdot 6}{9} \) modulo 11 by applying residue arithmetic to this equation.

Notice that 9 \( \not\mid \) 3 and 9 \( \not\mid \) 6, so the product \( 3 \cdot 6 = 18 \) must be formed and left un reduced modulo 11. Now, 9 \( \mid \) 18 and we have

\[
\frac{3 \cdot 6}{9} \equiv \frac{18}{9} \pmod{11}
\]

\[
= \left| \frac{18}{9} \right|_{11} = \left| 2 \right|_{11}
\]

If we had reduced \( \left| \frac{18}{9} \right|_{11} \) at the start we would have

\[
\left| \frac{3 \cdot 6}{9} \right|_{11} = \left| \frac{18}{9} \right|_{11} = \left| \frac{18}{9} \right|_{11} = \left| \frac{7}{9} \right|_{11}
\]

But 9 \( \not\mid \) 7 which means that the division rule for residue arithmetic is not well-defined here. Although forming the product \( \left| 7 \cdot 9^{-1}(11) \right|_{11} \)
at this point leads to the correct solution, it also illustrates an attempt to find a quotient by applying residue arithmetic to the initial problem, getting stuck and then completing the solution by working over the appropriate Galois field. This method of problem solving should be avoided.

Having defined basic arithmetic operations, we can turn back to Corollary 2.1.3.1 and Example 2.1.4 and point out the real problem they help us solve. That is, given the base \( \{m_1, m_2, \ldots, m_s\} \), where \( (m_i, m_j) = 1 \) for \( i \neq j \), and a multiple modulus residue representation, \( \{r_1, r_2, \ldots, r_s\} \), how is the unique integer in the interval \([0, M-1]\) or \(\left\lfloor \frac{-N+1}{M_1}, \frac{N+1}{M_1}\right\rfloor\), \( M = \prod_{i=1}^{s} m_i \), that this represents determined? One solution to this problem is given by the Chinese Remainder Theorem.

**Theorem 2.1.8** Let \( \{m_1, m_2, \ldots, m_s\} \) be the base for a residue number system where \( (m_i, m_j) = 1 \) for \( i \neq j \), and let

\[
M = \prod_{i=1}^{s} m_i .
\]

Also, let

\[
\hat{M}_j = \frac{M}{m_j} .
\]

Now if \( x \) has the multiple modulus residue representation

\[
x \sim \{r_1, r_2, \ldots, r_s\}
\]
where \( r_i = |x|_{m_i} \), i = 1, 2, ..., s ,

then

\[
|x|_M = \left| \sum_{j=1}^{s} \hat{a}_j r_j \cdot \hat{a}_j^{-1}(m_j) \right|_{m_j, M}
\]

\[
= \left| \hat{a}_1 r_1 \cdot \hat{a}_1^{-1}(m_1) \right|_{m_1, M} + \cdots + \left| \hat{a}_s r_s \cdot \hat{a}_s^{-1}(m_s) \right|_{m_s, M}
\]

Example 2.1.9 Let the base be \( \{3, 5, 7\} \), then \( M = 3 \cdot 5 \cdot 7 = 105 \).

What integer in the interval \([0, 104]\) does \( \{1, 3, 2\} \) represent?

\( \hat{a}_1 = 35 \), \( |\hat{a}_1|_3 = 2 \), \( \hat{a}_1^{-1}(3) = 2^{-1}(3) = 2 \)

\( \hat{a}_2 = 21 \), \( |\hat{a}_2|_5 = 1 \), \( \hat{a}_2^{-1}(5) = 1^{-1}(5) = 1 \)

\( \hat{a}_3 = 15 \), \( |\hat{a}_3|_7 = 1 \), \( \hat{a}_3^{-1}(7) = 1^{-1}(7) = 1 \)

\[
|x|_{105} = \left| 35 \cdot |1\cdot 2|_3 + 21 \cdot |3\cdot 1|_5 + 15 \cdot |2\cdot 1|_7 \right|_{105}
\]

\[
= \left| 35 \cdot 2 + 21 \cdot 3 + 15 \cdot 2 \right|
\]

\[
= |70 + 63 + 30|_{105}
\]

\[
= |163|_{105}
\]

\[
= 58
\]

The unique integer in the interval \([-52, 52]\) is

\[
|x|_{105} = 58 - 105 = -47.
\]
From a computational standpoint, one undesirable quality of the Chinese Remainder Theorem is that it requires doing arithmetic modulo $M$ which in most cases is quite large. In Chapter 7, an alternative method which works with the symmetric multiple modulus residue representations and requires no arithmetic modulo $M$ is given.

This concludes the basic integer residue arithmetic theory necessary for this research. It should be pointed out that many more theorems and properties related to residue arithmetic exist and can be found in Szabó and Tanaka (1967) or in other Number Theory texts.

2.2 Residue Arithmetic for Matrices

The following material will contain some of the necessary matrix algebra theory for residue arithmetic. Many of these results are analogous to the results in Section 2.1.

**Definition 2.2.1** If $A = (a_{ij})$ and $B = (b_{ij})$ are $p \times q$ integral matrices and $m \in \mathbb{Z}^+$, and if

$$a_{ij} \equiv b_{ij} \pmod{m}$$

for all $i$ and $j$, then we write

$$A \equiv B \pmod{m}$$

and say $A$ is congruent to $B$ modulo $m$.

**Definition 2.2.2** Let $X = (x_{ij})$ be a $p \times q$ integral matrix and $m > 1$ an integer. If $R = (r_{ij})$ is the matrix with elements defined by
for all \( i \) and \( j \), then we write

\[ r_{ij} = |x_{ij}|_m \]

and say \( R \) is the residue of \( X \) modulo \( m \).

Since \( |x_{ij}|_m \) is unique for all \( i \) and \( j \), the next theorem should be obvious.

**Theorem 2.2.1**

Given any \( pxq \) integral matrix \( X \) and any integer \( m > 1 \), 

\[ |X|_m \]

is unique.

The next set of matrix algebra properties can be thought of in the following way. Let \( X \) and \( Y \) be integral matrices, \( m > 1 \) an integer, and let \( |X|_m \) and \( |Y|_m \) be the residue of \( X \) and \( Y \) modulo \( m \). If \( H(X, Y) \) is any function involving scalar addition or multiplication with respect to the matrix elements in \( X \) and \( Y \) then,

\[ |H(X, Y)|_m = |H(|X|_m, |Y|_m)|_m \]

This means that forming the function over the set of all integers and reducing the result modulo \( m \) is equivalent to mapping \( X \) and \( Y \) into the appropriate ring (or field) and forming the function over this ring (or field).
Theorem 2.2.2  If $X$ and $Y$ are $p \times q$ integral matrices and $m > 1$ is an integer then,

$$|X \pm Y|^m = \left| |X|^m \pm |Y|^m \right|_m.$$  

Also,

$$|X \cdot Y'|_m = \left| |X|^m \cdot |Y'|^m \right|_m$$

where $Y'$ and $|Y|^m$ are the transposes of $Y$ and $|Y|^m$ respectively.

Theorem 2.2.3  If $X$ is an $n \times n$ integral matrix and $m > 1$ is an integer then,

i) $|\text{trace}(X)|_m = |\text{tr}(X)|_m = |\text{tr}(|X|^m)|_m$

ii) $|\text{determinant}(X)|_m = |\text{det}(X)|_m = |\text{det}(|X|^m)|_m$.

Since we are interested in matrix inversion, it is necessary to define what is meant by $X^{-1}(m)$.

Definition 2.2.3  If $X$ and $Y$ are $n \times n$ integral matrices and $m > 1$ is an integer, and if

i) $|X \cdot Y|^m = I = |Y \cdot X|^m$

ii) $|Y|^m = Y$,

then we write

$$Y = X^{-1}(m)$$

and call $Y$ the multiplicative inverse of $X$ modulo $m$. 
Since $|Y \cdot X|_m = \left| Y \cdot \frac{|X|_m}{|X|_m} \right| = I = \left| \frac{|X|_m \cdot Y}{|X|_m} \right| = |X \cdot Y|_m$, without loss of generality we will refer to $X^{-1}(m)$ as the multiplicative inverse of the residue of $X$ modulo $m$, $|X|_m$. The elements in $X^{-1}(m)$ are from the ring (or field) so, $X^{-1}(m)$ is actually the inverse of $|X|_m$ over the ring (or field) generated by $m$. It is not thought of as $|X^{-1}|_m$ because $X^{-1}$ is most likely not integral. It is true that if $X^{-1} = \begin{bmatrix} p_{ij} \\ q_{ij} \end{bmatrix}$ and if $|X^{-1}|_m$ was formed as $(\left| p_{ij} \cdot q_{ij}^{-1}(m) \right|)_m$, providing $q_{ij}^{-1}(m)$ exists for all $i, j$, then $|X^{-1}|_m = X^{-1}(m)$. As discussed in Section 2.1, if $q_{ij} \nmid p_{ij}$ then $|p_{ij} \cdot q_{ij}^{-1}(m)|_m$ is not a well-defined residue arithmetic operation. It is, however, a valid ring operation. If $X^{-1}$ is not integral, $X^{-1}(m)$ has no meaning outside the context of the ring (or field) generated by $m$.

**Theorem 2.2.4** If $X$ is an $n \times n$ integral matrix and if $X^{-1}(m)$ exists, then it is unique.

The question of existence of $X^{-1}(m)$ needs to be answered, but first two more definitions.

**Definition 2.2.4** An $n \times n$ integral matrix $X$ is said to be nonsingular modulo $m$ if and only if both

i) $|\det(X)|_m \neq 0$

ii) $(\det(X), m) = 1$.
**Definition 2.2.5** If $X$ is an $n \times n$ integral matrix, the adjoint matrix modulo $m$ is defined to be

$$|X^{\text{adj}}|_m = |(X_{ij})|_m$$

where $X_{ij}$ are the cofactors of $X$.

Now the theorem on existence of $X^{-1}(m)$.

**Theorem 2.2.5** $X^{-1}(m)$ exists if and only if $X$ is nonsingular modulo $m$. In this case,

$$X^{-1}(m) = \left| \left\{ \det(X) \right\}^{-1}(m) \left| X^{\text{adj}} \right|_m \right|_m.$$

Saying that $X$ is nonsingular modulo $m$ is equivalent to saying $|X|_m$ is nonsingular over the ring (or field) generated by $m$. This is true since

$$|\det(X)|_m = |\det(|X|_m)|_m$$

and

$$|X^{\text{adj}}|_m = |(X_{ij})|_m$$

$$= |((|X|_m)_{ij})|_m$$

$$= X^{\text{adj}}(m),$$

where $X^{\text{adj}}(m)$ is the adjoint matrix of $|X|_m$ over the ring (or field) generated by $m$. Therefore,

$$X^{-1}(m) = \left| \left\{ \det(|X|_m) \right\}^{-1}(m) \cdot X^{\text{adj}}(m) \right|_m$$
and $X^{\text{adj}}(m) = |\det(|x|^m)| \cdot X^{-1}(m)|_m$ when $X^{-1}(m)$ exists. The $|\det(|x|^m)|_m$ and $X^{\text{adj}}(m)$ always exist for any $nxn$ integral matrix $X$ and any integer $m > 1$. In some cases, $|\det(|x|^m)|_m$ may be zero and $X^{\text{adj}}(m)$ may be the null matrix. In these situations, $X$ is singular modulo $m$ or $|x|^m$ is singular over the ring (or field) generated by $m$.

The next matrix property to be discussed is the rank of a matrix. We will assume the reader is familiar with what is meant by $\text{rank}(A)$, $A$ an $nxn$ real matrix, over the field of real numbers. The next few results can be found in Herstein (1975) or in other algebra textbooks. Herstein gives these results with respect to an arbitrary field $F$. We will be stating these results with respect to a Galois field generated by a prime number $p$, $G(p)$.

First, it is necessary to verify that a set of vectors, \{\alpha_1, \alpha_2, \ldots, \alpha_n\}, with entries from $G(p)$ form a vector space over $G(p)$. A general vector space definition is as follows.

**Definition 2.2.6** A nonempty set of vectors $V$ with entries from a field $F$ is said to be a vector space over $F$ if $V$ is a commutative group under an operation which can be denoted by $+$, and if for every $a \in F$ and $\alpha \in V$ there is defined an element, written $a \alpha$, in $V$ subject to
i) \( a \cdot (\alpha + \beta) = a \cdot \alpha + a \cdot \beta; \)

ii) \( (a + b) \cdot \alpha = a \cdot \alpha + b \cdot \alpha; \)

iii) \( a \cdot (b \cdot \alpha) = (a \cdot b) \cdot \alpha; \)

iv) \( 1 \cdot \alpha = \alpha; \)

for all \( a, b \in F \) and \( \alpha, \beta \in V \).

Let \( \alpha_p = (|\alpha_1|_p, |\alpha_2|_p, \ldots, |\alpha_s|_p) \) and \( \beta_p = (|\beta_1|_p, |\beta_2|_p, \ldots, |\beta_s|_p) \), \( \alpha_i, \beta_i \in \mathbb{Z}, i = 1, 2, \ldots, s \), be vectors with entries from \( G(p) \), p-prime.

Define the operation \( + \) in Definition 2.2.6 as

\[
|\alpha + \beta|_p = (|\gamma_1|_p, |\gamma_2|_p, \ldots, |\gamma_s|_p),
\]

where

\[
|\gamma_i|_p = |\alpha_i + \beta_i|_p = \left| |\alpha_i|_p + |\beta_i|_p \right|_p,
\]

for \( i = 1, 2, \ldots, s \). Let \( a \cdot \alpha_p, a \in \mathbb{Z} \), be defined as

\[
|a \cdot \alpha|_p = (|w_1|_p, |w_2|_p, \ldots, |w_s|_p),
\]

where

\[
|w_i|_p = |a \cdot \alpha_i|_p = \left| |a|_p \cdot |\alpha_i|_p \right|_p,
\]

then \( |a \cdot \alpha|_p \) is a vector with entries from \( G(p) \). With vector addition and scalar multiplication over \( G(p) \) defined in this way, a set of vectors with entries from \( G(p) \) form a vector space over \( G(p) \).
Definition 2.2.7  Let $V$ be a vector space over $G(p)$, $p$-prime, and $v_1, v_2, \ldots, v_n \in V$, then any vector of the form

$$ |a_1 v_1 + a_2 v_2 + \ldots + a_n v_n |_p = |a_1 v_1 |_p + |a_2 v_2 |_p + \ldots + |a_n v_n |_p $$

where $a_i \in \mathbb{Z}$ for $i = 1, 2, \ldots, n$, is a linear combination over $G(p)$ of the vectors $v_1, v_2, \ldots, v_n$.

Definition 2.2.8  Let $V$ be a vector space over $G(p)$, $p$-prime, then the vectors $v_1, v_2, \ldots, v_n \in V$ are said to be linearly dependent over $G(p)$ if there exists scalars $|a_1|_p, |a_2|_p, \ldots, |a_n|_p$, $a_i \in \mathbb{Z}$ for $i = 1, 2, \ldots, n$, not all zero, such that the linear combination over $G(p)$ of the vectors $v_1, v_2, \ldots, v_n$ is

$$ |a_1 v_1 + a_2 v_2 + \ldots + a_n v_n |_p = |a_1 v_1 |_p + |a_2 v_2 |_p + \ldots + |a_n v_n |_p $$

$$ = |a_1 v_1 |_p + |a_2 v_2 |_p + \ldots + $$

$$ = |a_n v_n |_p $$

$$ = 0. $$

A set of vectors with entries from $G(p)$ which are not linearly dependent over $G(p)$ are linearly independent over $G(p)$.

Definition 2.2.9  A subset $H$ of a vector space $V$ over $G(p)$, $p$-prime, is called a basis for $V$ if every vector in $V$ can be expressed as a linear combination over $G(p)$ of the vectors in $H$. 
Theorem 2.2.6 Any two bases for a vector space \( V \) over \( G(p) \),
p-prime, contain the same number of vectors.

Definition 2.2.10 The dimension of a vector space \( V \) over \( G(p) \),
p-prime, is the number of vectors in the basis for \( V \).

Definition 2.2.11 If \( S \) is a nonempty set of vectors with entries
from \( G(p) \), p-prime, then \( L(S) \), the linear span of \( S \), is the set of
all linear combinations over \( G(p) \) of the vectors in \( S \).

Theorem 2.2.7 If \( S \) is a nonempty set of vectors with entries
from \( G(p) \), p-prime, then \( L(S) \) forms a vector space over \( G(p) \).

Now, if \( |A|_p \), p-prime, is an \( n \times s \) matrix with entries from \( G(p) \),
then the columns of \( |A|_p \) span a vector space over \( G(p) \), \( L(|A|_p) \).
Also, the rows of \( |A|_p \) (i.e., the columns of \( |A|'_p \)) span a vector space
over \( G(p) \), \( L(|A|'_p) \).

Theorem 2.2.8 Let \( |A|_p \) be an \( n \times s \) matrix with entries from \( G(p) \),
p-prime, then

\[
\text{dimension of } L(|A|_p) = \text{dimension of } L(|A|'_p).
\]

With the aid of these definitions and theorems, the definition of
the rank\((|A|_p)\) can be understood. Consider the following definition.
**Definition 2.2.12** Let $|A|_p$ be an nxs matrix with entries from $G(p)$, p-prime, then

\[
\text{rank}(|A|_p) = \text{dimension of } L(|A|_p) \\
= \text{dimension of } L(|A|'_p) \\
= \text{maximum number of linearly independent columns of } |A|_p, \text{ linearly independent over } G(p) \\
= \text{maximum number of linearly independent rows of } |A|_p, \text{ linearly independent over } G(p).
\]

It is important that the notation rank($A$) and rank($|A|_p$) is clearly understood. If $A$ is a real matrix ($A$ integral or $A$ rational falls into this category) then the rank($A$) is the maximum number linearly independent columns (or rows) in $A$, linearly independent over the field of real numbers. If $|A|_p$, p-prime, is the residue of $A$ modulo $p$, then the rank($|A|_p$) is the maximum number of linearly independent columns (or rows) in $|A|_p$, linearly independent over $G(p)$. If $A$ is an nxs integral matrix with rank($A$) $\leq n-1$, then $\det(A) = 0$ and $A$ is singular. Also, if rank($|A|_p$) $\leq n-1$, then $|\det(|A|_p)|_p = 0$ and $|A|_p$ is singular over $G(p)$ or $A$ is singular modulo $p$. It should be obvious that the rank($A$) does not necessarily equal the rank($|A|_p$).

**Example 2.2.1** Let $A = \begin{bmatrix} 20 & 15 \\ 15 & 18 \end{bmatrix}$, rank($A$) = 2.

For $p = 3$, $|A|_3 = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}$ and rank($A_3$) = 1.
For \( p = 5 \), \( |A|_5 = \begin{bmatrix} 0 & 0 \\ 0 & 3 \end{bmatrix} \) and \( \text{rank}(|A|_5) = 1 \).

For \( p = 7 \), \( |A|_7 = \begin{bmatrix} 6 & 1 \\ 1 & 4 \end{bmatrix} \) and \( \text{rank}(|A|_7) = 2 \).

Throughout this section we have been giving the results in the single modulus residue number system framework. These results extend easily into the multiple modulus residue number systems.

**Definition 2.2.13** If \( X \) is a \( pxq \) integral matrix, then \( X \) has a multiple modulus residue representation for the base \( \{m_1, m_2, \ldots, m_s\} \), \((m_i, m_j) = 1 \) for \( i \neq j \) and \( m_i > 1 \) for all \( i = 1, 2, \ldots, s \), if and only if \( |X|_{m_i} \) exists for all \( i = 1, 2, \ldots, s \). In this case

\[
X = (x_{ij}) \sim \{|x_{ij}|_{m_1}, |x_{ij}|_{m_2}, \ldots, |x_{ij}|_{m_s}\}.
\]

The symmetric multiple modulus residue representation would be

\[
X = (x_{ij}) \sim \{|x_{ij}/m_1, x_{ij}/m_2, \ldots, x_{ij}/m_s\}.
\]

(Note that if \( |X|_m \) exists if and only if \( /X/_{m} \) exists.)
Extreme care needs to be taken if \( X^{-1}(M) \) is to be expressed for the base \( \{m_1, m_2, \ldots, m_s\}, (m_i, m_j) = 1 \) for \( i \neq j \), \( m_i > 1 \) for all \( i = 1, 2, \ldots, s \), and \( M = \prod_{i=1}^{s} m_i \). This has a multiple modulus residue representation if and only if \( X \) is nonsingular modulo \( m_i \) for all \( m_i \) in the base. The adjoint matrix, on the other hand, always has a multiple modulus residue representation for any base.

We will be introducing and developing more matrix theory for residue arithmetic throughout this research. This section contains the foundation for the matrix theory necessary to solve the error-free nonsingular matrix inversion problem.

### 2.3 Error-Free Nonsingular Matrix Inversion

In residue arithmetic, as we have seen, we work only with integers so computations can be performed exactly, even in a digital computer. In the past work done on this topic, the restriction that the matrix \( A \) be integral was never considered to be a serious problem since if \( A \) is rational it can be converted to integers by scaling. We will address this problem, but not until Chapter 6. So, for the time being we too will assume \( A \) is integral. The problems to be solved in this section are how to find \( |\det(A)|_M \) and \( A^{\text{adj}}(M) \). From Section 2.2, we know for any integer \( M > 1 \) these both exist. Next, once we have \( |\det(A)|_M \) and \( A^{\text{adj}}(M) \) we will see how to use them to find \( \det(A) \) and \( A^{\text{adj}} \).
Suppose a base of prime moduli, \( \{p_1, p_2, \ldots, p_s\} \), \( M = \prod_{i=1}^{s} p_i \), is chosen such that for an \( n \times n \) nonsingular integral matrix \( A \)

\[
\frac{-M}{2} < \det(A) < \frac{M}{2}
\]

and

\[
\frac{-M}{2} < (A_{ij}) < \frac{M}{2}
\]

for all the cofactors of \( A \). According to Corollary 2.1.3.1, the symmetric multiple modulus residue representations for the integers in the interval \( \left[ \frac{-M}{2}, \frac{M}{2} \right] \) are unique. If the multiple modulus residue representations for the determinant of \( A \) and all the cofactors were known, the Chinese Remainder Theorem could be used to generate \( \det(A) \mod M \) and \( A_{ij} \mod M \). Then the symmetric residue modulo \( M \) can be formed and in fact

\[
\frac{\det(A)}{M} = \det(A)
\]

and

\[
\frac{(A_{ij})}{M} = (A_{ij})
\]

Newman (1967) pointed out this result. What is actually done is that \( M = \prod_{i=1}^{s} p_i \) is found such that

\[
M > 2|\det(A)|
\]

and

\[
M > 2|A_{ij}| \quad \text{for all } i \text{ and } j.
\]

Newman (1967), Howell and Gregory (1969a,b), and Cabay and Lam (1977a,b) used Hadamard's inequality to bound the determinant of \( A \) and its' cofactors. So if
\[ M > 2 \prod_{i=1}^{n} \prod_{j=1}^{n} a_{i,j}^{2} \]
\[ > 2 \max(\{|\text{det}(A)|, \max |A_{ij}|\}) \]

then the base \( \{p_1, p_2, ..., p_s\} \), where the product modulus \( M = \prod_{i=1}^{s} p_i \), is chosen to satisfy this equation.

Once the base has been chosen, the multiple modulus residue representation of \( \text{det}(A) \) and \( A^{\text{adj}} \) need to be computed. Howell and Gregory (1969a,b) found these quantities by performing Gauss-Jordan elimination on \( A \) using multiple modulus residue arithmetic.

To invert an \( nxn \) matrix \( A \), using Gauss-Jordan elimination, the process is as follows. First, the \( nxn \) identity \( I \) is adjoined to \( A \) to form an \( nx2n \) matrix, \( (A:I) \). Elementary row operations (multiplication of a row by a nonzero constant and addition of a multiple of one row to another), and row interchanges are performed on \( (A:I) \) to bring \( A \) to upper triangular form with ones on the diagonal. We will call this the forward course. Then using only elementary row operations, the transformation of \( A \) to the identity is completed. We will call this the return course. The Gauss-Jordan process is denoted as

\[ (A:I) \rightarrow (I:A^{-1}) \]

In the forward course, to get the ones on the diagonal, each row must be multiplied by the reciprocal of its diagonal element. These diagonal elements are called the pivots. If \( r \) is the number of row interchanges then
\[ \det(A) = (-1)^r \prod_{i=1}^{n} x_i, \text{where } x_i = i^{th} \text{ pivot,} \]

and

\[ A^{\text{adj}} = \det(A) A^{-1}. \]

Applying residue arithmetic to the Gauss-Jordan process does not present a problem provided \( A \) is nonsingular modulo \( p \), \( p \)-prime. Consider the following result.

**Theorem 2.3.1** If an \( n \times n \) integral matrix \( A \) is nonsingular modulo \( p \), \( p \)-prime, then there exists an inverse modulo \( p \) for every pivot element of \( A \).

**Proof:** Recall that \( x^{-1}(p) \) exists if and only if both

i) \( |x|_p \neq 0 \)

ii) \((x, p) = 1\).  

Now if \( A \) is nonsingular modulo \( p \) then \( p \nmid \det(A) \),

\[ \det(A) = (-1)^r \prod_{i=1}^{n} x_i, \text{where } x_i = i^{th} \text{ pivot.} \]

Therefore, \( p \nmid x_i \) and \((x_i, p) = 1\) for all \( i = 1, 2, \ldots, n \).

Now \( |\det(A)|_p \neq 0 \) therefore \( |x_i|_p \neq 0 \) for all \( i = 1, 2, \ldots, n \).

Consequently, provided \( A \) is nonsingular modulo \( p \) each pivot in the Gauss-Jordan procedure can be performed using residue arithmetic. What
Howell and Gregory did was to only allow those prime moduli in their base such that \( A \) was nonsingular modulo \( p \). If for a given modulus \( p \) a zero pivot was encountered, those results were discarded and that modulus was replaced with a new prime. This hunt and peck type procedure was continued until \( |A_{\text{adj}}|_{p_i} \) and \( |\det(A)|_{p_i} \) had been computed for enough \( p_i \) such that the product modulus \( M = \Pi p_i \) satisfied Equation 2.3.1. These multiple modulus residue representations were then combined, via the Chinese Remainder Theorem, yielding \( \det(A) \) and \( A_{\text{adj}} \).

Cabay and Lam (1977a,b) used extended Gauss-Jordan elimination as defined by Shapiro (1963). Extended Gauss-Jordan elimination is Gauss-Jordan elimination with one additional elementary row operation. This new row operation is as follows. When a row of zeros, say the \( i^{\text{th}} \) row, is encountered in the transform of \( A \), the diagonal element of that row is changed to 1, and in the augmented portion of the matrix all other rows are zeroed out, leaving the \( i^{\text{th}} \) row unchanged. In this situation

\[
\begin{align*}
(A:I) & \rightarrow (I:S), \\
\det(A) & = 0
\end{align*}
\]

and

\[
A_{\text{adj}} = \left\{(-1)^{\mathbf{r}} \prod_{i=1}^{n} b_i\right\} \cdot S
\]

where

\[
b_i = \begin{cases} 
\text{\(i^{\text{th}}\) pivot, if \(i^{\text{th}}\) pivot \neq 0} \\
1, \text{ otherwise}
\end{cases}
\]

(Note that if \( \text{rank}(A) = n-1 \), \( \det(A) = 0 \) but \( A_{\text{adj}} \) is not necessarily null, if
rank(A) < n-1 then $A^{adj} = \phi$. Cabay and Lam (1977a,b) applied residue arithmetic to the extended Gauss-Jordan elimination process. However, if for a given prime modulus, p, it was determined that rank($|A|_p$) < n-1 this modulus was removed from the base. They too would continue to try new prime moduli, $p_i$, until enough usable results were accumulated such that the product modulus $M = \prod p_i$ satisfied Equation 2.3.1.

The undesirable features of these methods are that usable results are being discarded and not until the problem is completed do we know what our base is. Both of these procedures do, however, provide an adequate solution to the error-free nonsingular matrix inversion problem.

2.4 Alternative Solution to the Error-Free Nonsingular Matrix Inversion Problem

In Section 2.3, we saw that a fair amount of work could be wasted in trying to find the multiple modulus residue representation of $A^{adj}$ and $\det(A)$ for an nxn nonsingular integral matrix $A$. In the case of Howell and Gregory (1969a,b), if

$$\text{rank}(|A|_p) < \text{rank}(A),$$

that prime modulus $p$ was discarded and a new prime was added to the base. Cabay and Lam (1971) came closer to eliminating this problem by using extended elimination. But, there was still confusion as to how to handle those moduli, $p$, for which $|A|_p$ is rank deficient of two or more.
In both of these methods, the subtle difference discussed in the previous sections between arithmetic and matrix operations over the Galois field generated by \( p \) and those defined by residue arithmetic can be made. Howell and Gregory (1969a,b) strictly applied residue arithmetic to the Gauss-Jordan elimination process. Knowing that if \( A \) is singular modulo \( p \), \( A^{-1}(p) \) is not defined, they did not use those moduli. Cabay and Lam (1971) realized that \( |A^{\text{adj}}|_p = A^{\text{adj}}(p) \) exists over \( G(p) \) even when \( A \) is singular modulo \( p \). Or, at least they thought \( A^{\text{adj}}(p) \) existed for those moduli, \( p \), such that \( \text{rank}(|A|_p) \geq n-1 \). There was confusion about the case when \( \text{rank}(|A|_p) < n-1 \) and \( A^{\text{adj}}(p) = \emptyset \). Their solution was to replace these moduli with new primes. These people knew that in order to find \( A^{\text{adj}} \) and the \( \det(A) \) from a multiple modulus residue representation that for some of the moduli in the base \( |A|_p \) must be full-rank. What it seems they were not sure of was the answer to the following question. Given a base \( \{p_1, p_2, \ldots, p_s\} \) such that \( \prod_{i=1}^{s} p_i \) satisfies Equation 2.3.1 is it possible for \( \text{rank}(|A|_{p_i}) < \text{rank}(A) \) for all \( i = 1, 2, \ldots, s \)? The next theorem answers this question.

**Theorem 2.4.1** Let \( A \) be an \( nxn \) nonsingular integral matrix. If the base \( \{p_1, p_2, \ldots, p_s\} \), \( p_i \)-prime and \( M = \prod_{i=1}^{s} p_i \), is chosen such that

\[
M > 2|\det(A)|
\]

then there exists at least one \( p_i \) such that

\[
\text{rank}(|A|_{p_i}) = \text{rank}(A) = n.
\]
Proof: Suppose $p_i | \det(A)$ for all $i = 1, 2, \ldots, s$. Then $M | \det(A)$ and $M | | \det(A)|$. Then there exists $k \in \mathbb{Z}^+$ such

$$| \det(A)| = k \cdot M.$$ 

This is a contradiction since $M > 2 | \det(A)|$. Hence, there exists at least one $p_i$, say $p^*$, such that

$$p^* \nmid \det(A).$$

Since $p^*$ is prime, $(\det(A), p^*) = 1$. Therefore, $A$ is nonsingular modulo $p^*$ or $\text{rank}(|A|^*) = n = \text{rank}(A)$.

This result tells us that a base $\{p_1, p_2, \ldots, p_s\}$ such that $\prod_{i=1}^s p_i$ satisfies Equation 2.3.1 can be chosen at the start of the problem and the multiple modulus residue representations of $A^\text{adj}$ and $\det(A)$ with respect to this base can be used to solve the problem. No elements of the base need to be replaced or results discarded. We end this chapter with a simple example that illustrates the error-free nonsingular matrix inversion process.

Example 2.4.1

Let $A = \begin{bmatrix} 6 & 3 \\ 3 & 9 \end{bmatrix}$,

$$M = 3 \cdot 5 \cdot 7 = 105 > 2 \max \{ |\det(A)|, \max_{i,j} |A_{i,j}| \}.$$ 

So we will use the base $\{3, 5, 7\}$. 

\[ |A|_3 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \text{ rank}(|A|_3) = 0 \]

So, \( A^\text{adj} (3) = \emptyset \) and \( |\det(|A|_3)|_3 = 0 \).

\[ |A|_5 = \begin{bmatrix} 1 & 3 \\ 3 & 4 \end{bmatrix}, \text{ rank}|A|_5 = 1 \]

Apply extended Gauss-Jordan elimination modulo 5:
\[
\begin{bmatrix}
1 & 3 & 1 & 0 \\
3 & 4 & 0 & 1
\end{bmatrix} \rightarrow 
\begin{bmatrix}
1 & 3 & 1 & 0 \\
0 & 0 & 2 & 1
\end{bmatrix} \{2x(\text{row 1}) + \text{row 2}_5\} \\
\rightarrow 
\begin{bmatrix}
1 & 3 & 0 & 0 \\
0 & 1 & 2 & 1
\end{bmatrix} \{\text{extended elimination}\} \\
\rightarrow 
\begin{bmatrix}
1 & 0 & 4 & 2 \\
0 & 1 & 2 & 1
\end{bmatrix} \{2x(\text{row 2}) + \text{row 1}_5\}
\]

\[ |\Pi (\text{nonzero pivots})|_5 = 1 \]

So, \( A^\text{adj} (5) = 1 \cdot \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix} \]

and \( |\det(|A|_5)|_5 = 0 \).

\[ |A|_7 = \begin{bmatrix} 6 & 3 \\ 3 & 2 \end{bmatrix}, \text{ rank}|A|_7 = 2 \]

Apply Gauss-Jordan elimination modulo 7.
\[
\begin{bmatrix}
6 & 3 & 1 & 0 \\
3 & 2 & 0 & 1 \\
\end{bmatrix} \rightarrow \begin{bmatrix}
1 & 4 & 6 & 0 \\
3 & 2 & 0 & 1 \\
\end{bmatrix} \{ |6^{-1}(7) \times \text{(row 1)}|_7 \}
\]

\[
\begin{bmatrix}
1 & 4 & 6 & 0 \\
0 & 4 & 3 & 1 \\
\end{bmatrix} \{ |4 \times \text{(row 1)} + \text{row 2}|_7 \}
\]

\[
\begin{bmatrix}
1 & 4 & 6 & 0 \\
0 & 1 & 6 & 2 \\
\end{bmatrix} \{ |4^{-1}(7) \times \text{(row 2)}|_7 \}
\]

\[
\begin{bmatrix}
1 & 0 & 3 & 6 \\
0 & 1 & 6 & 2 \\
\end{bmatrix} \{ |3 \times \text{(row 2)} + \text{row 1}|_7 \}
\]

\[
A^{-1}(7) = \begin{bmatrix}
3 & 6 \\
6 & 2 \\
\end{bmatrix},
\]

\[
|\det(A)|_7 = |6 \cdot 4|_7 = 3, \]

and

\[
A^{\text{adj}}(7) = 3 \cdot \begin{bmatrix}
3 & 6 \\
6 & 2 \\
\end{bmatrix} |_7
\]

\[
= \begin{bmatrix}
2 & 4 \\
4 & 6 \\
\end{bmatrix}.
\]

So,

\[
\det(A)_{105} \sim \{0, 0, 3\}
\]

and

\[
A^{\text{adj}}_{105} \sim \left\{ \begin{bmatrix}
0 & 0 \\
0 & 0 \\
\end{bmatrix}, \begin{bmatrix}
4 & 2 \\
2 & 1 \\
\end{bmatrix}, \begin{bmatrix}
2 & 4 \\
4 & 6 \\
\end{bmatrix} \right\}.
\]
Chinese Remainder Theorem:

\[ M = 105 \]
\[ \hat{m}_1 = 35 \quad , \quad \hat{m}_1^{-1}(3) = 2 \]
\[ \hat{m}_2 = 21 \quad , \quad \hat{m}_2^{-1}(5) = 1 \]
\[ \hat{m}_3 = 15 \quad , \quad \hat{m}_3^{-1}(7) = 1 \]

\[ |\text{det}(A)|_{105} = \begin{vmatrix} 0 & 0 & 15 \\ 3 & 1 & 7 \\ 105 & 0 & 0 \end{vmatrix}_{105} = 45 \]

\[ |A^{\text{adj}}|_{105} = |A^{\text{adj}}|_{105} = \begin{vmatrix} 0 & 0 \\ 2 & 1 \\ 105 & 0 \end{vmatrix} + 21 \begin{vmatrix} 4 & 2 \\ 1 & 1 \\ 105 & 0 \end{vmatrix} + 15 \begin{vmatrix} 2 & 4 \\ 4 & 6 \\ 105 & 0 \end{vmatrix} \]

\[ = \begin{vmatrix} 84 & 42 \\ 42 & 21 \end{vmatrix} + \begin{vmatrix} 30 & 60 \\ 60 & 90 \end{vmatrix}_{105} \]

\[ = \begin{vmatrix} 114 & 102 \\ 102 & 111 \end{vmatrix}_{105} \]

\[ = \begin{vmatrix} 9 & 102 \\ 102 & 6 \end{vmatrix} \]

and

\[ \text{det}(A) = |\text{det}(A)|_{105} = 45 \]

\[ A^{\text{adj}} = /A^{\text{adj}}/_{105} = \begin{bmatrix} 9 & -3 \\ -3 & 6 \end{bmatrix} \]

Therefore,

\[ A^{-1} = \frac{1}{45} \begin{bmatrix} 9 & -3 \\ -3 & 6 \end{bmatrix} . \]
It should be noted that Howell and Gregory's method would have replaced the moduli 3 and 5 and Cabay and Lam (1977a,b) would have replaced the modulus 3 in the base.
Given an arbitrary nxs integral matrix A it is possible to compute an error-free reflexive generalized inverse for A (over the field of rational numbers) using multiple modulus residue arithmetic. The matrix $A^-$ is the reflexive generalized inverse of A if both $AA^-A = A$ and $A^-AA = A^-$. In Chapter 1, some of the previous work on generating an error-free $A^-$ over the field of rational numbers using single modulus residue arithmetic was cited. As was indicated there, these methods either required knowledge of $\text{rank}(A)$, $\text{rank}(|A|^p)$ and $\text{rank}(|AA'|_p)$ (Sen and Shamin (1978) and Stallings and Boullion 1979)) or required working with complicated forms like $A'(AA'AA')^-AA'$ (Rao et al. (1976).) The problem to be solved in this chapter is as follows. Given an arbitrary nxs integral matrix A, find $A^-$ in a straightforward manner without the need for additional information about A, like the $\text{rank}(A)$, at the onset of the problem.

Section 3.1 will approach the generalized inverse problem from a field theory point of view. Section 3.2 will consider the situation of applying residue arithmetic to a well-established generalized inverse process. In Section 3.3, the choice of the product modulus bound for this problem will be discussed. Section 3.4 gives two integral methods for finding $A^-$ using multiple modulus residue arithmetic.
3.1 Existence of Generalized Inverses Over Various Fields and Rings

This section deals with the existence of generalized inverses over the Galois field generated by a prime \( p \) and over the commutative ring generated by the product modulus \( M = \prod_{i=1}^{s} p_i \), \( p_i \)-prime. Pearl (1968) discusses the existence of generalized inverses over arbitrary fields. Since the field of interest is known, a specific form of the generalized inverse over this field can be characterized. Given an arbitrary \( n \times s \) integral matrix \( A \), the generalized inverse of \( |A|_p \), \( p \)-prime, will be characterized and its usefulness in finding the generalized inverse of \( A \) over the field of rational numbers will be discussed.

In Section 2.2, the Galois field generated by a prime \( p \), \( G(p) \), and the commutative ring generated by the product modulus \( M = \prod_{i=1}^{s} p_i \), \( p_i \)-prime, \( R(M) \), were introduced. One of the basic differences between the commutative ring and the Galois field with the operations + and \( \cdot \) as defined in Theorem 2.1.4 is that every nonzero element in the Galois field has a multiplicative inverse in the field. The only elements in the ring with multiplicative inverses are those integers relatively prime with \( M \). Another field property that will be used here is if \( x \) and \( y \) are elements of the field then \( x \cdot y = 0 \) if and only if either \( x = 0 \) or \( y = 0 \) (a field has no zero divisors). These two field properties make it possible to discuss in detail linear independence with respect to a set of distinct vectors from the Galois field generated by the prime \( p \), \( G(p) \).
Consider the n×s matrix $|A|_p$, p-prime, with entries over $G(p)$. Suppose

$$\text{rank}(|A|_p) = r,$$

then there exists permutation matrices $P$ and $Q$ such that

$$P|A|_p Q = \begin{bmatrix} |A_{11}|_p & |A_{12}|_p \\ |A_{21}|_p & |A_{22}|_p \end{bmatrix}$$

where

$$\text{rank}(|A_{11}|_p) = \text{order} |A_{11}|_p = r.$$

All of the entries in $P$ and $Q$ are either zero or one. Thus, $|P|A|_p Q|_p = P|A|_p Q$

and

$$|A|_p = P' \begin{bmatrix} |A_{11}|_p & |A_{12}|_p \\ |A_{21}|_p & |A_{22}|_p \end{bmatrix} Q'.$$

Without loss of generality let

$$|A|_p = \begin{bmatrix} |A_{11}|_p & |A_{12}|_p \\ |A_{21}|_p & |A_{22}|_p \end{bmatrix}$$

where

$$\text{rank}(|A|_p) = \text{rank}(|A_{11}|_p) = \text{order} |A_{11}|_p = r.$$

The submatrix $|A_{11}|_p$ is of full-rank over $G(p)$, so the columns of

$$\begin{bmatrix} |A_{11}|_p \\ |A_{21}|_p \end{bmatrix}$$

are linearly independent. Since the rank($|A_{11}|_p$) = rank($|A|_p$),
the columns of \( \begin{bmatrix} |A_{11}|_p \\ |A_{21}|_p \end{bmatrix} \) must be linearly dependent with the columns of \( \begin{bmatrix} |A_{12}|_p \\ |A_{22}|_p \end{bmatrix} \). Therefore, there exists nonnull matrices \( L, (r) \times (s-r) \), and \( K, (s-r) \times (s-r) \) with entries over the field such that

\[
\begin{bmatrix} |A_{11}|_p \\ |A_{21}|_p \end{bmatrix} L + \begin{bmatrix} |A_{12}|_p \\ |A_{22}|_p \end{bmatrix} K = \phi \begin{pmatrix} n \times (s-r) \end{pmatrix}.
\]

(3.1.1)

The matrix \( K \) is diagonal because each column in \( \begin{bmatrix} |A_{12}|_p \\ |A_{22}|_p \end{bmatrix} \) is linearly dependent with the set of columns \( \begin{bmatrix} |A_{11}|_p \\ |A_{21}|_p \end{bmatrix} \).

**Theorem 3.1.1** Let \( |A|_p \), p-prime, be an nxs matrix with entries over \( G(p) \). Assume

\[
|A|_p = \begin{bmatrix} |A_{11}|_p & |A_{12}|_p \\ |A_{21}|_p & |A_{22}|_p \end{bmatrix}
\]

where

\[
\text{rank}(|A_{11}|_p) = \text{order} |A_{11}|_p = \text{rank}(|A|_p) = r.
\]

Then there exists an \( (r) \times (s-r) \) matrix \( T \) over \( G(p) \) such that

\[
\begin{bmatrix} |A_{12}|_p \\ |A_{22}|_p \end{bmatrix} = \begin{bmatrix} |A_{11}|_p \\ |A_{21}|_p \end{bmatrix} T.
\]
Proof: Let \( \alpha_1, \alpha_2, \ldots, \alpha_r \) be the columns of \( \begin{bmatrix} |A_{11}|_p \\ |A_{12}|_p \end{bmatrix} \) and \( \beta_1, \beta_2, \ldots, \beta_r \) be the columns of \( \begin{bmatrix} |A_{12}|_p \\ |A_{22}|_p \end{bmatrix} \). The vectors \( \alpha_1, \alpha_2, \ldots, \alpha_r \) are linearly independent over \( G(p) \); therefore, according to Definition 2.2.8 if the linear combination over \( G(p) \) of 
\[ a_1 \alpha_1 + a_2 \alpha_2 + \ldots + a_r \alpha_r \big|_p = \phi \]
where \( a_i \in \mathbb{Z} \) and \( |a_i|_p \in G(p) \), implies \( |a_i|_p = 0 \) for all \( i = 1, 2, \ldots, r \).

Let the \((r) \times (s-r)\) matrix \( L = (l_{ij}) \) and the \((s-r) \times (s-r)\) matrix \( K = \text{diag}(k_{jj}) \) be defined over \( G(p) \) as in Equation 3.1.1,
\[
\begin{bmatrix}
\sum_{i=1}^{r} a_{i1} l_{i1} \\
\sum_{i=1}^{r} a_{i2} l_{i2} \\
\vdots \\
\sum_{i=1}^{r} a_{i(s-r)} l_{i(s-r)}
\end{bmatrix}
\begin{bmatrix}
l_{11} \\
l_{22} \\
\vdots \\
l_{(s-r)(s-r)}
\end{bmatrix}
= \begin{bmatrix}
\phi \\
\phi \\
\vdots \\
\phi
\end{bmatrix}.
\]

Need to show \( |k_{jj}|_p \neq 0 \) for all \( j = 1, 2, \ldots, s-r \). Suppose \( |\beta_j k_{jj}|_p = \phi \) for some \( j \), then either \( \beta_j = \phi \) or \( |k_{jj}|_p = 0 \) or both (the field \( G(p) \) has no zero divisors.) If \( |k_{jj}|_p = 0 \) and \( \beta_j \neq \phi \) implies 
\[ \sum_{i=1}^{r} a_{i1} \beta_{ij} \big|_p = 0 \] which implies \( |l_{ij}|_p = 0 \) for all
i = 1, 2, ..., r. This means $\beta_j$ is another linearly independent column over $G(p)$, a contradiction. Consequently, if $\beta_j \neq \phi$ for some $j$, $|k_{jj}|_p \neq 0$. If $\beta_j = \phi$, for some $j$, then $|\beta_j k_{jj}|_p = \phi$ for any value of $|k_{jj}|_p$. Without loss of generality let $|k_{jj}|_p = 1$ in this case. Therefore, $|k_{jj}|_p \neq 0$ for all $j = 1, 2, ..., s-r$, and $k_{jj}^{-1}(p)$ exists for all $j = 1, 2, ..., s-r$, and $K^{-1}(p) = \text{diag}(k_{jj}^{-1}(p))$.

Then the equations become

$$
\begin{bmatrix}
\begin{array}{r}
\sum_{i=1}^{r} a_{i1} k_{11}^{-1}(p) + \beta_1
\\
\sum_{i=1}^{r} a_{i2} k_{22}^{-1}(p) + \beta_2
\\
\vdots
\\
\sum_{i=1}^{r} a_{is-r} k_{(s-r)(s-r)}^{-1}(p) + \beta_{s-r}
\end{array}
\end{bmatrix}
= \phi
$$

and are equivalent to

$$
\begin{bmatrix}
\begin{array}{r}
|A_{11}|_p
\\
|A_{21}|_p
\\
|A_{12}|_p
\\
|A_{22}|_p
\end{array}
\end{bmatrix} L \cdot K^{-1}(p) + \begin{bmatrix}
\begin{array}{r}
|A_{12}|_p
\\
|A_{22}|_p
\end{array}
\end{bmatrix}
= \phi
.$$
The generalized inverse of $|A_p|$, p-prime, over the Galois field generated by $p$ can now be defined.

**Definition 3.1.1** Let $|A_p|$, p-prime, be an $n \times s$ matrix with entries over $G(p)$. Then $A^{-}(p)$ is a generalized inverse of $|A_p|$ over $G(p)$ if
\[
\]

**Proof of existence:** Suppose $\text{rank}(|A_p|) = r$, then there exists permutation matrices $P$ and $Q$ such that
\[
P|A_p|Q = \begin{bmatrix}
|A_{11}|_p & |A_{12}|_p \\
|A_{21}|_p & |A_{22}|_p
\end{bmatrix}
\]
where
\[
\text{rank}(|A_{11}|_p) = \text{order} |A_{11}|_p = r.
\]
From Theorem 3.1.1, there exists a $(r) \times (s-r)$ matrix $T$ over $G(p)$ such that
\[
\begin{bmatrix}
|A_{11}|_p \\
|A_{21}|_p
\end{bmatrix} T_p = \begin{bmatrix}
|A_{12}|_p \\
|A_{22}|_p
\end{bmatrix}.
\]
Then,
\[
|A_{11}|_p T_p = |A_{12}|_p
\]
and
\[
|A_{11}^{-}(p) |A_{11}|_p T_p = |A_{12}^{-}(p) |A_{12}|_p p
\]
Therefore,

\[ T = |T|_p = \begin{bmatrix} \mathcal{A}_{11}^{-1}(p) & \mathcal{A}_{12} \end{bmatrix}_p \]

and

\[ |\mathcal{A}_{22}|_p = \begin{bmatrix} |\mathcal{A}_{21}|_p T \end{bmatrix}_p \]

\[ = \begin{bmatrix} |\mathcal{A}_{21}|_p \mathcal{A}_{11}^{-1}(p) \mathcal{A}_{12} \end{bmatrix}_p. \]

Let

\[ \mathcal{A}^{-}(p) = Q \begin{bmatrix} \mathcal{A}_{11}^{-1}(p) & \phi \\ \phi & \phi \end{bmatrix}_p. \]  \hspace{1cm} (3.1.2)

Then,

\[ \begin{bmatrix} |\mathcal{A}|_p \mathcal{A}^{-}(p) |\mathcal{A}|_p \end{bmatrix}_p = \mathcal{A}(p). \]

Also, note that \( \mathcal{A}^{-}(p) \) satisfies the equation

\[ \begin{bmatrix} \mathcal{A}^{-}(p) |\mathcal{A}|_p \mathcal{A}^{-}(p) \end{bmatrix}_p = \mathcal{A}^{-}(p). \]

Therefore, \( \mathcal{A}^{-}(p) \) expressed as in Equation 3.1.2, is a reflexive generalized inverse which is defined as follows.

**Definition 3.1.2**  
Let \( |\mathcal{A}|_p \), p-prime, be an nxs matrix with entries over \( G(p) \). Then \( \mathcal{A}^{-}(p) \) is a reflexive generalized inverse of \( |\mathcal{A}|_p \) over \( G(p) \) if

\[ \begin{bmatrix} |\mathcal{A}|_p \mathcal{A}^{-}(p) |\mathcal{A}|_p \end{bmatrix}_p = |\mathcal{A}|_p \]

and

\[ \begin{bmatrix} \mathcal{A}^{-}(p) |\mathcal{A}|_p \mathcal{A}^{-}(p) \end{bmatrix}_p = \mathcal{A}^{-}(p). \]
The existence of a reflexive generalized inverse of $|A|^p$ over $G(p)$ has already been shown in the existence proof for Definition 3.1.1.

The proof of Theorem 3.1.1 and the existence proof for Definition 3.1.1, hinge on the fact that $p$ is prime. The following example illustrates what may happen if the generalized inverse of $|A|_M$, $M$-composite, is formed as in Equation 3.1.2. The existence of a generalized inverse of $|A|_M$ over the ring generated by $M$, $R(M)$, has not yet been given.

**Example 3.1.1**

Let $A = \begin{bmatrix} 5 & 1 \\ 1 & 3 \end{bmatrix}$ and $M = 77$.

$|A|_{77} = \begin{bmatrix} 5 & 1 \\ 1 & 3 \end{bmatrix}$ and $|\text{det}(|A|_{77})| = 14$.

$|A|_{77}$ is singular over $R(77)$ because $(14, 77) = 7$. Now, $\text{rank}(|A|_{77}) = 1$ and $|5|_{77}$ is a 1x1 minor of $|A|_{77}$ of rank one. Therefore, $P = Q = I$ and Equation 3.1.2 yields

$$I \cdot \begin{bmatrix} 5^{-1}(77) & 0 \\ 0 & 0 \end{bmatrix} \cdot I = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}.$$ 

$$|A|_{77} \begin{bmatrix} 31 & 0 \\ 0 & 0 \end{bmatrix} |A|_{77} = \begin{bmatrix} 5 & 1 \\ 1 & 31 \end{bmatrix} \leftrightarrow |A|_{77}$$

Therefore, $A^{-}(77) \leftrightarrow \begin{bmatrix} 31 & 0 \\ 0 & 0 \end{bmatrix}$. 
Part of the problem here is that \( R(77) \) has zero divisors which means Theorem 3.1.1 does not hold. Consider

\[
\begin{bmatrix} 5 \\ 1 \end{bmatrix} \cdot 11 + \begin{bmatrix} 1 \\ 3 \end{bmatrix} \cdot 22 \equiv \begin{bmatrix} 0 \end{bmatrix} \pmod{77},
\]

neither 11 or 22 have inverses over \( R(77) \), so \( \begin{bmatrix} 1 \\ 3 \end{bmatrix} \) cannot be expressed as a multiple of \( \begin{bmatrix} 5 \\ 1 \end{bmatrix} \).

The next theorem determines when \( A^{-}(M) \) exists over the ring generated by \( M = \prod_{j=1}^{s} p_j \), \( p_j \)-prime.

**Theorem 3.1.2** Given generalized inverses \( A^{-}(p_j) \), \( p_j \)-prime, \( i = 1, 2, \ldots, s \) and \( M = \prod_{i=1}^{s} p_i \). Then a generalized inverse of \( |A|_M, A^{-}(M) \), over \( R(M) \) is produced by applying the Chinese Remainder Theorem to the \( A^{-}(p_j) \), \( i = 1, 2, \ldots, s \).

**Proof:** For each \( i = 1, 2, \ldots, s \)

\[
\left| \begin{bmatrix} A \\ p_i \end{bmatrix} A^{-}(p_i) \begin{bmatrix} A \\ p_i \end{bmatrix} \right| = \left| A \right|_{p_i}.
\]

Since \( A^{-}(M) \) is the result of combining \( A^{-}(p_1), A^{-}(p_2), \ldots, A^{-}(p_s) \) via the Chinese Remainder Theorem,

\[
\left| A^{-}(M) \right|_{p_i} = A^{-}(p_i) \text{ for all } i.
\]

Need to show

\[
\left| \begin{bmatrix} A \\ M \end{bmatrix} A^{-}(M) \begin{bmatrix} A \\ M \end{bmatrix} \right| = \left| A \right|_M.
\]
It suffices to show

\[ \left| \left| A \right|_M A^{-M} \right|_{p_i} = \left| A \right|_{p_i} \text{ for all } i. \]

To show \( \left| A \right|_{p_i} = \left| A \right|_{p_1} \). Let \( A = (a_{st}) \). If \( a_{st} \equiv x \) mod \( M \), then

\[ a_{st} \equiv x \text{ mod } M \text{ and } a_{st} - x = k \cdot M \text{ for some integer } k. \]

Then,

\[ x = a - k \cdot M \]

\[ = a_{st} - k \left( \prod_{j=1, j \neq i}^{s} p_j \right) \]

and

\[ \left| x \right|_{p_i} = \left| a_{st} \right|_{p_i} \]

\[ = \left| a_{st} - k \left( \prod_{j=1, j \neq i}^{s} p_j \right) \right|_{p_i} \]

\[ = \left| a_{st} \right|_{p_i}, \text{ since } \left| k \left( \prod_{j=1, j \neq i}^{s} p_j \right) \right|_{p_i} = 0. \]

Therefore,

\[ \left| A \right|_{p_i} = \left| A \right|_{p_1} \]

and

\[ \left| \left| A \right|_M A^{-M} \right|_{p_i} = \left| \left| A \right|_M \right|_{p_i} \left| A^{-M} \right|_{p_i} \]

\[ = \left| A \right|_{p_1} A^{-M} (p_1) \left| \left| A \right|_{p_1} \right|_{p_1} \]
Corollary 3.1.1.1 \hspace{1em} Let $A^-(p_i)$, $p_i$-prime, $i = 1, 2, \ldots, s$ and $M = \prod_{i=1}^{s} p_i$, if $A^-(p_i)$ is a reflexive generalized inverse of $|A|_{p_i}$ over $G(p_i)$ for all $i = 1, 2, \ldots, s$ then $A^-(M)$ is a reflexive inverse of $|A|_M$ over $R(M)$.

Proof: \hspace{1em} For all $i$

$$
|A^-(M) |A|_M A^-(M)|_M = |A^-(M)|_{p_i} |A|_{M_{p_i}} |A^-(M)|_{p_i}^{-1}_{p_i} \\
= |A^-(p_i) |A|_{p_i} A^-(p_i)|_{p_i}^{-1}_{p_i} \\
= A^-(p_i) \\
= |A^-(M)|_{p_i}^{-1}_{p_i} \hspace{1em} \Box
$$

It is now possible to rework Example 3.1.1 to find $A^-(77)$. 
Example 3.1.2 Let \( A = \begin{bmatrix} 5 & 1 \\ 1 & 3 \end{bmatrix} \) and let \( \{7, 11\} \) be the multiple modulus base.

\[
\text{rank}(|A|_7) = 1 \quad \text{and} \quad \text{rank}(|A|_{11}) = 2
\]

\[
A^{-1}(7) = \begin{bmatrix} 3 & 0 \\ 0 & 0 \end{bmatrix} \quad \text{and} \quad A^{-1}(11) = A^{-1}(11) = \begin{bmatrix} 1 & 7 \\ 7 & 9 \end{bmatrix}
\]

Chinese Remainder Theorem yields

\[
M = 77, \quad \hat{m}_1 = 11, \quad \hat{m}_1^{-1}(7) = 2
\]
\[
\hat{m}_2 = 7, \quad \hat{m}_2^{-1}(11) = 8
\]

and

\[
A^{-}(77) = \left| 11 \cdot 2 \cdot A^{-}(7) \right|_7 + 7 \cdot 8 \cdot A^{-}(11) \right|_{11} \left| 77
\]

\[
= \left| 11 \begin{bmatrix} 6 & 0 \\ 0 & 0 \end{bmatrix} + 7 \begin{bmatrix} 8 & 1 \\ 1 & 6 \end{bmatrix} \right|_{77}
\]

\[
= \begin{bmatrix} 45 & 7 \\ 7 & 42 \end{bmatrix}
\]

This is in fact a reflexive generalized inverse of \( |A|_{77} \) over \( \mathbb{R}(77) \).

It would seem, as in the nonsingular matrix inversion situation, that for a product modulus \( M \) large enough, the generalized inverse of \( A, A^{-} \), over the field of rational numbers could be determined from \( A^{-}(M) \). For
the nonsingular matrix inversion problem, the integral quantities 
det(A) and \( A^{\text{adj}} \) were derived from their unique symmetric multiple 
modulus residue representations in the interval \( (-\frac{M}{2}, \frac{M}{2}) \). The facts 
that \( |\det(A)|_p \), p-prime, is equivalent to the \( \det(|A|_p) \) over \( G(p) \) and 
\( |A^{\text{adj}}|_p \) is equivalent to \( A^{\text{adj}}(p) \), made finding the inverse 
of \( |A|_p \), \( A^{-1}(p) \), over \( G(p) \) a worthwhile approach to the nonsingular 
matrix inversion problem. Applying this same reasoning to the 
generalized inverse situation presents some new problems. First, \( A^{-1} \) 
must be expressed as 
\[
A^{-1} = \frac{1}{\text{cf}} A^{*},
\]
(3.1.3)
where \( \text{cf} \) is an integer and \( A^{*} \) is an integral matrix. If \( A \) is 
integral, \( A^{-1} \) will be a matrix with entries from the field of rational 
numbers. Consequently, every \( A^{-1} \) can be expressed as in Equation 
3.1.3. In the nonsingular matrix inversion problem, there is only one 
way to write 
\[
A^{-1} = \frac{1}{\text{cf}} A^{*}
\]
and that is 
\[
A^{-1} = \frac{1}{\det(A)} A^{\text{adj}}.
\]
(If \( A \) is integral the \( A^{\text{adj}} \) is integral.) Working with generalized 
inverses the uniqueness property of the inverse disappears. The 
next example helps show if given \( A^{-1}(p_1), A^{-1}(p_2), \ldots, A^{-1}(p_s) \) and
A^- (M = \Pi_{i=1}^{s} p_i), a unique symmetric multiple modulus residue representation in the interval \((-\frac{M}{2}, \frac{M}{2})\) for \(c\) (a common scale factor of \(A^-\)) and \(A^*\) cannot necessarily be found to form \(A^-\) as in Equation 3.1.3.

**Example 3.1.3**

Let \(A = \begin{bmatrix} 5 & 1 \\ 1 & 3 \end{bmatrix} \).

Consider \( |A|_7 = \begin{bmatrix} 5 & 1 \\ 1 & 3 \end{bmatrix} \) a possible generalized inverse of \( |A|_7 \) is \(A^-(7) = \begin{bmatrix} 3 & 0 \\ 0 & 0 \end{bmatrix} \) or \(A^-(7) = \begin{bmatrix} 0 & 0 \\ 0 & 5 \end{bmatrix} \).

Consider \( |A|_{11} = \begin{bmatrix} 5 & 1 \\ 1 & 3 \end{bmatrix} \)

\(A^-(11) = A^{-1}(11) = \begin{bmatrix} 1 & 7 \\ 7 & 9 \end{bmatrix} \).

Combining \( A^-(7) = \begin{bmatrix} 3 & 0 \\ 0 & 0 \end{bmatrix} \) and \( A^-(11) \) using the Chinese Remainder Theorem the result is
A^{-1}(77) = \begin{bmatrix} 45 & 7 \\ 7 & 42 \end{bmatrix}

= \begin{bmatrix} 3\cdot 3\cdot 5 & 7 \\ 7 & 2\cdot 3\cdot 7 \end{bmatrix}.

In this example, knowing the result

A = A^{-1} = \frac{1}{14} \begin{bmatrix} 3 & -1 \\ -1 & 5 \end{bmatrix},

we would expect \( M = 77 \) to be more than large enough so that \( A^{-1} \) can be extracted from the multiple modulus solution. However, given \( A^{-1}(77) \) the notion of common factor over \( R(77) \) is not clearly defined. It would appear as though \( A^{-1}(77) \) has no common factor. Suppose a different generalized inverse of \( |A|_p \) over \( G(7) \), \( A^{-1}(7) = \begin{bmatrix} 0 & 0 \\ 0 & 5 \end{bmatrix} \), had been combined with \( A^{-1}(11) \). The result would be

\[ A^{-1}(77) = \begin{bmatrix} 56 & 7 \\ 7 & 75 \end{bmatrix} \]

= \begin{bmatrix} 2\cdot 2\cdot 2\cdot 7 & 7 \\ 7 & 3\cdot 5\cdot 5 \end{bmatrix}.

We see that we now have two valid generalized inverses of \( |A|_{77} \) over \( R(77) \). In this example, \( A^{-1} = A^{-1} \) over the field of rational numbers is unique. From which of the \( A^{-1}(77) \) should \( A^{-1} \) over the field of rationals try to be extracted? We have no answer to this question.
This example indicates two major problems when trying to extract \( A^- \) from \( A^-(M) \). First, the concept of a common factor modulo \( M \) is not well-defined. Second, \( A^-(M) \) is not unique. There may be a way to use \( A^-(M) \) directly to generate \( A^- \) over the field of rational numbers but this sort of solution eludes us at the present time. If we restricted ourselves to only finding integral generalized inverses of integral matrices, then some of the problems of extracting \( A^- \) from \( A^-(M) \), mentioned in this section, may be able to be avoided. Hurt and Carter (1970), Batigne (1978), Batigne et al. (1978), and Erickson (1980) discuss the existence of generalized inverses over an integral domain. This is not the problem we choose to consider because given an arbitrary integral matrix there does not always exist an integral generalized inverse.

The approach to the multiple modulus generalized inverse problem given in this section seemed like the most logical path to take. It, however, points out the necessity to consider integral methods which generate specific forms of \( A^- \). The next few sections describe how residue arithmetic can be applied to some integral methods to generate an exact \( A^- \) over the field of rational numbers.

3.2 Multiple Modulus Residue Arithmetic
Generalized Matrix Inversion

The previous section illustrated the need to consider an integral method which will generate a specific form of \( A^- \), the generalized inverse of an arbitrary \( n \times n \) integral matrix \( A \). An integral method is one in which no division is performed unless the result of the division
is integral. The problem with most integral processes is that the size of the integers at various steps of computation get very large. However, if multiple modulus residue arithmetic is applied to an integral method, the size of the integers is controlled by the size of each individual modulus in the multiple modulus base. The only part of the process that needs to concern itself about how to handle large integers is the last step where the multiple modulus solution is combined via the Chinese Remainder Theorem or some other technique. The integral method used to generate $A^-$ in this section follows the logic used in Searle (1971) and many other texts on linear models for finding $A^-$ over the field of real numbers.

Once again it is assumed $A$ is an $n \times s$ integral matrix with rank($A$) = $r$. If $A$ is a rational matrix then $A$ can be scaled so that

$$A = \frac{1}{a} A^*$$

where $a$ is an integer and $A^*$ is integral, then

$$A^- = a \cdot A^*.$$  

Chapter 6 treats in detail the problem of $A$ having rational entries.

Given an $n \times s$ integral matrix $A$, there exists permutation matrices $P$ and $Q$ such that

$$PAQ = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$  \hspace{1cm} (3.2.1)

with

$$\text{rank}(A_{11}) = \text{order } A_{11} = r.$$  

The submatrix $A_{11}$ is the largest nonsingular minor of $A$. This means there are no $(r+1) \times (r+1)$ full-rank minors of $A$. Permutation matrices like $P$ and $Q$ can be found to place any $r \times r$ nonsingular minor into the
submatrix $A_{11}$. A reflexive generalized inverse for $A$ is

$$A^{-} = \frac{1}{\det(A_{11})} Q \begin{bmatrix} A_{11}^{\text{adj}} & \phi \\ \phi & \phi \end{bmatrix} P$$

(3.2.2)

In Section 3.1, it was determined that $A^{-}$ needed to have a specific form like

$$A^{-} = \frac{1}{\text{cf}} A^* ,$$

where $\text{cf}$ is an integer and $A^*$ is integral. Equation 3.2.2 satisfies this criterion.

The problem to be solved in this section is as follows. Given an arbitrary $n \times n$ integral matrix $A$, without any additional information about $A$, find permutation matrices $P$ and $Q$ to satisfy Equation 3.2.1 and then invert the $r \times r$ full-rank minor $A_{11}$. All of the above is to be accomplished using multiple modulus residue arithmetic. This could obviously be done without the use of multiple modulus residue arithmetic, but the size of the integers could become quite large.

**Example 3.2.1** Let $A = \text{diag}(2, 4, 8, 16, 32, 64, 128, 256)$ be an $8 \times 8$ integral matrix. In this example, $P = Q = I_8$ and $A_{11} = A$. Then, $\det(A_{11}) = 2^{36}$ and

$$A_{11}^{\text{adj}} = \text{diag}(2^{35}, 2^{34}, 2^{33}, 2^{32}, 2^{31}, 2^{30}, 2^{29}, 2^{28}) ,$$

If the integral method above was implemented in fixed-point arithmetic (integer arithmetic) on a computer like an IBM 370, whose manufacturers fixed-point word length is 32 bits, almost every element in the integral solution of this $A_{11}^{-1}$ would overflow this fixed-point word. Using 32 bits, only integers in the range $[-2^{31}, 2^{31}]$ are representable.
The biggest problem to be solved here is how to find the permutation matrices P and Q. Once the permutation matrices P and Q are found then the product PAQ of Equation 3.2.1 is formed to yield the \( r \times r \) nonsingular minor, \( A_{11} \), of \( A \). To invert \( A_{11} \) using multiple modulus residue arithmetic it is simply a matter of appealing to the nonsingular matrix inversion results found in Section 2.3 and 2.4.

Gaussian elimination can be applied to \( A \) to find P and Q. Row and column interchanges are performed until no more nonzero pivots can be found. The matrices P and Q are constructed to reflect these row and column interchanges. The matrix \( A \) is transformed as follows,

\[
A \rightarrow \begin{bmatrix} I_r & \phi \\ \phi & \phi \end{bmatrix}.
\]

As can be seen, the \( \text{rank}(A) = r \) is determined at the end of this process. Again, we wonder if it is necessary to use multiple modulus residue arithmetic to generate P and Q. Consider the next example.

**Example 3.2.2** Let

\[
A = \begin{bmatrix} 1 & 256 & 0 \\ 0 & 1 & 256 \\ -256 & 256 & 1 \end{bmatrix},
\]

at a glance it is clear that \( A \) is nonsingular so that the transformation of \( A + I_3 \) should generate \( P = Q = I \). Suppose this simple matrix is
sent into a computer implemented Gaussian elimination fixed-point arithmetic method to generate P and Q.

\[
A = \begin{bmatrix}
1 & 2^8 & 0 \\
0 & 1 & 2^8 \\
-2^8 & 2^8 & 1 \\
\end{bmatrix}
\begin{bmatrix}
1 & 2^8 & 0 \\
0 & 1 & 2^8 \\
0 & 2^{24} & 1 \\
\end{bmatrix}
\{2^8 \times \text{row } 1 + \text{row } 3\}
\]

\[
+ \begin{bmatrix}
1 & 2^8 & 0 \\
0 & 1 & 2^8 \\
0 & 2^{32} & -1 \\
\end{bmatrix}
\{2^{24} \times \text{row } 2 - \text{row } 3\}
\]

At this point, if the manufacturers fixed-point word length is 32 bits, some of the integers have overflowed their permissible range and the process cannot continue correctly.

It is true that the P and Q of Example 3.2.2 could be found by using a computer implemented Gaussian elimination floating-point arithmetic method, but given other ill-conditioned matrices it is not unlikely that enough error due to rounding and possibly under/overflow could accumulate so that the P and Q formed would yield a minor of less than full pseudorank. Multiple modulus residue arithmetic, just as fixed-point and floating-point arithmetic, does have its limitations in terms of computer implementation. Actually, there is only one limitation. This is that it may be impossible to find a multiple modulus base \(\{p_1, p_2, \ldots, p_s\}\) such that for all of the \(p_i\),
i = 1, 2, ..., s, arithmetic modulo $p_i$ does not overflow the manufacturers fixed-point word (i.e., $p_i^2$ is within the range of representation.) This may appear to be a difficult problem, but it is not. In multiple modulus residue arithmetic problems it is the product modulus $M = \prod_{i=1}^{s} p_i$, not the individual base elements, that needs to bound the solution and in most cases be quite large. The advantage to the type of limitation imposed by multiple modulus residue arithmetic is that at the onset of the problem it can be determined whether or not there exists a multiple modulus base, \{p_1, p_2, ..., p_s\}, that may solve the problem such that arithmetic modulo $p_i$ can be performed on the computer for all the elements in the base. In the case of floating-point arithmetic, the extent of damage due to rounding error is not known until the solution to the given problem is examined by some method of error analysis. These ideas will be discussed in more detail in Chapter 7.

The next few theorems show that given an arbitrary nxn integral matrix $A$, there exists a base such that permutation matrices $P$ and $Q$ can be found to satisfy Equation 3.2.1. The first result has to do with an arbitrary matrix over the field of real numbers.

**Theorem 3.2.1** Let $A$ be a real nxn matrix with $r = \text{rank}(A)$, if the permutation matrices $P$ and $Q$ are such that

$$PAQ = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$
with $\det(A_{11}) \neq 0$, then $A_{11}$ is contained in an $r \times r$ nonsingular minor of $A$. (The containment need not be proper.)

**Proof:** If $\det(A_{11}) \neq 0$ and $A_{11}$ is of order $r_1 \leq r$ then

$$\text{rank}(A_{11} A_{12}) = \text{rank}\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix} = r_1.$$ 

A well-known result for a real matrix, $A$, is row rank($A$) = column rank($A$). Therefore, there exists an additional $r - r_1$ linearly independent rows of $A$ among the rows in $(A_{11} A_{12})$ and an additional $r - r_1$ linearly independent columns of $A$ among the columns in $[A_{12}]$. Hence, the rows and columns of $A_{11}$ are contained in some $r \times r$ nonsingular minor of $A$.  

The next result relates linear independence of a set of vectors over $\mathbb{F}(p)$ to linear independence of a corresponding set of vectors over the field of rational numbers.

**Theorem 3.2.2** Given an $n \times s$ integral matrix $A$ and the residue of $A$ modulo $p$, $|A|_p$, $p$-prime, if a set of columns in $|A|_p$ are linearly independent over $\mathbb{F}(p)$ then the corresponding set of columns in $A$ are linearly independent over the field of rational numbers.
Proof: Without loss of generality let

$$|A_p| = \begin{pmatrix} A_{11}^p & A_{12}^p \\ A_{21}^p & A_{22}^p \end{pmatrix}$$

where $$\begin{pmatrix} A_{11}^p \\ A_{21}^p \end{pmatrix}$$ is an nxr set of linearly independent columns over $G(p)$. Let $\alpha_1, \alpha_2, \ldots, \alpha_r$ represent the columns in $$\begin{pmatrix} A_{11}^p \\ A_{21}^p \end{pmatrix}$$.

By Definition 2.2.8 if

$$a_{\alpha_1} + a_{\alpha_2} + \ldots + a_{\alpha_r} = 0$$

where $a_{\alpha_i} \in \mathbb{Z}$, then $a_{\alpha_i} = 0$ for all $i = 1, 2, \ldots, r$. Since $p$ is prime either $a_{\alpha_i} = 0$ or $a_{\alpha_i} = kp$ for $k \in \mathbb{Z}$, $q \in \mathbb{Z}^+$ and $p \nmid k$.

Suppose the corresponding set of columns of $A$, $$\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}$$, are linearly dependent. Let $\alpha_1^*, \alpha_2^*, \ldots, \alpha_r^*$ represent the columns of $$\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix}$$. If $a_{\alpha_1}^* + a_{\alpha_2}^* + \ldots + a_{\alpha_r}^* = 0$ then there exists at least one scalar $a_{\alpha_i}^* \neq 0$. Without loss of generality it can be assumed that $a_{\alpha_i} \in \mathbb{Z}$ for all $i = 1, 2, \ldots, r$. If not, they could be scaled to integers and the common denominator divided out of both sides of the equation. Suppose that $a_{\alpha_i} \neq 0$ for $i = 1, 2, \ldots, \ell \leq r$ then these $a_{\alpha_i}$ can be written as $a_{\alpha_i} = k_i p^{q_i}$ for $k_i \in \mathbb{Z}$, $q_i \in \mathbb{Z}^+$ and $p \nmid k_i$.

This gives

$$k_p^{q_1} \alpha_1^* + k_2 p^{q_2} \alpha_2^* + \ldots + k_{\ell} p^{q_{\ell}} \alpha_{\ell}^* = 0.$$
Suppose \( q_j = \min(q_1, q_2, \ldots, q_n) \), then

\[
k_1 p^{q_1 - q_j} a_1^{*} + k_2 p^{q_2 - q_j} a_2^{*} + \cdots + k_j a_j^{*} + \cdots + k_n p^{q_n - q_j} a_n^{*} = \phi,
\]

and

\[
|k_1 p^{q_1 - q_j} a_1^{*} + k_2 p^{q_2 - q_j} a_2^{*} + \cdots + k_j a_j^{*} + \cdots + k_n p^{q_n - q_j} a_n^{*}| = \phi.
\]

Therefore,

\[|k_j a_j^{*}|_p = \phi.\]

Now, \( p \nmid k_j \) so \( |k_j|_p \neq 0 \) and \( |a_j^{*}|_p = a_j = \phi \). This is a contradiction, hence the set of columns \( \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix} \) from \( A \) are linearly independent. \( \square \)

**Corollary 3.2.2.1** Given an \( n \times n \) integral matrix \( A \) and the residue of \( A \) modulo \( p \), \( |A|_p \), p-prime, if a set of rows in \( |A|_p \) are linearly independent over \( \mathbb{G}(p) \) then the corresponding set of rows in \( A \) are linearly independent over the field of rational numbers.

**Proof:** Consider \( A' \) and \( |A'|_p \), then appeal to Theorem 3.2.2. \( \square \)

These last two results will prove to be very useful when trying to find the permutation matrices of Equation 3.2.1 using multiple modulus residue arithmetic. If at the start of the problem the rank(\( A \)) = \( r \) was known, it would just be a matter of trying enough
prime moduli until one of them generated a $P$ and $Q$ such that the resulting nonsingular minor from the product $P|A|_p Q$ was of order $r$.

The next two theorems illustrate how permutation matrices $P$ and $Q$ satisfying Equation 3.2.1 can be found using multiple modulus residue arithmetic without knowing the rank$(A)$ apriori.

**Theorem 3.2.3** Given an $n \times s$ integral matrix $A$ and the residue of $A$ modulo $p$, $|A|_p$, $p$-prime, there exists permutation matrices $P_p$ and $Q_p$ such that

$$P_p |A|_p Q_p = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

with rank$(|A|_p) = \text{rank}(A_{11}) = \text{order } A_{11}$. If the following product is formed,

$$P_p A Q_p = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},$$

then rank$(A_{11}) = \text{rank}(A_{11}) \leq \text{rank}(A)$ and $A_{11} = |A_{11}|_p$.

**Proof:** The entries of $P_p$ and $Q_p$ are zeros and ones, therefore

$$|P_p A Q_p|_p = P_p |A|_p Q_p$$

and

$$A_{11} = |A_{11}|_p.$$
This means $A_{11}$ is nonsingular modulo $p$, hence $|\text{det}(A_{11})|_p \neq 0$ which implies $\text{det}(A_{11}) \neq 0$. By Theorem 3.2.1, $A_{11}$ is contained in a largest nonsingular minor of $A$ therefore,

$$\text{rank}(A_{11}) = \text{rank}(A_{11}) \leq \text{rank}(A).$$

**Theorem 3.2.4** Given $A$ an $n \times n$ integral matrix and a multiple modulus base $\{p_1, p_2, \ldots, p_s\}$, $p_i$-prime, $i = 1, 2, \ldots, s$, such that the product modulus

$$M = \prod_{i=1}^{s} p_i > 2|\det(\text{any minor of } A)|,$$

then there exists at least one $p_i$ such that

$$\text{rank}(|A|_{p_i}) = \text{rank}(A).$$

**Proof:** Let $r = \text{rank}(A)$ and suppose $A_{11}$ is a $r \times r$ nonsingular minor of $A$. Now, $M > 2|\det(A_{11})|$ and by Theorem 2.4.1 there exists at least one modulus, say $p^*$, from the multiple modulus base such that

$$\text{rank}(A_{11}) = \text{rank}(|A_{11}|_{p^*}) = \text{order } A_{11}.$$

Also,

$$\text{rank}(A) \geq \text{rank}(|A|_{p^*}) \geq \text{rank}(|A_{11}|_{p^*}) = \text{order } A_{11} = \text{rank}(A)$$

therefore,

$$\text{rank}(A) = \text{rank}(|A|_{p^*}).$$
These last two theorems show that for every prime modulus, \( p \), permutation matrices \( P_p \) and \( Q_p \) can be found such that

\[
P_p A Q_p = \begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\]

and \( \text{rank}(A_{11}) = \text{rank}(|A|_p) = \text{order } A_{11} \). Also, if the multiple modulus base, \( \{p_1, p_2, \ldots, p_s\} \), \( p_1 \)-prime, is chosen so the product modulus satisfies Theorem 3.2.4 then at least one of the moduli in the base, say \( p^* \), will generate matrices \( P_{p^*} \) and \( Q_{p^*} \) yielding

\[
P_{p^*} A Q_{p^*} = \begin{bmatrix}
A_{11}^* & A_{12}^* \\
A_{21}^* & A_{22}^*
\end{bmatrix}
\]

with \( \text{rank}(A) = \text{rank}(|A|_{p^*}) = \text{rank}(|A_{11}^*|_{p^*}) = \text{rank}(A_{11}^*) = \text{order } A_{11}^* \).

There is no mystery as to which element of the base is \( p^* \), it is a prime for which

\[
\text{rank}(|A|_{p^*}) = \max_{1 \leq i \leq s} (\text{rank}(|A|_{p_i})).
\]

This modulus, \( p^* \), is not necessarily unique. Several of the elements in the multiple modulus base can yield \( \text{rank}(|A|_p) = \text{rank}(A) \) and generate permutation matrices \( P_p \) and \( Q_p \) satisfying Equation 3.2.1. It should be understood that even if \( \text{rank}(|A|_{p_i}) = \text{rank}(|A|_{p_j}) \), the nonsingular minors of \( A \) generated as a result of forming the products \( P_{p_i} A Q_{p_i} \) and \( P_{p_j} A Q_{p_j} \) are not necessarily the same. For constructing
a reflexive generalized inverse in the form of Equation 3.2.2, any
permutation matrices $P$ and $Q$ which generate a $r \times r$ nonsingular
minor, $A_{11}$, of $A$ will work.

The choice of a base, $\{p_1, p_2, \ldots, p_s\}$, to satisfy Theorem
3.2.4 is not a trivial matter. In addition to the product modulus
satisfying Theorem 3.2.4 so a $r \times r$ nonsingular minor, $A_{11}$, of $A$ can
be found, the product modulus must also satisfy

$$M > 2 \max\left(|\det(A_{11})|, \max\{|A_{ij}^{(11)}|\}\right),$$

where $(A_{ij}^{(11)})$ are the cofactors of $A_{11}$. This is to insure
$\det(A_{11})$ and $A_{11}^{\text{adj}}$ can be extracted from their multiple modulus
residue representations. However,

if $M > 2|\det(\text{any minor of } A)|$

then $M > 2|\det(A_{11})|$

and $M > 2|\det(A_{ij}^{(11)})|$ for all $i$ and $j$. This is true because the
submatrix $A_{11}$ is a $r \times r$ minor of $A$ and each cofactor of $A_{11}$ is
simply the determinant of a $(r-1) \times (r-1)$ minor of $A$. How to find
this product modulus, $M$, is discussed in the next section.

Once a $r \times r$ nonsingular minor of $A$ is found the multiple modulus
residue arithmetic nonsingular matrix inversion method outlined in Section
2.3 and 2.4 can be used to invert this minor. Finally, the construction of
$A^-$ can be completed by forming the product given in Equation 3.2.2. In
Section 3.2.4, two algorithms using multiple modulus residue arithmetic to
compute $A^{-1}$ as described in this section are given. One method employs a modified version of Gaussian elimination to invert an nxs integral matrix. The other method is a bordering process which inverts an nxn positive semidefinite integral symmetric matrix.

3.3 The Multiple Modulus Base and Corresponding Product Modulus

To use multiple modulus residue arithmetic to obtain an exact generalized inverse of an arbitrary nxs integral matrix it is necessary to choose an appropriate multiple modulus base, $\{p_1, p_2, \ldots, p_s\}$. This section discusses how to choose such a base.

It was determined in Section 3.2 that the product modulus, $M = \prod p_i$, must satisfy

$$M > 2|\det(\text{any minor of } A)|.$$  \hspace{1cm} (3.3.1)

Hadamard's Inequality for the determinant of an nxn real matrix and a bound due to Schinzel (1978) for the determinant of an nxn real matrix will be modified so they can be used to find a product modulus to satisfy Equation 3.3.1. A third and fairly new determinant bound due to Johnson and Newman (1980) for the determinant of a real nxn matrix will be given. All three of these determinant bounds are conservative and none of them is uniformly smaller than the other two. As far as determinantal bounds are concerned, no uniformly best bound for the determinant of an arbitrary nxn real matrix has yet been discovered.
First, Hadamard's Inequality for the determinant of an \( nxn \) real matrix will be examined.

**Theorem 3.3.1** \ Let \( A = (a_{ij}) \) be a real \( nxn \) matrix, then Hadamard's Inequality for the determinant of \( A \) is

\[
|\text{det}(A)| \leq \prod_{i=1}^{n} \left( \sum_{j=1}^{n} a_{ij}^2 \right)^{\frac{1}{2}}.
\]

This is a well-known result so the proof will be omitted. The next corollary extends Hadamard's Inequality to give a bound for the determinant of any minor from a real \( nxs \) matrix.

**Corollary 3.3.1.1** \ Let \( A = (a_{ij}) \) be a real \( nxs \) \((n \leq s)\) matrix, then

\[
|\text{det(any minor of } A)| \leq \prod_{i=1}^{n} \left( \sum_{j=1}^{s} a_{ij}^2 \right)^{\frac{1}{2}}.
\]

**Proof:** \ Let \( A^* \) be a \( k \times k \) minor of \( A \). Without loss of generality let \( A^* \) be made up of the first \( k \) rows and the first \( k \) columns of \( A \). Then,

\[
|\text{det}(A^*)| \leq \prod_{i=1}^{k} \left( \sum_{j=1}^{k} a_{ij}^2 \right)^{\frac{1}{2}}
\]

\[
\leq \prod_{i=1}^{n} \left( \sum_{j=1}^{s} a_{ij}^2 \right)^{\frac{1}{2}}. \quad \Box
\]
Notice in the above corollary, if $n = s$ then the usual Hadamard's Inequality is also a bound for $|(A_{ij})|$, where $(A_{ij})$ are the cofactors of the square matrix. The cofactors are just the determinants of $(n-1) \times (n-1)$ minors of $A$.

Next consider a determinant inequality due to Schinzel (1978).

**Theorem 3.3.2** Let $A = (a_{ij})$ be a real $n \times n$ matrix, for each $i = 1, 2, \ldots, n$ set

$$R_i^+ = \sum_{j=1}^{n} \max(0, a_{ij}) \quad \text{and} \quad R_i^- = -\sum_{j=1}^{n} \min(0, a_{ij}).$$

Then,

$$|\det(A)| \leq \prod_{i=1}^{n} \max(R_i^+, R_i^-).$$

This theorem can also be extended to give a bound for the determinant of any minor from a real $n \times s$ matrix.

**Corollary 3.3.2.1** Let $A = (a_{ij})$ be a real $n \times s$ $(n \leq s)$ matrix. For each $i = 1, 2, \ldots, n$ set

$$R_i^+ = \sum_{j=1}^{s} \max(0, a_{ij}) \quad \text{and} \quad R_i^- = -\sum_{j=1}^{s} \min(0, a_{ij}).$$

If for some $i$, $R_i^+ = R_i^- = 0$ then set $R_i^+ = R_i^- = 1$. (This insures that the $\max(R_i^+, R_i^-) \neq 0$, for all $i$.) Then,

$$|\det(\text{any minor of } A)| \leq \prod_{i=1}^{n} \max(R_i^+, R_i^-).$$
Proof: Let $A^*$ be a $k \times k$ minor of $A$. For each row, $i$, in $A^*$ set

$$R_{i}^{+} = \sum_{j} \max(0, a_{ij}) \quad \text{and} \quad R_{i}^{-} = -\sum_{j} \min(0, a_{ij}).$$

Then,

$$|\det(A^*)| \leq \Pi \max(R_{i}^{+}, R_{i}^{-}).$$

For each row $i$ in $A^*$

$$R_{i}^{+} = \sum_{j=1}^{s} \max(0, a_{ij}) \geq R_{i}^{*+} \quad \text{and} \quad R_{i}^{-} = -\sum_{j=1}^{s} \min(0, a_{ij}) \geq R_{i}^{*-}.$$ 

Hence,

$$\max(R_{i}^{+}, R_{i}^{-}) \geq \max(R_{i}^{*+}, R_{i}^{*-}),$$

and

$$|\det(A^*)| \leq \Pi \max(R_{i}^{*+}, R_{i}^{*-}).$$

When $n = s$, this corollary also gives a bound for the cofactors of the square matrix. This would not necessarily be the case if the condition that for all $i = 1, 2, \ldots, n$, $\max(R_{i}^{+}, R_{i}^{-}) \neq 0$. 


Given an mxs (n ≤ s) integral matrix A either the extended Hadamard’s Inequality or the extended Schinzel’s Inequality for the bound of the determinant of any minor can be computed. Once a bound is computed then a multiple modulus base, \{p_1, p_2, \ldots, p_s\}, is chosen such that the product modulus, M = \Pi p_i, exceeds this bound.

**Example 3.3.1** Let A = 
\[
\begin{bmatrix}
5 & -3 & -1 \\
0 & -1 & 3 \\
\end{bmatrix}
\]

and find M such that

\[M > 2|\text{det(any minor of A)}| .\]

The Extended Hadamard’s Inequality gives

\[|\text{det(any minor of A)}| \leq (25 + 9 + 1)^{\frac{1}{2}}(0 + 1 + 9)^{\frac{3}{2}} \approx 19.\]

The base \{3, 5, 7\} yields

\[M = 105 > 2(19).\]

The Extended Schinzel’s Inequality gives

\[R_1^+ = 5 , \quad R_1^- = 4\]
\[R_2^+ = 1 , \quad R_2^- = 3\]

and

\[|\text{det(any minor of A)}| \leq 5\cdot 3 = 15.\]

The base \{5, 7\} yields

\[M = 35 > 2(15) .\]

Either base found in this example could be used to generate A exactly using the multiple modulus residue arithmetic method discussed.
in Section 3.2. In this example, Schinzel's Inequality gave a smaller bound than Hadamard's Inequality. This is not always the case as the following example shows.

Example 3.3.2 Let \( A = \begin{bmatrix} 6 & 5 \\ 4 & 4 \end{bmatrix} \), \( A \) is nonsingular so find \( M \) such that \( M > 2|\det(A)| \).

Hadamard's Inequality gives
\[
|\det(A)| < (36 + 25)^{\frac{1}{2}}(16 + 16)^{\frac{1}{2}} \approx 44.
\]
The base \( \{3, 5, 7\} \) yields
\( M = 105 > 2(44) \).

Schinzel's Inequality gives
\[
\begin{align*}
R_1^+ &= 11, \\
R_1^- &= 0, \\
R_2^+ &= 8, \\
R_2^- &= 0
\end{align*}
\]
and
\[
|\det(A)| \leq 11 \cdot 8 = 88.
\]
The base \( \{3, 5, 7, 11\} \) yields
\( M = 1155 > 2(88) \).

Once again either base could be used to generate \( A^- (= A^{-1} \) in this case) using multiple modulus residue arithmetic. This time Hadamard's Inequality gave a tighter bound. In this simple example, the \( \det(A) = 4 \) and the \( \max_{i,j} |A_{ij}| = 6 \), so the multiple modulus base of \( \{3, 5\} \) with the product modulus of \( M = 15 > 2(6) \) would suffice to generate \( A^{-1} \) using
multiple modulus residue arithmetic. This illustrates that these two bounds can be quite conservative; however, they do both provide an appropriate multiple modulus base. As was indicated in these two small examples, one could benefit by computing both bounds and constructing the multiple modulus base and product modulus based on the minimum of the two bounds.

At the start of the multiple modulus residue arithmetic generalized inverse problem, it is necessary to choose a multiple modulus base where the product modulus satisfies Equation 3.3.1. This is because until permutation matrices \( P \) and \( Q \) are found it is not known which \( r \times r, r = \text{rank}(A) \), nonsingular minor they will generate. Section 3.2 pointed out it is possible for different prime moduli to generate different permutation matrices which in turn yield different \( r \times r \) nonsingular minors. Once a \( r \times r \) nonsingular minor, say \( A_{11} \), has been determined then the multiple modulus base need only have enough elements so that the product modulus satisfies

\[
M > 2 \max \left\{ \det(A_{11}), \max_{1, j} \left| (A_{11})_{ij} \right| \right\}, \tag{3.3.2}
\]

where \( (A_{11})_{ij} \) are the cofactors of \( A_{11} \). The multiple modulus base that was used to find \( A_{11} \) will satisfy this equation, but it is possible that a much smaller base would suffice for inverting \( A_{11} \) using multiple modulus residue arithmetic.
Example 3.3.3  Let \( A = \begin{bmatrix} 5 & 1 & 5 & 2 \\ 1 & 3 & 1 & 3 \\ 5 & 1 & 5 & 2 \end{bmatrix} \), find the appropriate multiple modulus base to generate permutation matrices \( P \) and \( Q \).

The Extended Hadamard's Inequality gives

\[ |\det(\text{any minor of } A)| \leq 246. \]

The base \{3, 5, 7, 11\} yields

\[ M = 1155 > 2(246). \]

The Extended Schinzel's Inequality gives

\[ |\det(\text{any minor of } A)| \leq 1352. \]

The base \{5, 7, 11, 13\} yields

\[ M = 5005 > 2(1352). \]

In this example, the rank(\( A \)) = 2 and a possible 2x2 nonsingular minor is

\[ A_{11} = \begin{bmatrix} 5 & 1 \\ 1 & 3 \end{bmatrix}. \]

For \( A_{11} \), Hadamard's Inequality gives

\[ \max[|\det(A_{11})|, \max_{i,j}|(A^{(11)})_{i,j}|] \leq 16.13. \]

The base \{5, 7\} yielding

\[ M = 35 > 2(16.13) \]

would suffice for inverting \( A_{11} \) using multiple modulus residue arithmetic.

Schinzel's Inequality for \( A_{11} \) gives

\[ \max[|\det(A_{11})|, \max_{i,j}|(A^{(11)})_{i,j}|] \leq 24. \]
The base \{7, 11\} yielding

\[ M = 77 > 2(24) \]

would suffice for inverting \( A_{11} \) using multiple modulus residue arithmetic.

There is one more bound for the determinant of a real \( nxn \) matrix which is of interest. This is a bound due to Johnson and Newman (1980).

**Theorem 3.3.3** Let \( A = (a_{ij}) \) be a real \( nxn \) matrix. For each \( i = 1, 2, \ldots, n \), set

\[
R_i^+ = \sum_{j=1}^{n} \max(0, a_{ij}) \quad \text{and} \quad R_i^- = -\sum_{j=1}^{n} \min(0, a_{ij}).
\]

Then

\[
|\det(A)| \leq \prod_{i=1}^{n} \max(R_i^+, R_i^-) - \prod_{i=1}^{n} \min(R_i^+, R_i^-).
\]

Johnson and Newman (1980) have shown that when restricted to determinantal inequalities which only involve the quantities \( R_i^+ \) and \( R_i^- \) for \( i = 1, 2, \ldots, n \), the bound in Theorem 3.3.3 is a minimum.

This means for a real \( nxn \) matrix \( A \), if

\[
|\det(A)| \leq f(R_1^+, R_1^-, \ldots, R_n^+, R_n^-)
\]

then

\[
\prod_{i=1}^{n} \max(R_i^+, R_i^-) - \prod_{i=1}^{n} \min(R_i^+, R_i^-) \leq f(R_1^+, R_1^-, \ldots, R_n^+, R_n^-).
\]
For instance, this bound due to Johnson and Newman (1980) is always less than or equal to Schinzel's bound in Theorem 3.3.2. Unfortunately, Hadamard's Inequality is not a function of $R^+_i$ and $R^-_i$ for $i = 1, 2, \ldots, n$. Johnson and Newman's bound does not directly extend to forming a bound for the determinant of any minor of a real $n \times n$ matrix. Also, their bound does not necessarily bound the cofactors of the $n \times n$ matrix.

**Example 3.3.4** Let $A = \begin{bmatrix} 4 & -3 \\ -3 & 3 \end{bmatrix}$, then

- $R^+_1 = 4$, $R^-_1 = 3$
- $R^+_2 = 3$, $R^-_2 = 3$.

Johnson and Newman's bound yields

$$|\det(A)| \leq 4 \cdot 3 - 3 \cdot 3 = 3,$$

but 3 does not bound the $\max_{i,j} |A_{ij}| = 4$.

This bound due to Johnson and Newman (1980) can be computed for each of the cofactors of the matrix. The maximum between the largest cofactor bound and the bound for the determinant can be used to construct the multiple modulus base. An algorithm for computing Johnson and Newman's bound for the determinant of an $n \times n$ real matrix $A$ and the bounds for all the cofactors of $A$ is given in Section 3.4. For some matrices, there is a substantial benefit for computing these bounds as opposed to using Hadamard's Inequality. In Example 3.3.4, according to Johnson and Newman's bounds, a product modulus of
M > 8 is sufficient for inverting A using multiple modulus arithmetic. For this example Hadamard's inequality requires a product modulus of M > 42.43. There are matrices, however, where Hadamard's Inequality gives a tighter bound on the determinant of the matrix and its cofactors. (See Example 3.3.2, here Johnson and Newman's bound is equivalent to Schinzel's bound.) Here again, a possible strategy would be to compute both types of bounds and use the smallest one.

It is not a trivial problem to find the bounds necessary such that the product modulus will be large enough to guarantee the solution of the matrix inversion problem over the field of rational numbers can be extracted from the multiple modulus residue arithmetic result. A more conservative bound will utilize unnecessary moduli in the multiple modulus base. Although the bounds discussed in this section may be conservative, the resulting multiple modulus bases are suitable for solving the multiple modulus residue arithmetic matrix inversion problem.

3.4 Algorithms for Finding Generalized Inverses Using Multiple Modulus Residue Arithmetic

The first algorithm given in this section computes the generalized inverse of an nx s (n ≤ s) integral matrix. The method that is used is identical to the method described in Section 3.2. The restriction n ≤ s is not serious because if n > s then simply compute 

\[(A')^- = (A^-)^t\]. The second algorithm in this section is for an integral
nxn positive semidefinite symmetric matrix, i.e., \( A = X'X \). If \( A \) is a nonnegative definite symmetric matrix the second algorithm is much more efficient than the first. Inverting a symmetric matrix is of particular interest to the statistician. The role this research plays in statistics is treated in Chapter 5. Both of these algorithms have been implemented in standard Fortran. The implementation are discussed and some sample output is given in Chapter 7.

3.4.1 Multiple modulus residue arithmetic Gaussian elimination algorithm for generalized matrix inversion of an integral matrix

Let \( A = (a_{ij}) \) be an integral nxn \((n \leq s)\) matrix. This algorithm computes \( A^{-1} \) by the method described in Section 3.2.

1. Compute the product modulus bound \( M \) such that
   \[ M > 2|\det(\text{any minor of } A)|. \]
   This bound \( M \) is found by using the extended Schinzel's Inequality given in Corollary 3.3.2.1.

   Set \( i = 0, \ M = 1 \)
   i) \( i = i + 1, \ R_i^+ = 0, \ R_i^- = 0, \ j = 1 \).
   ii) \( R_i^+ = R_i^+ + \max(0, a_{ij}) \) and \( R_i^- = R_i^- - \min(0, a_{ij}) \).
   iii) If \( j < s \) set \( j = j + 1 \) and go to (ii). Else go to (iv).
   iv) \( M = M \cdot \max(R_i^+, R_i^-, 1) \), if \( i < n \) go to (i). Else go to (v).
   v) Exit with \( M = 2 \cdot M \).

2. Choose the primes \( \{p_1, p_2, \ldots, p_s\} \) such that
   \[ \prod_{i=1}^{s} p_i > M, \]
These primes make up the multiple modulus base to be used to find permutation matrices $P$ and $Q$ and the rank($A$).

3. Find permutation matrices $P$ and $Q$ and the rank($A$) such that the product $PAQ$ satisfies Equation 3.2.1. To accomplish this, start with the prime modulus $p_1$ and perform Gaussian elimination modulo $p_1$ on $A$ invoking row and column interchanges to bring $A$ to upper triangular form. That is

$$|A|_{p_1} \rightarrow \begin{bmatrix} \phi & * \\ * & * \\ \phi & \phi \end{bmatrix}.$$ 

Create the permutation matrices $WP$ and $WQ$ to reflect the row and column interchanges. The rank($|A|_{p_1}$) is equal to the number of pivots performed. At this step, assign rank($A$) = rank($|A|_{p_1}$) and $P = WP$, $Q = WQ$. Next, repeat the same Gaussian elimination process for $p_i$, $i = 2, 3, \ldots, s$. If rank($|A|_{p_i}$) > rank($A$) set rank($A$) = rank($|A|_{p_i}$), $P = WP$ and $Q = WQ$. The rank($|A|_{p_i}$) is retained for all $i = 1, 2, \ldots, s$, because at Step 7 $|A|_{p_i}$ is immediately set to the null matrix if rank($|A|_{p_i}$) < rank($A$) - 1 and then computation skips to the next prime. Set $q = 0$, $P = I$, $Q = I$, rank = 0.

i) $i = 1$, $j = n$, $q = q + 1$, rank$^q = n$, $WP = I$, $WQ = I$.

ii) Compute $|A|_{p_q} = (a_{ij})$

iii) If $a_{ii} \neq 0$ go to (iv).

iiia) Set $k = i$, if $k = s$ go to (iiic). Else go to (iiib).

iiib) Set $k = k + 1$. If $a_{kk} \neq 0$ interchange columns $i$ and $k$ in $WQ$ and $|A|_{p_q}$, then go to (iv). If $a_{kk} = 0$ and $k < s$ repeat step (iiib). Else go to (iiic).
iii) If \( i = \text{rank}_q \), set \( \text{rank}_q = \text{rank}_q - 1 \) and go to (vii).

Else interchange rows \( i \) and \( \text{rank}_q \) in \( WP \) and \( |A|_{p_q} \),
set \( \text{rank}_q = \text{rank}_q - 1 \) and go to (iii).

iv) If \( i = \text{rank}_q \), go to (vii). Else multiple row \( i \) by \( a_{ii}^{-1}(p_q) \)
reducing modulo \( p_q \), then set \( \ell = i + 1 \).

v) If \( a_{\ell\ell} = 0 \), go to (vi). Else multiple row \( i \) by \( p_q - a_{i\ell} \)
and add row \( i \) to row \( \ell \) reducing modulo \( p_q \).

vi) If \( \ell = n \), set \( i = i + 1 \) and go to (iii). Else set \( \ell = \ell + 1 \)
and go to (v).

vii) If \( \text{rank}_q > \text{rank} \), set \( \text{rank} = \text{rank}_q \), \( P = WP \) and \( Q = WQ \).

viii) If \( q = s \) exit with \( \text{rank} = \text{rank}(A), P \) and \( Q \). Else go to (i).

4. Compute

\[
PAQ = \begin{bmatrix}
A^* & \ast \\
\ast & \ast
\end{bmatrix},
A^* = (a_{ij}^*).
\]

Let \( r = \text{rank} = \text{rank}(A) = \text{rank}(A^*) = \text{order } A^* \).

5. Compute the bound \( M^* \) such that

\[
M^* > 2 \max\{|\det(A^*)|, \max_{i,j} |(A^*)_{ij}|\}.
\]

This will be done by using the bound due to Johnson and Newman (1980). So,

\[
|\det(A^*)| \leq \prod_{i=1}^{r} \max(R_i^+, R_i^-) - \prod_{i=1}^{r} \min(R_i^+, R_i^-),
\]

where \( R_i^+ = \sum_{j=1}^{r} \max(0, a_{ij}^*) \) and \( R_i^- = - \sum_{j=1}^{r} \min(0, a_{ij}^*) \).
The cofactor bounds are

\[
\left| (A^*_{\mathbf{L},m}) \right| \leq \prod_{i=1}^{r} \max(R^*_{i}, R^*_{i}) - \prod_{i=1}^{r} \min(R^*_{i}, R^*_{i}),
\]

where \( R^*_{i} = R_{i}^+ - \max(0, a_{i}^*) \) and \( R^*_{i} = R_{i}^- - \min(0, a_{i}^*) \).

If \( R^+_{i} = 0 \) or \( R^-_{i} = 0 \) for any \( i \), then

\[
\prod_{i=1}^{r} \min(R^+_{i}, R^-_{i}) = 0
\]

and if \( R^+_{i} = 0 \) or \( R^-_{i} = 0 \) for any \( i \neq \ell \), then

\[
\prod_{i=1}^{r} \min(R^*_{i}, R^*_{i}) = 0.
\]

Set \( i = 0, \text{zero} = 0, \text{Bdmax} = 1, \text{Bdmin} = 1 \).

i) Set \( i = i + 1, R^+_{i} = 0, R^-_{i} = 0, j = 1 \).

ii) \( R^+_{i} = R^+_{i} + \max(0, a_{ij}^*), \) and \( R^-_{i} = R^-_{i} - \min(0, a_{ij}^*) \).

iii) If \( j < r \), set \( j = j + 1 \) and go to (ii).

If \( i < r \), set \( \text{zero} = 1 \) if \( R^+_{i} = 0 \) or \( R^-_{i} = 0 \) and go to (i).

Else set \( i = 0 \) and go to (iv).

iv) Set \( i = i + 1, \text{Bdmax} = \text{Bdmax} \cdot \max(R^+_{i}, R^-_{i}) \)

and \( \text{Bdmin} = \begin{cases} \text{Bdmin} \cdot \min(R^+_{i}, R^-_{i}) & \text{zero} = 0 \\ 0 & \text{zero} = 1. \end{cases} \)

v) If \( i = n \), \( \text{BdDET}(A^*) = \text{Bdmax} - \text{Bdmin} \), set \( i = 0 \) and \( m = 0 \)

then go to (vi). Else to to (iv).

vi) Set \( M = m + 1, \ell = 1, \text{zero} = 0, i = 0, \text{Bdmax} = 1 \) and \( \text{Bdmin} = 1 \).

vii) Set \( i = i + 1, \) if \( i \neq \ell \) set \( R^+_{i} = R^+_{i} - \max(0, a_{i}^*) \) and \( R^-_{i} = R^-_{i} + \min(0, a_{i}^*) \) then go to (viii). If \( \ell = r \) go to (ix). Else set \( \ell = \ell + 1 \) and repeat step (vii).
viii) If $R_{1}^{+} = 0$, or $R_{1}^{-} = 0$ set zero = 1.

ix) If $i < r$, go to (vii). Else set $i = 0$ and go to (x).

x) Set $i = i + 1$, if $i 
eq l$

$$Bd_{\text{max}} = \text{Bd}_{\text{max}} \cdot \max(R_{1}^{+}, R_{1}^{-})$$

and

$$Bd_{\text{min}} = \left\{ \begin{array}{ll}
\text{Bd}_{\text{min}} \cdot \min(R_{1}^{+}, R_{1}^{-}) & \text{zero} = 0 \\
0 & \text{zero} = 1.
\end{array} \right.$$  

xi) If $i < r$, go to (x). Else set $Bd_{\text{A}}^* = Bd_{\text{max}} - Bd_{\text{min}}$.

xii) If $l < r$, set $l = l + 1$, zero = 0, $i = 0$ and go to (vii).

If $m < r$, go to (vi). Else go to (xiii).

xiii) Compute $Bd_{\text{ADJ}}(A^*) = \max_{1 \leq m < r, 1 \leq j < r}(Bd_{\text{A}}^*_{jm})$.

xiv) Exit with $M^* = 2 \max(\text{BdDET}(A^*), \text{BdADJ}(A^*))$.

6. Choose the primes $p_q, q = 1, 2, \ldots, h$ from the original multiple modulus base found in Step 2, such that

$$\Pi_{q=1}^{h} p_q > M^*.$$  

These primes make up the multiple modulus base to be used to invert $A^*$ exactly using multiple modulus residue arithmetic.

7. Compute $|A^*_{\text{adj}}|_{p_q}$ and $|\text{det}(A^*)|_{p_q}$ for all $q = 1, 2, \ldots, h$, by applying residue arithmetic to extended Gauss-Jordan elimination as discussed in Section 2.3. If for some $p_q$ the

$$\text{rank}(|A^*|_{p_q}) < \text{rank}(A)-1$$

set $A^*_{\text{adj}}|_{p_q} = \emptyset$ and $|\text{det}(A^*)|_{p_q} = 0$. Continue
then to the next prime, \( p_{q+1} \). If \( \text{rank}(|A^*|_p) \geq \text{rank}(A) - 1 \), the multiple modulus residue arithmetic extended Gauss-Jordan procedure outlined by Cabay and Lam (1977a,b) can be used to generate \( |A^{adj}|_p \)
and \( |\det(A^*)|_p \).

8. Let \( \mathcal{M} = \prod_{q=1}^{h} p_q \) and form \( |A^{adj}|_{\mathcal{M}} \) and \( |\det(A^*)|_{\mathcal{M}} \) by combining the multiple modulus residue representations of these quantities. (The details of this step will be given in Chapter 7.)

9. Set \( \det(A^*) = |\det(A^*)|_{\mathcal{M}} \) and

\[
A^{adj} = |A^{adj}|_{\mathcal{M}}.
\]

10. Form the product

\[
A^{-} = \frac{1}{\det(A^*)} Q \begin{bmatrix} A^{adj} & \phi \\ \phi & \phi \end{bmatrix} P.
\]

This concludes the first algorithm. It is probably the most complicated method, from a computational point of view, included in this research. Chapter 4 deals with using multiple modulus residue arithmetic to find the Moore-Penrose inverse of an arbitrary \( n \times s \) integral matrix. Oddly enough, the algorithm which generates the Moore-Penrose inverse has much more ease of computation than this algorithm. This method, however, is probably the most straightforward and easiest to understand of all the multiple modulus residue arithmetic generalized matrix inversion processes given in this thesis.
The next method to be discussed is a generalized matrix inversion method for an \( n \times n \) positive semidefinite symmetric matrix. The matrix \( A \) is a positive semidefinite symmetric matrix has the form \( A = X'X \) for some real matrix \( X \). Once again it will be assumed that \( A \) is integral. (For \( A \) to be integral does not necessarily require that \( X \) be integral.) In this method, only one permutation matrix, \( P \), will be sought such that the product

\[
PAP' = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}
\]

is symmetric with \( \text{rank}(A) = \text{rank}(A_{11}) = \text{order } A_{11} \). Multiple modulus residue arithmetic will be applied to a bordering process to find a permutation matrix \( P \) to satisfy Equation 3.4.1 and simultaneously invert the full-rank minor, \( A_{11} \).

It is first necessary to describe matrix inversion by bordering for a symmetric nonsingular matrix. This method can be found in Hemmerle (1967). Consider the \( k \times k \) symmetric matrix \( A_k \) partitioned as

\[
A_k = \begin{bmatrix}
    a_{11} & a_{12} & \cdots & a_{1(k-1)} & a_{1k} \\
    a_{12} & a_{22} & \cdots & a_{2(k-1)} & a_{2k} \\
    \vdots & \vdots & \ddots & \vdots & \vdots \\
    a_{1(k-1)} & a_{2(k-1)} & \cdots & a_{(k-1)(k-1)} & a_{(k-1)k} \\
    a_{1k} & a_{2k} & \cdots & a_{(k-1)k} & a_{kk}
\end{bmatrix}
\]

\[
= \begin{bmatrix}
    A_{k-1} & b_k \\
    b_k' & a_{kk}
\end{bmatrix}
\]
The bordering process which is a special case of the Frobenius-Schur relation gives

\[
A_k^{-1} = \begin{bmatrix}
-A_k^{-1} + (A_{k-1}^{-1} b_k b_k' A_{k-1}^{-1})/c_k & -(A_{k-1}^{-1} b_k)/c_k \\
-(b_k' A_{k-1}^{-1})/c_k & 1/c_k
\end{bmatrix}
\]  

(3.4.3)

where \( c_k = a_{kk} - b_k' A_{k-1}^{-1} b_k \).

In order to use residue arithmetic with the bordering method it is necessary to express \( A_k^{-1} \) as

\[
A_k^{-1} = \frac{1}{\text{cf}} A^* 
\]

(3.4.4)

where \( \text{cf} \) is an integer and \( A^* \) is integral. The next result will make this possible.

**Theorem 3.4.1** If \( A_k \) is a \( k \times k \) symmetric matrix as defined in Equation 3.4.2, then

\[
\det(A_k) = \det(A_{k-1}) c_k. 
\]

**Proof:** We need to show that

\[
\det(A_k) = \det(A_{k-1}) c_k = \det(A_{k-1}) a_{kk} - b_k' A_{k-1} \text{adj} b_k. 
\]

Let \( A_{(k)} \) represent the \((j,k)\)th cofactor of \( A_k \), then

\[
\det(A_k) = \sum_{j=1}^{k} a_{jk} (A_{j,k}^{(k)}) = \sum_{j=1}^{k-1} a_{ij} (A_{j,k}^{(k)}) + a_{kk} (-1)^{2k} \det(A_{k-1}).
\]

Let \( A_{k}^{(j,i)} \) represent \( A_k \) with \( j \)th row and \( i \)th column removed,
\[
A_{k}^{(j,1)} = \begin{bmatrix}
a_{11} & \cdots & a_{1(i-1)} & a_{1(i+1)} & \cdots & a_{1k} \\
\vdots & & \vdots & \vdots & & \vdots \\
a_{(j-1)1} & \cdots & a_{(j-1)(i-1)} & a_{(j-1)(i+1)} & \cdots & a_{(j-1)k} \\
a_{(j+1)1} & \cdots & a_{(j+1)(i-1)} & a_{(j+1)(i+1)} & \cdots & a_{(j+1)k} \\
\vdots & & \vdots & \vdots & & \vdots \\
a_{kl} & \cdots & a_{k(i-1)} & a_{k(i+1)} & \cdots & a_{kk}
\end{bmatrix}.
\]

So, 
\[A_{j,k}^{(k)} = (-1)^{j+k} \det(A_{k}^{(j,k)})\]
and 
\[\det(A_{k}^{(j,k)}) = \sum_{i=1}^{k-1} (-1)^{i+k} \det(A_{k-1}^{(j,i)})\]

The \(j^{th}\) row of \(A_{k}^{\text{adj}}\) contains \((-1)^{j+i} \det(A_{k-1}^{(j,i)})\), \(i = 1, 2, \ldots, k-1\)

and 
\[a_{jk}(A_{j,k}^{(k)}) = (-1)^{j+k} \det(A_{k}^{(j,k)})\]

\[
= a_{jk} (-1)^{j+k} \sum_{i=1}^{k-1} (-1)^{i+k-1+i} (-1)^{j+i} \det(A_{k-1}^{(j,i)})
\]

\[
= a_{jk} (-1)^{2j+2k-1} \text{\(j^{th}\) row of } A_{k-1}^{\text{adj}} \text{ of } A_{k}^{\text{adj}} \quad b_k.
\]

Therefore, 
\[\det(A_k) = a_{kk} \det(A_{k-1}) + \sum_{j=1}^{k-1} (-1)^{j} a_{jk} \text{\(j^{th}\) row of } A_{k-1}^{\text{adj}} \text{ of } A_{k}^{\text{adj}} \quad b_k
\]

\[
= a_{kk} \det(A_{k-1}) - b_k A_{k-1}^{\text{adj}} b_k
\]

\[
= \det(A_{k-1}) c_k.
\]
With this result in mind Equation 3.4.3 can be rewritten as

\[
\begin{bmatrix}
\frac{\text{adj } A_{k-1}}{\text{det}(A_{k-1})} & \frac{\text{adj } b_k b'_k A_{k-1}}{\text{det}(A_{k-1})^2 c_k} & -\frac{\text{adj } b_k}{\text{det}(A_{k-1}) c_k} \\
-b'_k A_{k-1} & \frac{1}{c_k} & \\
\frac{1}{\text{det}(A_{k-1}) c_k} & & \\
\end{bmatrix}
\]

\hspace{1cm} (3.4.5)

With \( cf = \text{det}(A_{k-1}) \text{det}(A_k) \), \( A_k^{-1} \) satisfies Equation 3.4.4. It may seem unfortunate that the common factor, \( cf \), is the product of two determi­nants; however, after the product sum, \( \text{det}(A_k) A_{k-1} \text{adj } + A_{k-1} b_k b'_k A_{k-1} \text{adj} \), is formed the \( \text{det}(A_{k-1}) \) can be factored out of the entire matrix. (If \( A_k \) is integral \( A_{k-1} \text{adj} \) is integral.) Then the end result of the \( k \)th step is

\[
A_k^{-1} = \frac{1}{\text{det}(A_k)} A_k \text{adj}
\]

and the bordering process can proceed to step \( k+1 \) with \( \text{det}(A_k) \) and \( A_k \text{adj} \) to form \( A_{k+1}^{-1} \).

Now, consider the \( n \times n \) semidefinite quadratic form \( A \). If \( A \) is not null, then there is at least one nonzero diagonal element. This method will search down the diagonal for the first nonzero element
and use it as $A_1$. Then a partial row and column is attached to $A_1$ to form the nonsingular symmetric submatrix $A_2$, provided the rank($A$) $\geq 2$. The permutation matrices $P$ and $P'$ reflect which rows and columns are being used to construct the submatrices $A_k$. The bordering process continues until at some step, say the $k^{th}$ step, none of the remaining rows and columns from $A$ can be adjoined to $A_{k-1}$ to form a nonsingular $k \times k$ submatrix $A_k$. Since the $\det(A_{k-1}) \neq 0$, by Theorem 3.2.1 $A_{k-1}$ must be contained in a rank($A$) $\times$ rank($A$) nonsingular minor of $A$. If no partial row and column can be attached to $A_{k-1}$ to yield a $k \times k$ nonsingular minor, then $A_{k-1}$ is not contained in a $k \times k$ nonsingular minor hence, the rank($A$) $= k-1$ and $A_{k-1}$ is a rank($A$) $\times$ rank($A$) nonsingular minor of $A$. This is how the bordering process builds the nonsingular minor $A_{11}$ of Equation 3.4.1.

Multiple modulus residue arithmetic will be used to find and compute the $A_k^{adj}$ and $\det(A_k)$ at each step $k = 1, 2, \ldots, \text{rank}(A)$. Equation 3.4.1 is equivalent to Equation 3.2.1 with $Q = P'$. Consequently, a multiple modulus base, $\{p_1, p_2, \ldots, p_s\}$ with a product modulus

$$M = \prod_{i=1}^{s} p_i$$

such that

$$M > 2|\det(\text{any minor of } A)|$$

will guarantee the $|\det(A_k)|_{p_i} \neq 0$ for all $i = 1, 2, \ldots, s$ unless $\det(A_k) = 0$. This gives a stopping criterion for using multiple modulus residue arithmetic on the bordering process. If at step $k$ none of the
remaining partial rows and columns of $A$ can be added to $A_{k-1}$ such that $|\det(A_k)|_{p_i} \neq 0$ for at least one $p_i$, $i = 1, 2, \ldots, s$, then the process stops concluding with $\text{rank}(A) = k-1$ and forming

$$A_k^* = \frac{1}{\det(A_{k-1})} \prod \begin{bmatrix} A_{k-1}^{\text{adj}} & \phi \\ \phi & \phi \end{bmatrix}_p,$$

where $\det(A_{k-1})$ and $A_{k-1}^{\text{adj}}$ are determined from their multiple modulus residue representations. It is important to understand that the quantities $|\det(A_k)|_{p_i}$ and $|A_{k-1}^{\text{adj}}|_{p_i}$, $i = 1, 2, \ldots, s$, and not the quantities $\det(A_k)$ and $A_{k-1}^{\text{adj}}$ are carried from step to step. It is not until the $\text{rank}(A)$ has been determined that multiple modulus residue representations are combined.

One important detail in this method warrants more explanation. The multiple modulus residue arithmetic bordering process is always working directly with the original matrix $A$, so it is important not to misinterpret some quantity modulo $p$ equal to zero. The $|\det(A_{k-1})|_p = 0$ does not imply $|\det(A_k)|_p = 0$, $|A_{k-1}^{\text{adj}}|_p = \phi$ or $|A_k^{\text{adj}}|_p = \phi$. When $|\det(A_{k-1})|_p = 0$ care needs to be taken in computing $|A_k^{\text{adj}}|_p$. This is because according to Equation 3.4.5.
In order to compute $A_k^{\text{adj}}$ using residue arithmetic modulo $p$ on this bordering equation, the inverse of $\det(A_{k-1})$ modulo $p$ must exist. If $|\det(A_{k-1})|_p \neq 0$ then

$$
\begin{vmatrix}
\det(A_k)^{\text{adj}}_{k-1} + A_{k-1}^{\text{adj}} b_{k} b_{k}^{\text{adj}} A_{k-1}^{\text{adj}} \\
-\det(A_{k-1}) b_{k}^{\text{adj}} A_{k-1}^{\text{adj}} \\
\end{vmatrix}
= \begin{vmatrix}
\det(A_{k-1})^{-1}(p) \det(A_k)^{\text{adj}}_{k-1} + \\
\det(A_{k-1})^2 \\
\end{vmatrix} 
$$

(Note that if $A_k$ is integral then $A_k^{\text{adj}}$ is integral, so $\det(A_{k-1})|\det(A_k)^{\text{adj}}_{k-1} + A_{k-1}^{\text{adj}} b_{k} b_{k}^{\text{adj}} A_{k-1}^{\text{adj}}|_p$.) If $|\det(A_{k-1})|_p = 0$, the bordering equations cannot be used to compute $|A_k|_p$. In this situation, $|A_k|_p$ will be found by using multiple modulus residue extended Gauss-Jordan elimination on $|A_k|_p$. Unfortunately, there is no rule which says if $|\det(A_{k-1})|_p = 0$ and $|\det(A_k)|_p = 0$ then $|A_k|_p = \phi$. Augmenting $A_{k-1}$ with an additional row and column does not necessarily introduce another rank deficiency into $|A_k|_p$. When $|\det(A_{k-1})|_p = 0$ and $|\det(A_k)|_p = 0$ it is possible for $\text{rank}(|A_{k-1}|_p) = k-2$ with $|A_{k-1}|_p \neq 0$ and for $\text{rank}(|A_k|_p) = k-1$ with $|A_k|_p \neq 0$. Consider the next example.
Example 3.4.1 \hspace{1em} \text{Let} \ A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 12 & 3 \\ 0 & 3 & 1 \end{bmatrix} \text{ and } p = 3.

\begin{align*}
A_1 &= \begin{bmatrix} 1 \end{bmatrix} \\
|A_1|_3 &= \begin{bmatrix} 1 \end{bmatrix}, \quad |\text{det}(A_1)|_3 = 1, \quad |A_1^{\text{adj}}|_3 = \begin{bmatrix} 1 \end{bmatrix}
\end{align*}

\begin{align*}
A_2 &= \begin{bmatrix} 1 & 0 \\ 0 & 12 \end{bmatrix} \\
|A_2|_3 &= \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad \text{rank}(|A_2|_3) = 1, \quad |\text{det}(A_2)|_3 = 0,
\end{align*}

|A_2^{\text{adj}}|_3 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}

\begin{align*}
A_3 &= A \\
|A_3|_3 &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \text{rank}(|A_3|_3) = 2, \quad |\text{det}(A_3)|_3 = 0,
\end{align*}

|A_3^{\text{adj}}| = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.

There is one consistent result in this multiple modulus residue arithmetic bordering method. If $|A_{k-1}|_p = \phi$, then $|A_k^{\text{adj}}|_p = |A_{k+1}^{\text{adj}}|_p = \ldots = \phi$.

In the multiple modulus residue arithmetic Gaussian elimination method, results were compared across moduli in the multiple modulus base to determine the rank(A) and to find a rank(A) x rank(A) nonsingular minor of A. This minor was then inverted and the multiple
modulus residue representations of the component parts of the inverse were combined to form $A^-$. The bordering process also compares results across the moduli in the base. With the bordering algorithm if $|\det(A_{k_i})|_{p_i} = 0$ for all $i = 1, 2, \ldots, s$, then the $\text{rank}(A) = k-1$ and $A_{k-1}$ is a $(k-1)x(k-1)$ nonsingular minor of $A$. The advantage of bordering process over the Gaussian elimination method is that at the time a $\text{rank}(A) \times \text{rank}(A)$ nonsingular minor of $A$, say $A_{k-1}$, has been located, the $|\det(A_{k-1})|_{p_i}$ and the $|A_{k-1}^{\text{adj}}|_{p_i}$ for $i = 1, 2, \ldots, s$ are also known. The only thing left to do is to combine these multiple modulus residue representations to form $\det(A_{k-1})$ and $A_{k-1}^{\text{adj}}$. Before these multiple modulus residue representations are combined, the product modulus bound can be refined such that

$$M^* > 2 \max\{|\det(A_{k-1})|, \max_{i,j}(A_{i,j}^{(k-1)})\},$$

where $(A_{i,j}^{(k-1)})$ are the cofactors of $A_{k-1}$. A subset (not necessarily a proper subset) of the original multiple modulus base, say $\{p_1, p_2, \ldots, p_h\}$, for which $\prod_{i=1}^{h} p_i \geq M^*$ can be selected. Only those results modulo $p_i$, $i = 1, 2, \ldots, h$ need be included in this final step of combining the multiple modulus residue representations to form $\det(A_{k-1})$ and $A_{k-1}^{\text{adj}}$. With these quantities, the product

$$A^- = \frac{1}{\det(A_{k-1})} P^T \begin{bmatrix} A_{k-1}^{\text{adj}} & \phi \\ \phi & \phi \end{bmatrix} P$$

is computed. The disadvantage of the multiple modulus residue arithmetic
bordering process is that $A$ needs to be a quadratic form.

Now, the multiple modulus residue arithmetic bordering algorithm will be given. It is assumed $A$ is an nxn positive semidefinite symmetric matrix.

3.4.2 Bordering algorithm for a symmetric semidefinite quadratic form, $A = X'X$, using multiple modulus residue arithmetic

1. Compute Schinzel's bound for $M$ as in the Gaussian elimination algorithm. Choose the primes \{p_1, p_2, \ldots, p_s\} such that

$$
\prod_{i=1}^{s} p_i > M \geq 2|\text{det(any minor of } A)|
$$

These primes make up the multiple modulus base.

2. Find a nonzero diagonal element.

Set $i = 1$, $P = I$, $W = A$.

i) If $a_{ii} \neq 0$ go to (iii). Else continue.

ii) Set $i = i + 1$, go to (i).

iii) Interchange rows 1 and $i$ in $A$ and $P$ and interchange columns 1 and $i$ in $A$.

iv) Compute $|\frac{a_{i1}}{p_q}|$ for $q = 1, 2, \ldots, s$.

Now $A_1 = \frac{a_{11}}{p_1}$,

$$
\text{det}(A_1) \sim \{|a_{11}|_{p_1}, \ldots, |a_{11}|_{p_s}\},
$$

and $|\frac{a_{i1}}{p_q}| = 1$ for all $q = 1, 2, \ldots, s$. 

3. The Bordering Process

\[ A_k = \begin{bmatrix} A_{k-1} & b_k \\ \hat{b}_k & a_{kk} \end{bmatrix} \]

At the end of this algorithm \( A = PWP' \) and \( \text{rank} = \text{rank}(A) \).

Set \( k = 1 \).

i) \( k = k+1, \ r = k \).

ii) \( b_k' = (a_{1k}, a_{2k}, \ldots, a_{(k-1)k}) \) compute \( |b_k'_{p_q}| \) and \( |a_{kk}'_{p_q}| \) for \( q = 1, 2, \ldots, s \).

iii) Compute \( v_q = \left\{ \begin{array}{ll} \frac{|A_{k-1}'_{p_q}|}{|b_k'_{p_q}|}, & |A_{k-1}'_{p_q}| \neq \phi \text{ for } q = 1, 2, \ldots, s \\ \phi, & |A_{k-1}'_{p_q}| = \phi \end{array} \right. \)

iv) Compute \( |\text{det}(A_k)'_{p_q}| = \left\{ \begin{array}{ll} |\text{det}(A_{k-1}'_{p_q})|_{p_q} |a_{kk}'_{p_q}|, & |A_{k-1}'_{p_q}| \neq \phi \\ 0, & |A_{k-1}'_{p_q}| = \phi \end{array} \right. \) for \( q = 1, 2, \ldots, s \).

v) If \( |\text{det}(A_k)'_{p_q}| \neq 0 \) for all \( q \) set \( q = 0 \) and go to (vii), else continue.

Note: If \( \text{det}(A_k) \sim \{0, 0, \ldots, 0\} \) then \( \text{det}(A_k) = 0 \).
vi) If \( r = n \), then exit with

\[
\begin{cases}
\det(A_{k-1}) & \ni \{ |\det(A_{k-1})|_{p_1}, \ldots, |\det(A_{k-1})|_{p_s} \} \\
A_{k-1}^{\text{adj}} & \ni \{ |A_{k-1}^{\text{adj}}|_{p_1}, \ldots, |A_{k-1}^{\text{adj}}|_{p_s} \} \\
\text{rank} & = k-1
\end{cases}
\]

Else \( r = r+1 \), interchange rows \( k \) and \( r \) in \( A \) and \( P \) and interchange columns \( k \) and \( r \) in \( A \). Go to (ii).

vii) \( q = q + 1 \).

If \( |A_{k-1}^{\text{adj}}|_{p_q} = \phi \) set \( |A_{k}^{\text{adj}}|_{p_q} = \phi \) and go to (xi).

If \( |A_{k-1}^{\text{adj}}|_{p_q} \neq \phi \) and \( |\det(A_{k-1})|_{p_q} = 0 \), do Gaussian elimination with extended elimination modulo \( p_q \) the upper left hand \( k \times k \) submatrix of \( |A|_{p_q} \), go to (xi). Else go to (viii).

viii) Compute \( X = \left| |\det(A_{k}^{\text{adj}})|_{p_q} (A_{k-1}^{\text{adj}})|_{p_q} + |v_q v'_q|_{p_q} \right|_{p_q} \).

ix) Compute \( \text{ADJ} = \{|\det(A_{k-1})|^{-1}(p_q) X|_{p_q} \} \).

x) Form \( |A_{k}^{\text{adj}}|_{p_q} = \begin{bmatrix} \text{ADJ} & -v_q \\ -v'_q & |\det(A_{k-1})|_{p_q} \end{bmatrix} \).

xi) If \( q < s \) go to (vii). Else go to (xii).
xii) If \( k < n \) go to (i). Else exit with

\[
\begin{align*}
\det(A) & \sim \{ |\det(A)|_{p_1}, \ldots, |\det(A)|_{p_q} \} \\
A_{\text{adj}} & \sim \{ |A_{\text{adj}}|_{p_1}, \ldots, |A_{\text{adj}}|_{p_q} \} \\
\text{rank} &= n
\end{align*}
\]

3. Refine the product modulus bound by computing Schinzel's bound, \( M^* \), for the determinant of the rank \( x \) rank minor and its cofactors. This minor is presently located in the upper left hand rank \( x \) rank submatrix of \( A \).

4. Choose the primes \( p_q, q = 1, 2, \ldots, h \), from those prime moduli in the multiple modulus base constructed at Step 1 such that

\[
\prod_{q=1}^{h} p_q > M^* \geq 2 \max_{i,j} \max \{|A_{(\text{rank } x \text{ rank})}|_{i,j} \}.
\]

5. Let \( M = \prod_{q=1}^{h} p_q \) and form \( |\det(A_{(\text{rank } x \text{ rank})})|_M \) and \( |A_{\text{adj}}^{(\text{rank } x \text{ rank})}|_M \) by combining the multiple modulus residue representation of the quantities.

6. Set \( \det(A_{(\text{rank } x \text{ rank})}) = \det(A_{(\text{rank } x \text{ rank})}) / M \)

and \( A_{\text{adj}}^{(\text{rank } x \text{ rank})} = A_{\text{adj}}^{(\text{rank } x \text{ rank})} / M \).

7. \( A^{-} = \frac{1}{\det(A_{(\text{rank } x \text{ rank})})} \left[ \begin{array}{cc}
A_{\text{adj}}^{(\text{rank } x \text{ rank})} & \phi \\
\phi & \phi
\end{array} \right] P \).
This concludes the bordering algorithm. It should be noted that when $A$ is an integral positive definite symmetric matrix, this multiple modulus residue arithmetic bordering process is also quite efficient for finding $A^{-1}$. Both of the algorithms in this section have been implemented in standard FORTRAN. The FORTRAN programs have been included in the Appendix and the actual implementation are discussed in more detail in Chapter 7.
4. MOORE-PENROSE INVERSE USING MULTIPLE MODULUS RESIDUE ARITHMETIC

The generalized inverse to be considered in this chapter was not discussed in Chapter 3. It is the Moore-Penrose inverse. By definition, the sxn matrix \( A^+ \) is the Moore-Penrose inverse of an nxs matrix \( A \) if all four of the following conditions hold,

1) \( AA^+ A = A \),
2) \( A^+ AA^+ = A^+ \),
3) \( (A^+ A)' = A^+ A \),
4) \( (AA^+)' = AA^+ \).

The Moore-Penrose inverse, \( A^+ \), is unique in the sense it is the only generalized inverse of \( A \) to satisfy these four properties. A reflexive generalized inverse satisfies the Conditions (i) and (ii). Reflexive generalized inverses have no uniqueness property. It was thought in Chapter 3 that perhaps the reason \( A^- \), the reflexive generalized inverse of an nxs integral matrix \( A \), could not be extracted directly from \( A^- (M) \), the reflexive generalized inverse of \( |A|_M \) over \( R(M) \), was due to the nonuniqueness property of reflexive generalized inverses. This chapter will address the similar problem of using multiple modulus residue arithmetic to find the unique Moore-Penrose inverse, \( A^+ \), of an nxs integral matrix \( A \), and more will be said about extraction of inverses.

Section 4.1 will approach this problem from a field theory point of view. Section 4.2 modifies and applies multiple modulus residue arithmetic to an integral matrix inversion method outlined by Stallings and Boullion (1972).
4.1 Existence of the Moore-Penrose Inverse Over Various Fields and Rings

The results in this section are actually included for the sake of completeness. Although the Moore-Penrose inverse, $A^+$, of an $n \times s$ integral matrix $A$, has a uniqueness property over the field of rational numbers, it does not have a specified unique characterization of the form

$$A^+ = \frac{1}{cf} A^*$$

where $cf$ is an integer and $A^*$ is integral. The same problems encountered in Section 3.1 when trying to extract $A^-$, the generalized inverse of $A$ over the field of rational numbers, from $A^{-}(M)$ occur when working with $A^+$ and $A^{+}(M)$ as is shown by the following.

The first thing to consider is the existence of $A_{(p)}^+$, $p$-prime, the Moore-Penrose inverse of $|A|^p$ over $G(p)$. The next theorem is due to Pearl (1968). Pearl's theorem is given with respect to an arbitrary field $F$, but to avoid confusion it will be stated in terms of $G(p)$, $p$-prime.

**Theorem 4.1.1** Let $A$ be an $n \times s$ integral matrix and $|A|^p$, $p$-prime, be the residue of $A$ modulo $p$. The Moore-Penrose inverse of $|A|^p$ over $G(p)$, $A^+(p)$, exists if and only if

$$\text{rank}(|A|^p) = \text{rank}(|A'A|^p) = \text{rank}(|AA'|^p).$$

The generalized inverse $A^-(p)$, $p$-prime, of Definition 3.1.1 always exists, but the Moore-Penrose inverse $A^+(p)$, $p$-prime, only exists when
the rank condition of Theorem 4.1.1 is satisfied. It should not be assumed that this rank condition holds for every $|A|_p$ over $G(p)$. Consider the following example.

**Example 4.1.1** Let $A = \begin{bmatrix} 5 & 0 & 0 \\ 1 & 3 & 0 \\ 0 & 0 & 7 \end{bmatrix}$. Show the rank condition of Theorem 4.1.1 does not hold for $|A|^p$ with $p = 5$.

- $\text{rank}(|A|_5) = \text{rank}\left(\begin{bmatrix} 0 & 0 & 0 \\ 1 & 3 & 0 \\ 0 & 0 & 2 \end{bmatrix}\right) = 2$,
- $\text{rank}(|A'A|_5) = \text{rank}\left(\begin{bmatrix} 1 & 3 & 0 \\ 3 & 4 & 0 \\ 0 & 0 & 4 \end{bmatrix}\right) = 2$,

but

- $\text{rank}(|AA'|_5) = \text{rank}\left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 4 \end{bmatrix}\right) = 1$.

The next theorem defines for which product moduli, $M = \prod p_i$, $p_i$-prime, there exists a Moore-Penrose inverse of $|A|^M$ over the ring generated by $M$, $R(M)$.

**Theorem 4.1.2** Let $A$ be an $n \times s$ integral matrix. Given a multiple modulus base $\{p_1, p_2, \ldots, p_s\}$ such that $A^+(p_i)$ exists for all $i = 1, 2, \ldots, s$, then $A^+(M)$, the Moore-Penrose inverse of $|A|^M$, $M = \prod p_i$, over $R(M)$, is produced by applying the Chinese Remainder Theorem to the $A^+(p_i)$, $i = 1, 2, \ldots, s$. 
Proof: We need to show \( A^+(M) \) satisfies the four Moore-Penrose inverse properties.

By Theorem 3.1.2, \( A^+(M) \) satisfies the first two Moore-Penrose inverse properties (the reflexive generalized inverse properties.)

We need to show

\[
\begin{align*}
\text{iii)} & \quad \left( A^+(M) \left| A\right|_M^\dagger \right)_M = \left. A^+(M) \left| A\right|_M \right)_M \\
\text{and iv)} & \quad \left( \left. A\right|_M A^+(M) \right)_M = \left. A\right|_M A^+(M) \right)_M
\end{align*}
\]

Now, \( A^+(M) \left|_{p_i} \right. = A^+(p_i) \), for \( i = 1, 2, \ldots, s \), since \( A^+(M) \) was constructed via the Chinese Remainder Theorem. Therefore, it suffices to show

\[
\begin{align*}
\left( A^+(M) \left| A\right|_M^\dagger \right)_M & = \left( A^+(p_i) \left| A\right|_{p_i}^\dagger \right)_{p_i} \\
\text{and} \quad \left( \left. A\right|_M A^+(M) \right)_M & = \left( \left. A\right|_{p_i} A^+(p_i) \right)_{p_i}
\end{align*}
\]

for all \( i = 1, 2, \ldots, s \).

Now,

\[
\begin{align*}
\left( A^+(M) \left| A\right|_M^\dagger \right)_M & = \left( \left. A\right|_M A^+\dagger(M) \right)_M \\
& = \left. A\right|_{p_i} A^+\dagger(p_i) \right)_{p_i} \\
& = \left( A^+(p_i) \left| A\right|_{p_i}^\dagger \right)_{p_i} \\
& = \left. A\right|_{p_i} A^+(p_i) \right)_{p_i}
\end{align*}
\]
And,
\[
\begin{vmatrix}
\begin{bmatrix}
|A|_M A^+(M)
\end{bmatrix}'
\end{vmatrix}_{M, p_1}
= \begin{vmatrix}
A^+(M)
|A|'_M
\end{vmatrix}_{M, p_1}
\]

\[
= \begin{vmatrix}
A^+(p_1)
|A|'_1
\end{vmatrix}_{p_1}
\]

\[
= \begin{vmatrix}
(|A|_{p_1} A^+(p_1))'
\end{vmatrix}_{p_1}
\]

\[
= \begin{vmatrix}
|A|_{p_1} A^+(p_1)
\end{vmatrix}_{p_1}
\].

Hence, the Theorem.

It is not clear to us if given \(A^+(M)\) for a sufficiently large product modulus \(M\) that \(A^+\), the Moore-Penrose inverse of \(A\) over the field of rational numbers, could be extracted from \(A^+(M)\). Even if this is possible a method based on these results would always have to work around the fact that \(A^+(p)\) does not exist over \(\mathbb{Q}(p)\) for all primes \(p\). To determine this existence, the rank(\(|A|_p\)), the rank(\(|A'A|_p\)) and the rank(\(|AA'|_p\)) would need to be computed. Consequently, our attention has been directed away from this approach to the multiple modulus residue arithmetic Moore-Penrose inverse problem. The next section considers applying multiple modulus residue arithmetic to an integral Moore-Penrose inversion technique in a way that leads to a solution.
4.2 Multiple Modulus Residue Arithmetic Moore–Penrose Inverse

The method to be modified in this section is an integral Moore–Penrose matrix inversion technique developed by Stallings and Bouillon (1972), based on an algorithm due to Decell (1965). We will first discuss Decell's algorithm for finding the Moore–Penrose inverse, $A^+$, of a real $n \times s$ matrix $A$. Then the case of finding $A^+$ when $A$ is an $n \times s$ integral matrix will be considered. Once the integral method has been established, a multiple modulus bound for the process will be determined and multiple modulus residue arithmetic will be applied.

The algorithm given by Decell (1965) is based on the following theorem.

**Theorem 4.2.1** Let $A$ be a real $n \times s$ matrix and let

$$f(\lambda) = (-1)^n(a_0 \lambda^n + a_1 \lambda^{n-1} + \ldots + a_k \lambda^{n-k} + \ldots + a_n)$$

be the characteristic polynomial of $AA^t$. If $k \neq 0$ is the largest integer such that $a_k \neq 0$ then

$$A^* = -\frac{1}{a_k} A'[ (AA^t)^{k-1} + a_1 (AA^t)^{k-2} + \ldots + a_{k-1} I ]$$

$$= -\frac{1}{a_k} A'R_{k-1}.$$

If $k = 0$ then $A^* = \phi$. 


As a consequence of this theorem, Decell (1965) was able to modify a modification of Leverrier's method given by Faddeev and Faddeeva (1963). Decell's Algorithm computes $-a_k$ and $B_{k-1}$ in the following way.

\textbf{Algorithm 4.2.1} \quad A \text{ is an } n \times n \text{ real matrix.}

\begin{align*}
A_0 &= \phi \quad q_0 = -1 \quad B_0 = I \\
A_1 &= AA' \quad q_1 = \text{tr}(A_1) \quad B_1 = A_1 - q_1 I \\
A_2 &= AA'B_1 \quad q_2 = \frac{1}{2} \text{tr}(A_2) \quad B_2 = A_2 - q_2 I \\
A_3 &= AA'B_2 \quad q_3 = \frac{1}{3} \text{tr}(A_3) \quad B_3 = A_3 - q_3 I \\
& \vdots \\
A_{k-1} &= AA'B_{k-2} \quad q_{k-1} = \frac{1}{(k-1)} \text{tr}(A_{k-1}) \quad B_{k-1} = A_{k-1} - q_{k-1} I \\
A_k &= AA'B_{k-1} \quad q_k = \frac{1}{k} \text{tr}(A_k) \quad B_k = A_k - q_k I .
\end{align*}

In this algorithm, $k = \text{rank}(A)$ and $q_i = -a_i$, $i = 1, 2, \ldots, k$, from Theorem 4.2.1. It is obvious $k = \text{rank}(A)$ since $(n-k)$ is the smallest power of $\lambda$ in the characteristic polynomial, $f(\lambda)$, of $AA'$. This means $F(\lambda) = \lambda^{n-k} h(\lambda)$ where $h(\lambda)$ is a polynomial of degree $k$. Thus, there exists $k$ nonzero eigenvalues of $AA'$ and $\text{rank}(AA') = \text{rank}(A) = k$. The fact that $q_i = -a_i$ is not so obvious. The proof of this was given by Faddeev and Faddeeva (1963).

Some careful observations show that if $A$ is an $n \times n$ integral matrix, the use of Decell's Algorithm to compute $A^+$ is an integral
method. If $A$ is integral, then the coefficients, $a_i$, $i = 1, 2, \ldots, k$, of the characteristic polynomial of $AA'$ in Theorem 4.2.1 are integers. This is because the characteristic polynomial of $AA'$ is formed by setting $\det(\lambda I - AA')$ equal to zero. In Decell's Algorithm, the only place where division occurs is in the computation of the $q_i$. But, $q_i = -a_i$, which means the result of this division is integral. Therefore, given an $n \times s$ integral matrix $A$, we have an integral method for computing $A^+$, the Moore-Penrose inverse of $A$ over the field of rational numbers.

Residue arithmetic modulo $p$, prime, can be applied to Decell's Algorithm (Algorithm 4.2.1) provided $p > k = \text{rank}(A)$. This is done as follows.

**Algorithm 4.2.2**  
$A$ is an $n \times s$ integral matrix and $A_p$, prime, is the residue of $A$ modulo $p$. 

| $A_0|_p$ = $\phi$ | $q_0|_p$ = $p-1$ | $B_0|_p$ = $I$ |
| $A_1|_p$ = $|AA'|_p$ | $q_1|_p$ = $|\text{tr}(A_1)|_p$ | $B_1|_p$ = $|A_1-q_1I|_p$ |
| $A_2|_p$ = $|AAB_1|_p$ | $q_2|_p$ = $2^{-1}(p)\text{tr}(A_2)|_p$ | $B_2|_p$ = $|A_2-q_2I|_p$ |

| $A_p|_p$ | $A_pB_1|_p$ | $|A_pA_pB_1|_p$ | $|2^{-1}(p)\text{tr}(A_2)|_p$ | $|A_2|_p-q_2I|_p|_p$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
The restriction of \( p > k \) is necessary to insure all of the \( |q_i|_p \), \( i = 1, 2, \ldots, k \), can be computed (i.e., \( i^{-1}(p) \) must exist for all \( i \)).

Stallings and Boullion (1972) give a complete discussion of using single modulus residue arithmetic on Decell's algorithm to compute the Moore-Penrose inverse, \( A^+ \), of an integral nxn matrix \( A \). In the single modulus residue arithmetic case, if the modulus \( p \), \( p \)-prime, is such that

\[
|A_{k-1}|_p = |A^t B_{k-2}|_p \quad |q_{k-1}|_p = |(k-1)^{-1}(p)tr(A_{k-1})|_p \quad |B_{k-1}|_p = |A_{k-1}|
\]

\[
= \left| A | A^t | B_{k-2} \right|_p \quad = \left| (k-1)^{-1}(p)tr(|A_{k-1}|_p) \right|_p \quad = \left| A_{k-1} \right|_p - q_{k-1} |I|_p
\]

\[
|A_k|_p = |A^t B_{k-1}|_p \quad |q_{k}|_p = |k^{-1}(p)tr(A_k)|_p
\]

\[
= \left| A | A^t | B_{k-1} \right|_p \quad = \left| k^{-1}(p)tr(|A_k|_p) \right|_p \quad |B_k|_p = |A_k - q_k |I|_p
\]

\[
= \left| -a_k \right|_p \quad = \left| A_k | -q_k |I|_p \right|_p
\]

\( (\text{4.2.1}) \)

The restriction of \( p > k \) is necessary to insure all of the \( |q_i|_p \), \( i = 1, 2, \ldots, k \), can be computed (i.e., \( i^{-1}(p) \) must exist for all \( i \)).

\( \text{4.2.1} \)

\[
p > 2 \max \{ |a_k|, \max_{i,j} |b_{ij}| \}
\]

where \( B_{k-1} = (b_{ij}) \), \( q_k = -a_k \), and \( k = \text{rank}(A) \), then

\[
-a_k = /-a_k/ = /q_k/ \quad \text{and} \quad B_{k-1} = /B_{k-1}/. \quad \text{Stallings and Boullion}
\]

\[
p > 2 \max \{ m^n, n(n-k+1)m^{n-1} \}
\]

where

\[
m = \min \{ \text{tr}(AA') , \|AA'\| \}
\]

for any norm \( ||\cdot|| \), satisfies Equation 4.2.1.
A major drawback to Decell's Algorithm (Algorithm 4.2.1) and Stallings and Boullion's single modulus residue arithmetic application to Decell's Algorithm (Algorithm 4.2.2) is that the rank(A) = k is required to be known at the beginning of the process. This way the exact number of steps to be taken in the algorithm is known a-priori. As in the reflexive generalized matrix inversion techniques, we would like to be able to compute $A^+$, the Moore-Penrose inverse of an nxn integral matrix A, using multiple modulus residue arithmetic without requiring the rank(A) to be known at the start of the problem.

The next theorem gives a stopping criterion for Decell's Algorithm which does not require knowledge of the rank(A) a-prior.

**Theorem 4.2.2** An integer $k$ satisfies Decell's Theorem (Theorem 4.2.1) if and only if $A_k \neq \phi$ and $A_{k+1} = \phi$ where $A_k$ and $A_{k+1}$ are the matrices in Decell's Algorithm (Algorithm 4.2.1).

**Proof:** Suppose $k$ satisfies Decell's Theorem, then

$$A_{k+1} = AA'B_k$$

$$= AA'(AA'B_{k-1} - q_k I)$$

$$= AA'(A(q_k A^+ - q_k I)$$

$$= q_k (AA'A^+ - AA')$$

$$= q_k (AA'A^+ - AA')$$

$$= q_k (AA' - AA')$$

$$= \phi ,$$
and since \( q_k = -a_k = \frac{1}{k} \text{tr}(A_k) \neq 0 \), then \( A_k \neq \phi \).

Suppose \( A_k \neq \phi \) and \( A_{k+1} = \phi \), then \( q_{k+1} = q_{k+2} = \ldots = 0 \), where \( q_i = -a_i \) of the characteristic polynomial, \( f(\lambda) \), of \( AA' \) in Decell's Theorem (Theorem 4.2.1). Also,

\[
A_{k+2} = A_{k+3} = \ldots = \phi.
\]

We need to show \( k \) satisfies Decell's Theorem. It suffices to show \( q_k = -a_k \neq 0 \). Suppose \( q_k = 0 \), then there exists some \( k^* < k \) such that

\[
q_{k^*} = -a_{k^*} \neq 0 \text{ and } q_{k^*} = q_{k^*+1} = \ldots = q_k = q_{k+1} = \ldots = 0.
\]

Therefore, \( k^* \) satisfies Decell's Theorem which implies

\[
A_{k^*} \neq \phi \text{ and } A_{k^*} = A_{k^*+1} = \ldots = A_k = A_{k+1} = \ldots = \phi.
\]

This is a contradiction to the assumption \( A_k \neq \phi \). Hence, the theorem.

Employing this theorem results in no need for additional information about \( A \) in order to use Decell's Algorithm for finding \( A^+ \). The process continues from step to step provided \( A_i \neq \phi \). When the first null matrix, say \( A_{k+1} \), is encountered the process stops. At this point, \(-a_k\) and \( E_{k-1}\) can be found by examining the results at steps \( k \) and \( k-1 \) respectively. The next example illustrates Decell's Algorithm in conjunction with the stopping criterion of Theorem 4.2.2.
Example 4.2.1

\[ A = \begin{bmatrix} 5 & 1 & 5 \\ 1 & 3 & 1 \\ 5 & 1 & 5 \end{bmatrix} \]

Use Decell's Algorithm to find \( A^+ \).

\[ A_0 = \phi \quad q_0 = -1 \quad B_0 = I \]

\[ A_1 = AA' = \begin{bmatrix} 51 & 13 & 51 \\ 13 & 11 & 13 \\ 51 & 13 & 51 \end{bmatrix} \quad q_1 = \text{tr}(A_1) = 113 \quad B_1 = A_1 - q_1 I = \begin{bmatrix} -62 & 13 & 51 \\ 13 & -102 & 13 \\ 51 & 13 & -62 \end{bmatrix} \]

\[ A_2 = AA'B_1 = \begin{bmatrix} -392 & 0 & -392 \\ 0 & -784 & 0 \\ -390 & 0 & -392 \end{bmatrix} \quad q_2 = \frac{1}{2} \text{tr}(A_2) = -784 \quad B_2 = A_2 - q_2 I = \begin{bmatrix} 392 & 0 & -392 \\ 0 & 0 & 0 \\ -392 & 0 & 392 \end{bmatrix} \]

\[ A_3 = AA'B_2 = \phi \rightarrow \text{STOP} \]

Therefore, \( \text{rank}(A) = 2 \) and

\[ A^+ = \frac{1}{q_2} A'B_1 = \frac{1}{-784} \begin{bmatrix} -42 & 28 & -42 \\ 28 & -280 & 28 \\ -42 & 28 & -42 \end{bmatrix} \cdot \]

Now consider the residue arithmetic application of Decell's Algorithm (Algorithm 4.2.2). In this method, \( k = \text{rank}(A) \) was used for a stopping criterion of the algorithm and was also included in the computation of the bound for the prime \( p \) of Equation 4.2.2. Assuming \( k > 1 \), then \( n > (n-k+1) \) and the inequality of Equation 4.2.2 can be rewritten as
\[ p > 2 \max\{m^n, n^2m^{n-1}\} \geq 2 \max\{m^n, n(n-k+1)m^{n-1}\} \]

where

\[ m = \min\{\text{tr}(AA'), ||AA'||\} . \]

Therefore, if \( p \)-prime satisfies Equation 4.2.3, then residue arithmetic modulo \( p \) can be applied to Decell's Algorithm, as in Algorithm 4.2.2, stopping when the first null \( |A_i|_p \) matrix is encountered. There is no problem in forming \( |q_i|_p = i^{-1}(p)\text{tr}(|A_i|)_p \), \( i = 1, 2, \ldots, k \), since according to Equation 4.2.3 \( p > n \), therefore, \( i^{-1}(p) \) exists for all \( i \leq n \) and \( n \leq \text{rank}(A) = k \).

The bound of Equation 4.2.3 may be quite conservative. For Example 4.2.1, a prime modulus of \( p > 2370910 \) would be necessary to satisfy Equation 4.2.3; whereas,

\[ p > 2 \max\{|a_k|, \max_{i,j}|b_{ij}|\} = 2 \max\{784, 102\} = 1568, \]

would be sufficiently large to find the Moore–Penrose inverse of the matrix in that example by applying arithmetic modulo \( p \) to Decell's Algorithm. Obviously, the \( \max\{|a_k|, \max_{i,j}|b_{ij}|\} \), where \( B_{k-1} = (b_{ij}) \), is not known at the start of the problem, so the inequality of Equation 4.2.3 must be used to find an appropriate modulus. But, as in the case of the simple matrix in Example 4.2.1 this bound can get very large. In Example 4.2.1, if \( p > 2370910 \) then \( p^2 > 5.6 \times 10^{12} \). Suppose Algorithm 4.2.2 was implemented in fixed-point arithmetic on a computer like the IBM 370, then arithmetic modulo \( p \) with \( p^2 > 5.6 \times 10^{12} \)
would not be able to be done without exceeding the permissible fixed-point number range of \([-2^{31}, 2^{31}] \approx [-2.2 \times 10^9, 2.2 \times 10^9]\).

This problem can be rectified by applying multiple modulus residue arithmetic to Decell's Algorithm. If a multiple modulus base, 
\(\{p_1, p_2, \ldots, p_h\}\), is chosen such that \(p_i > n\) of all \(i = 1, 2, \ldots, h\) and the product modulus \(M = \prod_{i=1}^{h} p_i\) satisfies

\[
M > 2 \max\{m^n, n^2m^{n-1}\}
\]  

(4.2.4)

where \(m = \min\{\text{tr}(AA'), ||AA'||\}\), then \(-a_k\) and the matrix \(B_{k-1}\) of Decell's Algorithm will have unique multiple modulus residue representations with respect to this multiple modulus base. According to Theorem 4.2.2, Decell's Algorithm stops at step \(k+1\) if the matrix \(A_{k+1}\) is the first null \(A_i\) matrix of the algorithm. If the product modulus \(M = \prod_{i=1}^{h} p_i\) satisfies Equation 4.2.4, then \(A_{k+1} = \phi\) if and only if \(|A_{k+1}|_{p_i} = \phi\) for all \(i = 1, 2, \ldots, h\). Therefore, we have a multiple modulus residue arithmetic stopping criterion for Decell's Algorithm which is not based on knowing the rank \((A)\) a-priori. It is important that \(p_i > n\) for all \(i = 1, 2, \ldots, h\). This is to insure the existence of \(k^{-1}(p_i)\) for the possible \(k = 1, 2, \ldots, n\). For Example 4.2.1, a multiple modulus base of \(\{5, 7, 11, 13, 17, 19, 23\}\) gives a product modulus \(m = 37182145\) which satisfies Equation 4.2.4.

To help keep the already conservative product modulus bound of Equation 4.2.4 as small as possible, the multiple modulus residue
arithmetic process will be applied to an nxs integral matrix $A$ where $n \leq s$. If $n > s$ then the Moore–Penrose inverse of $A'$, $(A')^+ = (A^+)'$, will be formed.

Once the multiple modulus base has been chosen, the application of multiple modulus residue arithmetic to Decell’s Algorithm is done as follows. First, note that if at step $j \leq k-1$, $k = \text{rank}(A)$, the matrix $|A_j|_p = \phi$, $p$-prime, then $|A_{j+1}|_p = |A_{j+2}|_p = \ldots = |A_k|_p = |A_{k+1}|_p = \phi$, $|q_j|_p = |q_{j+1}|_p = \ldots = |q_k|_p = 0$, and $|B_j|_p = |B_{j+1}|_p = \ldots = |B_{k-1}|_p = \phi$.

This means, for a given prime in the multiple modulus base, at the step where the first null $|A_j|_p$ matrix is encountered, it can be assumed that all future steps will yield zeros and null matrices. Suppose the multiple modulus base is $\{p_1, p_2, \ldots, p_h\}$. Arithmetic modulo $p_i$ is applied to Decell’s Algorithm as in Algorithm 4.2.2 for each prime in the multiple modulus base. For each prime, $p_i$, in the base, the number of steps, say $k_i$, completed before encountering the first null $|A_j|_{p_i}$ matrix is recorded and the quantities $|q_k|_{p_i}$ and $|B_{k-1}|_{p_i}$ are saved. Then $k = \text{rank}(A)$ is determined by

$$k = \max \{k_i\}.$$  

If for a given prime, $p_i$, $k_i < k$ then $|a_{k}|_{p_i}$ is set to zero, if $k_i < k-1$ then $|B_{k-1}|_{p_i}$ is set to the null matrix. Otherwise, if
the multiple modulus residue arithmetic representations of $-a_k$ and $B_{k-1}$ are known and can be combined to form $|-a_k|_M$ and $|B_{k-1}|_M$ where $M = \prod_{i=1}^{h} p_i$. Then $-a_k = -a_k / M$ and $B_{k-1} = B_{k-1} / M$. This multiple modulus residue arithmetic application to Decell's algorithm is outlined below.

**Algorithm 4.2.3**

Given an nxn integral matrix $A$ with $n \leq s$, this multiple modulus residue arithmetic method will compute the Moore-Penrose inverse of $A$ over the field of rational numbers. Requiring $n \leq s$ is not a severe restriction because if $n > s$ then $(A^\prime)^+$ can be computed where $(A^\prime)^+ = (A^+)^\prime$.

1. Compute the product modulus bound to satisfy Equation 4.2.4 as

$$M = \begin{cases} 2^m n^2, & \text{if } m \geq n^2 \\ 2^{n^2} m^{n-1}, & \text{if } m < n^2 \end{cases}$$

where $m = \min\{\text{tr}(AA^\prime), ||AA^\prime||\}$.

2. Choose a multiple modulus base, $\{p_1, p_2, \ldots, p_h\}$, $p_i$-prime, such that $\prod_{i=1}^{h} p_i \geq M$ and $p_i > n$ for all $i = 1, 2, \ldots, h$. 

$k_i = k, |a_k|_{p_i} = |q_k|_{p_i}$ and $|B_{k-1}|_{p_i} = |B_{k-1}|_{p_i}$. At this point
3. For each $p_i$, $i = 1, 2, \ldots, h$, apply residue arithmetic modulo $p_i$ to Decell's Algorithm, as in Algorithm 4.2.2, using the stopping criterion of Theorem 4.2.2. Compute $k_{p_i}$, the number of steps completed before the first null $|A_j|_{p_i}$ matrix is encountered. Save the quantities $k_{p_i}$, $|q_k|_{p_i} = \frac{|-a_k|_{p_i}}{p_i}$ and $|B_{k-1}|_{p_i}$.

4. Compute $k = \text{rank}(A) = \max \{k_{p_i}\}$.

5. Form the multiple modulus residue representations of $-a_k$ and $B_{k-1}$. For each $i = 1, 2, \ldots, h$, set
   
   $$\frac{|-a_k|_{p_i}}{p_i} = \begin{cases} \frac{|-a_k|_{p_i}}{p_i}, & \text{if } k_{p_i} = k \\ 0, & \text{if } k_{p_i} < k \end{cases}$$

   and
   
   $$\frac{|B_{k-1}|_{p_i}}{p_i} = \begin{cases} \frac{|B_{k-1}|_{p_i}}{p_i}, & \text{if } k_{p_i} \geq k-1 \\ \phi, & \text{if } k_{p_i} < k-1 \end{cases}$$

6. Combine the multiple modulus residue representations from Step 5 to form $\frac{|-a_k|_M}{M}$ and $\frac{|B_{k-1}|_M}{M}$ where $M = \Pi_{i=1}^{h} p_i$. Then extract $-a_k = -a_k/M$ and $B_{k-1} = B_{k-1}/M$.

7. Form the Moore-Penrose inverse of $A$ as
   
   $$A^+ = \frac{1}{-a_k} A'B_{k-1}.$$
Algorithm 4.2.3 computes an error free Moore-Penrose inverse of an integral nxn matrix. Compared to the multiple modulus residue arithmetic generalized matrix inversion algorithms given in Chapter 3, this method is simpler from a computational point of view but more sophisticated from a theoretical point of view. Even in the event $A$ is an integral nxn nonsingular matrix, Algorithm 4.2.3 provides a very nice multiple modulus residue arithmetic nonsingular matrix inversion process. Algorithm 4.2.3 has been implemented in standard Fortran. The Fortran program is included in the Appendix and the implementation is discussed in Chapter 7.
5. LINEAR LEAST SQUARES SOLUTIONS USING MULTIPLE MODULUS RESIDUE ARITHMETIC

Computing matrix inverses is of slightly indirect interest to Statisticians, but computing estimates of parameters in linear models under the least squares criterion for goodness of fit is certainly of direct interest. This chapter deals with error-free computation of least squares estimates.

Consider the General Linear Model (GLM)

\[ y = X\beta + \varepsilon \]  

where

- \( y \) is an nx1 vector of observed values;
- \( X \) is an nxs matrix of fixed known numbers;
- \( \beta \) is a sx1 vector of unobservable parameters defined in a parameter space \( \Omega_\beta \);
- \( \varepsilon \) is an nx1 unobservable random vector such that \( E(\varepsilon) = \phi \) and \( E(\varepsilon\varepsilon') = \sigma^2 I \).

The Best Linear Unbiased Estimator (BLUE) of \( \beta \) is given by the method of least squares. The method of least squares minimizes the sum of squares

\[ (y - X\beta)'(y - X\beta) \]

with respect to \( \beta \). Any \( \hat{\beta} \) which minimizes this sum of squares is a solution to the normal equations

\[ X'X\hat{\beta} = X'y. \]
The normal equations are a consistent set of equations and the class of solutions for \( \beta \) is

\[
\{ b : b = (X'X)^{-1}X'y + (I - (X'X)^{-1}X'X)z, \]

for any generalized inverse of \( X'X \) and \( z \) an arbitrary vector with entries over the field of real numbers.

It can be shown by minimizing

\[
(\phi - Ib)'(\phi - Ib)
\]

with respect to \( X'Xb = X'y \) that \( \hat{b} = X'^+t \hat{y} \) is the shortest vector (minimum Euclidean norm) solution to the least squares minimization problems. The General Linear Model (GLM) and the least squares properties just outlined can be found in Graybill (1976) or other texts on linear models. The least squares solutions for \( \beta \) of the GLM to be discussed in this section are the ones usually considered,

\[
b_1 = (X'X)^{-1}X'y \quad \text{and} \quad b_2 = X'^+y. \tag{5.2}
\]

(Keep in mind if \( X'X \) is nonsingular then \( (X'X)^{-1} = (X'X)^{-1} \).) So we see estimating \( \beta \) in the GLM boils down to computing matrix inverses.

Suppose the entries in the vector \( y \) and the matrix \( X \) of the GLM, Equation 5.1, are rational numbers. This is an extremely realistic supposition because given data (i.e., \( y \) and \( X \)) a least squares solution for \( \beta \) is usually found with the aid of a computer. Numbers entered on the computer are always terminating decimals, hence they are rational. If \( y \) and \( X \) are rational then the least squares solutions
for $\beta$ given in Equation 5.2 are also rational. Let

$$y = \frac{1}{c_f y^*} \quad \text{and} \quad x = \frac{1}{c_f x^*}$$

(5.3)

where $c_f y, c_f x \in \mathbb{Z}$ and $y^*$ and $x^*$ are integral.

Then,

$$b_1 = c_f x (x^*'x^*)^{-1} \frac{1}{c_f x^*} x^* \frac{1}{c_f y^*} y^*$$

$$= \frac{c_f x}{c_f y} (x^*'x^*)^{-1} x^* y^*$$

(5.4)

and

$$b_2 = c_f x x^* \frac{1}{c_f y^*} y^*$$

$$= \frac{c_f x}{c_f y} x^* y^*.$$  

(5.5)

Methods were developed in Chapters 2 and 3 for finding an error-free reflexive generalized inverse of a positive semidefinite symmetric integral matrix and an exact Moore-Penrose inverse of an $n \times s$ integral matrix. Thus, provided the data are rational, the solutions to the GLM given in Equation 5.2 can be computed using these exact generalized inverses.

To simplify the discussion we are going to assume from now on $y$ and $X$ of the GLM, Equation 5.1, are integral. If they are rational, it is just a matter of scaling $y$ and $X$ as in Equation 5.3 and then multiplying the solution to the integral GLM by the quotient of these scale factors as in Equations 5.4 and 5.5.
Even after computing an error-free generalized inverse of $X'X$ or an exact Moore-Penrose inverse of $X$, error could still be introduced into the solution for $\beta$ of this integral GLM when the final products of Equation 5.2 are formed. The solution to this problem is to use multiple modulus residue arithmetic to complete the computation of the least squares estimator for $\beta$ (i.e., multiply the exact generalized inverse by $X'y$ or multiply the exact Moore-Penrose inverse by $y$.)

Section 5.1 discusses finding the solution $b_1$ in Equation 5.2 exactly, and Section 5.2 discusses finding the solution $b_2$ in Equation 5.2 exactly.

5.1 The Multiple Modulus Residue Arithmetic Least Squares Solution $b_1 = (X'X)^{-1}X'y$

This section deals with exact computation of a least squares solution for $\beta$,

$$b_1 = (X'X)^{-1}X'y,$$

of the integral GLM, Equation 5.1, assuming $y$ and $X$ are integral. This will be done using multiple modulus residue arithmetic.

In Chapter 2, a multiple modulus residue arithmetic method for computing $(X'X)^{-1}$ exactly was given. In this method

$$(X'X)^{-1} = \frac{1}{c_f} B \quad (5.1.1)$$

where $c_f$ is an integer and $B = (b_{ij})$ is integral. The multiple modulus base $\{p_1, p_2, \ldots, p_h\}$ was chosen such that the product
modulus \( M = \prod_{i=1}^{\text{h}} p_i \) satisfied
\[
M > 2 \max\{|cf|, \max_{i,j}|b_{ij}|\}.
\] \hspace{1cm} (5.1.2)

With a multiple modulus base chosen in this way, \( cf \) and \( B \) have unique multiple modulus residue representations with \( cf = /cf/_{M} \) and \( B = /B/_{M} \).

Expressing \( (X'X)^{-1} \) as in Equation 5.1.1, the least squares solution for \( \beta \) of the integral GLM we are considering is
\[
b_{1} = \frac{1}{cf} BX'y.
\] \hspace{1cm} (5.1.3)

The product of Equation 5.1.3 can be formed exactly using multiple modulus residue arithmetic with a multiple modulus base \( \{p_1, p_2, \ldots, p_s\} \), providing the product modulus \( M^* = \prod_{i=1}^{s} p_i \) satisfies
\[
M^* > 2 \max\{|cf|, \max_{i,j}|b_{ij}|, (\text{maximum absolute entry of } BX'y)\}.
\] \hspace{1cm} (5.1.4)

Let \( \tilde{w} = X'y \), if \( M \) satisfies Equation 5.1.2, then
\[
M^* \geq M \sum_{i=1}^{n} |w_i|
\]
satisfies Equation 5.1.4. Therefore, given the multiple modulus residue representations of \( BX'y \sim \{|B|_{p_1} |X|'_{p_1} |y|_{p_1}, |B|_{p_2} |X|'_{p_2} |y|_{p_2}, \ldots\} \),
\[
|B|_{p_s} |X|'_{p_s} |y|_{p_s} \}
\]
and

\[ cf \sim \{ |cf|_{p_1}, |cf|_{p_2}, \ldots, |cf|_{p_s} \}, \]

then \[ |BX'y|_M^* \text{ and } |cf|_M^* \] can be formed and the quantities

\[ cf = |cf|_M^* \text{ and } BX'y = /BX'y/_M^* \]

extracted. A method which does this is outlined below.

**Algorithm 5.1.1** Given \( y \) an \( nx1 \) vector of integers and \( X \) an \( nxs \) integral matrix of fixed known numbers, this method computes \( b_1 = (X'X)^{-1}X'y \) exactly using multiple modulus residue arithmetic.

1. Compute the product modulus bound \( M^* \). First compute

\[ M > 2|\text{det(any minor of } X'X)|, \]

by one of the inequalities given in Section 3.3. Then form \( w = X'y \) and compute

\[ M^* \geq M \sum_{i=1}^{n} |w_i|. \]

2. Choose a multiple modulus base, \( \{p_1, p_2, \ldots, p_s\}, p_i \text{-prime}, \)

such that

\[ \prod_{i=1}^{s} p_i \geq M^*. \]

3. Perform Steps 2 and 3 of the multiple modulus residue arithmetic bordering process, Algorithm 3.4.2, for the multiple modulus base given in Step 2 of this algorithm. The outcome for this will be
\[
\begin{align*}
k_i &= \text{rank}(|X'X|_{p_i}) \text{ for } i = 1, 2, \ldots, s, \\
k &= \text{rank}(X'X), \\
\left| A_k^{\text{adj}} \right|_{p_i} &= \text{the adjoint modulo } p_i \text{ of a } k \times k \text{ nonsingular minor of } X'X, i = 1, 2, \ldots, s, \\
\left| \text{det}(A_k) \right|_{p_i} &= \text{the determinant modulo } p_i \text{ of the } k \times k \text{ nonsingular minor of } X'X, i = 1, 2, \ldots, s, \\
P &= \text{a permutation matrix such that} \\
(X'X)^{-1} &= \frac{1}{\text{det}(A_k)} p' \left[ \begin{array}{cc} A_k^{\text{adj}} & \phi \\ \phi & \phi \end{array} \right] p.
\end{align*}
\]

In terms of the quantities modulo } p_i \text{ in Equation 5.1.1,

\[
\left| cf \right|_{p_i} = \left| \text{det}(A_k) \right|_{p_i}
\]

and

\[
\left| B \right|_{p_i} = p' \left[ \begin{array}{cc} A_k^{\text{adj}} & \phi \\ \phi & \phi \end{array} \right] p.
\]

4. For all } p_i, i = 1, 2, \ldots, s, \text{ form

\[
\begin{cases}
\left| B \right|_{p_i} \left| X' \right|_{p_i} \left| y \right|_{p_i} = p' \left[ \begin{array}{cc} A_k^{\text{adj}} & \phi \\ \phi & \phi \end{array} \right] p \left| X' \right|_{p_i} \left| y \right|_{p_i} & , \\
\left| X' y \right|_{p_i} = & \text{if } k_i \geq k-1 \\
\phi & , \text{ if } k_i < k-1
\end{cases}
\]

\[
\text{if } k_i \geq k-1
\]

\[
\phi & , \text{ if } k_i < k-1
\]
5. Combine the multiple modulus residue representations of

\[ cf \sim \{ |\text{det}(A_k)|_{p_1}, |\text{det}(A_k)|_{p_2}, \ldots, |\text{det}(A_k)|_{p_s} \} \]

and

\[ BX'y \sim \{ |BX'y|_{p_1}, |BX'y|_{p_2}, \ldots, |BX'y|_{p_s} \} \]

to form \( |cf|_M \) and \( |BX'y|_M \), where \( M = \prod_{i=1}^{s} p_i \). Then extract

\[ cf = /cf|_M \text{ and } BX'y = /BX'y|_M \]

6. Form

\[ b_i = \frac{1}{cf} BX'y \]

In the multiple modulus residue arithmetic generalized matrix inversion methods given in Chapter 3, after a rank x rank nonsingular minor was found the product modulus was refined and a subset of the original multiple modulus base was used to complete the problem. A similar thing can be done here provided the refined product modulus bound incorporates the vector \( \omega = X'y \). Therefore, if the multiple modulus residue arithmetic least squares process is started with an appropriate product modulus \( M^* \) such that

\[ M^* > M \sum_{i=1}^{n} |w_i|, \]
where \( M > 2 \det(\text{any minor of } (X'X)) \), then after the rank \((X'X) \times \text{rank}(X'X)\) minor, say \( A_k \), of the matrix \( X'X \) has been found, a new product modulus \( M^{**} \) can be computed such that

\[
M^{**} > 2 \max \{|\det(A_k)|, (\max|A_{i,j}^{(k)}|) \sum_{\ell=1}^{n} |w_\ell| \}.
\]

The new product modulus \( M^{**} \) is less than or equal to \( M^* \), hence a subset (not necessarily a proper subset) of the original multiple modulus base can be used to complete the process.

This multiple modulus residue arithmetic least squares solution to the integral GLM can also be used when \( X'X \) is nonsingular. In fact, the multiple modulus residue arithmetic bordering process is quite efficient when it comes to inverting a positive definite symmetric matrix.

This concludes our first multiple modulus residue arithmetic least squares solution for \( \tilde{b}, \tilde{b}_1 = (X'X)^{-1}X'y \), of the integral GLM.

5.2 The Multiple Modulus Residue Arithmetic Least Squares Solution \( \tilde{b}_2 = X'^+y \)

This section deals with exactly computing the minimum Euclidean length least squares solution of \( \beta \),

\[
\tilde{b}_2 = X'^+y,
\]

for the integral GLM. This will be done using multiple modulus residue arithmetic.

First, consider using the multiple modulus residue arithmetic Moore-Penrose inversion method given in Section 4.2 to find the error-free Moore-Penrose inverse of the matrix \( X \) in the integral GLM. The
method given in Section 4.2 is for an nxs integral matrix where \( n \leq s \).

For the GLM to be meaningful, it must be assumed the number of realizations, \( n \), is greater than the number of parameters, \( s \), that are being estimated. Hence, \( X \) is an nxs integral matrix with \( n \geq s \). The exact Moore-Penrose inverse process of Section 4.2 will compute \((X')^+\) as

\[
(X')^+ = \frac{1}{-a_k} XB_{k-1}.
\]

(The quantities \(-a_k\) and \(B_{k-1}\) are discussed at length in Section 4.2.) Let \(B_{k-1} = B = (b_{ij})\) and \(-a_k = -a\), then the Moore-Penrose inverse of \(X\) is

\[
X^+ = \frac{1}{-a} B'X'
\]

and the shortest vector solution to the least squares minimization problem can be expressed as

\[
b_2 = X_y^+ = \frac{1}{-a} B'X'y.
\]

Given the base, \(\{p_1, p_2, \ldots, p_k\}\), the integer \(-a\) and the matrix \(B\) used to form the Moore-Penrose inverse of \(X\) can be computed exactly using multiple modulus residue arithmetic provided the product modulus, \(M = \prod p_i\), satisfies,

\[
M > 2 \max\{|-a|, \max_{i,j}|b_{ij}|\}.
\]

The result of Equation 5.2.2 can be formed exactly using multiple modulus residue arithmetic with a multiple modulus base,
\{(p_1, p_2, \ldots, p_s), \text{ provided the product modulus } M^* = \prod_{i=1}^{s} p_i, \text{ satisfies}

M^* > 2 \max\{|-a|, \max_{i,j} |b_{ij}|, \text{ (maximum absolute entry of } B'X'y)\}. \tag{5.2.4}

Let \( \omega = X'y \), if \( M \) satisfies Equation 5.2.3, then

\[ M^* \geq \sum_{i=1}^{n} |w_i|, \]

satisfies Equation 5.2.3. Therefore, given the multiple modulus residue representations of

\[ B'X'y \sim \{|B|'_{p_1} |X|'_{p_1} |y|_{p_1} |p_1 \}, \ldots, |B|'_{p_s} |X|'_{p_s} |y|_{p_s} \] \]

and

\[ -a \sim \{|-a|_{p_1}, \ldots, |-a|_{p_s} \}, \]

then \( |B'X'y|_{M^*} \) and \( |-a|_{M^*} \) can be formed and the quantities

\[ B'X'y = /B'X'y//_{M^*} \text{ and } -a = /-a//_{M} \text{ extracted. A method which does } \]

this is outlined below.

Algorithm 5.2.1 Given \( y \) an \( n \times 1 \) vector of integers and \( X \) an \( n \times s \) integral matrix of fixed known numbers, this method computes

\( b_2 = X'y \) exactly using multiple modulus residue arithmetic.
1. Compute the product modulus $M^*$. First, compute

$$M > 2 \max\{|-a|, \max_{i,j} |b_{ij}|\}$$

as done in Step 1 of Algorithm 4.2.3, replacing $A$ in that algorithm with $X'$, (i.e., form $\text{tr}(X'X)$ and $|X'X|$.) Then form $\gamma = X'y$ and compute

$$M^* > M \sum_{i=1}^{n} |w_i|.$$

2. Choose a multiple modulus base $\{p_1, p_2, \ldots, p_s\}$, $p_1$-prime, such that

$$\prod_{i=1}^{s} p_i > M^*.$$

3. Perform Steps 3-5 of Algorithm 4.2.3 on $X'$ for the multiple modulus base in Step 2 of this algorithm. The outcome of this in terms of the quantities modulo $p_i$ in Equation 5.2.1 is

- $k_{p_i} = \text{number of steps completed for the prime } p_i,$
  
i = 1, 2, \ldots, s,

- $k = \text{rank}(X') = \text{rank}(X) = \max_{1 \leq i \leq s} \{k_{p_i}\}$

$$|B_{k-1}|_{p_i} = |B|_{p_i}, \text{ for } i = 1, 2, \ldots, s,$$

and

$$|-a_k|_{p_i} = |-a|_{p_i}, \text{ for } i = 1, 2, \ldots, s.$$
4. For all $p_i$, $i = 1, 2, ..., s$ form

$$|B'X'y|_{p_i} = \begin{cases} |B'|_{p_i} |X'|_{p_i} |y|_{p_i} & \text{if } k_{p_i} > k-1 \\ \phi & \text{if } k_{p_i} < k-1 \end{cases}$$

5. Combine the multiple modulus residue representations

$$|B'X'y|_{p_i} \text{ and } |-a|_{p_i}$$

for all $i = 1, 2, ..., s$, to form $|B'X'y|_{M^*}$ and $|-a|_{M^*}$, $M^* = \prod_{i=1}^{s} p_i$. Then extract

$$B'X'y = /B'X'y/_{M^*} \text{ and } -a = /-a/_{M^*}.$$

6. Form

$$\beta_2 = \frac{1}{-a} B'X'y.$$

This concludes the second multiple modulus residue arithmetic least squares solution for $\beta$, $\beta_2 = X^+_2 y$, of the integral GLM. Algorithm 5.2.1 has been implemented in standard FORTRAN. The FORTRAN program is included in the Appendix and the implementation is discussed in Chapter 7.

As was indicated in this chapter, once the multiple modulus residue arithmetic generalized matrix inversion methods were established, it was a fairly simple problem to find least squares solution of $\beta$ for the integral GLM using multiple modulus residue arithmetic. The problem boiled down to choosing an appropriate product modulus bound. These same ideas will carry over when trying to compute other statistical quantities exactly using multiple modulus residue arithmetic.
6. RATIONAL MATRICES

Throughout this research it has been assumed the nxs matrix A on which multiple modulus residue arithmetic methods are applied is integral. Obviously, the matrices for which generalized inverses are required will not always be integral. We can assume, however, the matrices will have at worst rational entries since matrix inversion methods are usually done with the aid of the computer. In the previous discussions, when confronted with the possibility of a matrix having rational entries, we stated that the procedure to follow was to scale the matrix to integers and proceed with the given integral process. From a computational point of view, this is not a small task. This chapter deals with the problem of how to handle rational input. Section 6.1 will discuss the standard solution of scaling A to an integral matrix. Section 6.2 outlines how this problem is handled on the computer.

6.1 Scaling a Rational Matrix to be Used for Multiple Modulus Residue Arithmetic

For this discussion, let \( A = \begin{pmatrix} a_{ij} \\ b_{ij} \end{pmatrix} \) with \( a_{ij}, b_{ij} \in \mathbb{Z} \), be an nxs matrix with rational entries. Let \( cf_A \) be an integer such that

\[
 cf_A \cdot A = A^* \tag{6.1.1}
\]

where \( A^* \) is integral. Some possible values for \( cf_A \) are
\[ cf_{A} = \text{lcm} \left( b_{ij} \right) \]

the least common multiple of the \( b_{ij} \)'s, or

\[ cf_{A} = \prod_{i,j} b_{ij} \]

For simplicity of discussion in this section, we will use the later value for \( cf_{A} \) yielding

\[ A^* = \left[ \frac{a_{ij} \left( \prod_{\ell,m} b_{\ell m} \right)}{b_{ij}} \right] = \left[ a_{ij} \cdot \left( \prod_{\ell,m} b_{\ell m} \right) \right] . \]

Chances are the entries in \( A^* \) will be quite large. Thinking in terms of computer implementation in fixed-point arithmetic, these numbers may exceed the permissible fixed-point number range. However, to use one of the multiple modulus residue arithmetic methods described in the previous chapters, \( A^* \) does not need to be formed. The quantity that is needed is \( |A^*|_{p} \), \( p \)-prime, for each prime modulus in the multiple modulus base. The residue of \( A^* \) modulo \( p \) can be formed without actually computing \( A^* \) by

\[ |A^*|_{p} = \left[ \frac{|a_{ij}|_{p} \cdot \prod_{\ell,m} |b_{\ell m}|_{p} \cdot \prod_{\ell \neq i, m \neq j}}{p} \right] . \]

For all the multiple modulus residue arithmetic generalized matrix inversion techniques given in Chapters 3 and 4, the generalized inverse of an \( n \times s \) integral matrix \( A^* \) was expressed as
\[ A^* = \frac{1}{cf} A^{**} \]  \hspace{1cm} (6.1.2)

where \( cf \) is an integer and \( A^{**} \) is integral. If the rational matrix \( A \) is expressed as

\[ A = \frac{1}{cf_A} A^* \]

\( cf_A \in \mathbb{Z} \) and \( A^* \) integral, then

\[ A^- = cf_A A^* = \frac{cf_A}{cf} A^{**} \]  \hspace{1cm} (6.1.3)

Thus, given a unique multiple modulus residue representation of \( cf \) and \( cf_A \cdot A^{**} \) for an appropriate base, an exact generalized inverse in the form of Equation 6.1.3 can be found. An appropriate base, \( \{p_1, p_2, \ldots, p_s\} \), is such that the product modulus \( M = \prod_{i=1}^{s} p_i \),

\[ -\frac{M}{2} < \max\{|cf|, |cf| \cdot \max_{i,j} |a_{ij}^{**}|\} < \frac{M}{2} \]  \hspace{1cm} (6.1.4)

where \( A^{**} = (a_{ij}^{**}) \). If the product modulus satisfies Equation 6.1.4, then the error-free generalized inverse is

\[ A^- = \frac{1}{|cf_A \cdot A^{**}|_M} \cdot |cf_A \cdot A^{**}|_M \cdot \frac{1}{M} \]

(Note, \( |cf_A \cdot A^{**}|_p = \left[ \prod_{l,m} |b_{lm}|_p \prod_{i,j} |a_{ij}^{**}|_p \right] \).

It has already been determined that the formation of \( A^* \) as in Equation 6.1.1, was not necessary for the construction of \( |A^*|_p \). Consequently, it would be desirable to be able to find a product
modulus bound which satisfies Equation 6.1.4 without forming \( A^* \). In terms of computer implementation, the matrix \( A^* \) could be formed in floating-point arithmetic and then a product modulus bound based on \( A^* \) could be computed. Although the outcome would be the same as computing the bound based on \( A^* \), we propose a technique which constructs the product modulus bound based on the integer \( c_{f_A} \) and the nxs rational matrix \( A \).

Three major bounds discussed in this research are

\[
|\text{det}(\text{any minor of } A)| \leq \prod_{i=1}^{n} \left( \sum_{j=1}^{s} a_{ij}^2 \right)^{\frac{1}{2}},
\]

\[
|\text{det}(\text{any minor of } A)| \leq \prod_{i=1}^{n} \max(R_i^+, R_i^-)
\]

(6.1.5)

with \( R_i^+ \) and \( R_i^- \) defined as in Corollary 3.3.2.1, and

\[
\max\{|-a_k|, \max_{i,j} |b_{ij}|\} \leq \max\{m^n, n^{2n-1}\}
\]

with \( m = \min\{|\text{tr}(AA')|, ||AA'||\} \) where \( -a_k \) and \( B_{k-1} = (b_{ij}) \) are defined as in Theorem 4.2.1. It should be noted that none of these bounds was conditional on the matrix being integral. However, to find \( A^* \) as in Equation 6.1.2 using multiple modulus residue arithmetic, a product modulus, \( M \), must be found such that

\[
-\frac{M}{2} < \max\{|c_{f_A}|, \max_{i,j} |a_{ij}^*|\} < \frac{M}{2}
\]

(6.1.6)

where \( A^* = (a_{ij}^*) \). Suppose the Inequalities 6.1.5 were formed
based on the rational nxs matrix $A$, then a multiple modulus base, 
\{p_1, p_2, ..., p_t\}, with a product modulus $M = \prod_{i=1}^{t} p_i$ such that

$$M > 2|\text{cf}_A|^n \prod_{i=1}^{n} \left[\sum_{j=1}^{s} a_{ij}^2\right]^{1/2},$$

(6.1.7)

$$M > 2|\text{cf}_A|^n \prod_{i=1}^{n} \max(R_i^+, R_i^-),$$
or

$$M > 2(\text{cf}_A)^{2n} \{\max m^n, n^{2n-1}\}$$

where $m = \min\{\text{tr}(AA'), ||AA'||\}$, would satisfy Equation 6.1.6
(depending on which multiple modulus residue arithmetic matrix
inversion technique is used.) This takes care of finding a product
modulus bound based on the integer $\text{cf}_A$ and the nxs rational matrix $A$
which can be used to find an appropriate multiple modulus base for
computing $A^*$ of Equation 6.1.2 exactly. But, to compute $A^-$ as
in Equation 6.1.3 exactly, a product modulus which satisfies Equation
6.1.4 is necessary. A simple adjustment to the Inequalities 6.1.7
solves this problem. Given a multiple modulus base, \{p_1, p_2, ..., p_t\},
p_i-prime, the product modulus $M = \prod_{i=1}^{t} p_i$ will satisfy Equation 6.1.4
provided

$$M > 2|\text{cf}_A|^{n+1} \prod_{i=1}^{n} \left[\sum_{j=1}^{s} a_{ij}^2\right]^{1/2},$$

$$M > 2|\text{cf}_A|^{n+1} \prod_{i=1}^{n} \max(R_i^+, R_i^-),$$
or

$$M > 2|\text{cf}_A|^{2n+1} \max\{m^n, n^{2n}\}$$
where \( m = \min\{\text{tr}(AA'), ||AA'||\} \). Therefore, forming \( |cf|_{p_i} \) and
\[
|cf_{A|p_i} A^{**}|_{p_i}
\]
for each \( p_i \) in the multiple modulus base results in unique multiple modulus residue representations of \( cf \) and
\( cf_A \cdot A^{**} \), the component parts of \( A^- \) given in Equation 6.1.3.

It should be pointed out that for computer implementation the integer quantity \( cf_A \) does not need to be formed in fixed-point arithmetic. It is the quantity \( |cf_A|_p = \prod_{i,j} |b_{ij}|_p \), \( p \)-prime, that needs to be representable in fixed-point arithmetic. Provided arithmetic modulo \( p \) can be performed in the computer fixed-point arithmetic mode, \( |cf_A|_p \) can be constructed. It is true that \( cf_A \) is needed in the computation of the product modulus bound, but these bounds are computed in floating-point arithmetic.

6.2 Scaling Rational Matrices on the Computer

This section addresses the problem of given a rational matrix, \( A \), entered on the computer in floating-point arithmetic mode, how \( cf_A \) is found and \( |A^*|_p \), \( p \)-prime, formed on the computer. These quantities were defined in Section 6.1 by

\[
 cf_A \cdot A = A^* \tag{6.2.1}
\]

with \( cf_A \in \mathbb{Z} \) and \( A^* \) integral.

It is first necessary to define the floating-point numbers. Kennedy and Gentle (1980) give a detailed discussion of the floating-
point number system. The general form of a normalized floating-point number is

\[ \pm . f_1 f_2 \ldots f_t \times \beta^e \]

where \( \beta \) is the base and the fixed radix digits \( f_1, f_2, \ldots, f_t \) are integers satisfying

\[ 1 \leq f_1 \leq \beta - 1, \]

\[ 0 \leq f_i \leq \beta - 1 \quad \text{for } i = 2, 3, \ldots, t, \]

and the exponent \( e \) is an integer such that

\[ -m \leq e \leq M. \]

For example, the IBM 370 uses \( \beta = 16, m = -64, M = 63 \), and \( t = 6 \) for single precision floating-point numbers. The computer stores floating-point numbers in \( t+2 \) separate pieces. The sign, + or -, the exponent, \( e \), and the fixed radix digits, \( f_i \) for \( i = 1, 2, \ldots, t \), are stored separately so that any of the pieces can be accessed independently of the others.

The format of the floating-point numbers and the way in which they are stored on the computer make finding \( \mathbf{cF} \) and forming \( |\mathbf{A}^*|_p \), as discussed in Section 6.1, quite easy. Assume \( \mathbf{A} \) is a rational matrix, then for \( a_{ij}, b_{ij} \in \mathbb{Z} \),

\[
\mathbf{A} = \begin{bmatrix} a_{ij} \\ b_{ij} \end{bmatrix} = (\pm . f_1 f_2 \ldots f_{t_{ij}} \times \beta^{e_{ij}}) \quad (6.2.2)
\]

\[ = (\pm f_1 f_2 \ldots f_{t_{ij}} \times \beta^{e_{ij} - t_{ij}}) \]
where \( f_{t_{ij}} \), \( t_{ij} \leq t \), is the last nonzero radix point for the \( i,j \)th entry of \( A \). If \( e_{ij} - t_{ij} \geq 0 \), then \( b_{ij} = 1 \) and

\[
a_{ij} = \pm f_1 f_2 \ldots f_{t_{ij}} \beta^{-e_{ij} - t_{ij}}.
\]

If \( e_{ij} - t_{ij} < 0 \), then \( b_{ij} = \frac{-(e_{ij} - t_{ij})}{1} \) and \( a_{ij} = \pm f_1 f_2 \ldots f_{t_{ij}} \). Consider the following example.

**Example 6.2.1** Suppose \( \beta = 10 \) and \( t = 4 \), then for

\[
A = \begin{bmatrix}
0.1234 \times 10^5 & -0.5670 \times 10^3 \\
-0.8900 \times 10^1 & 0.3102 \times 10^2
\end{bmatrix}
\]

we have the following representation.

\[
A = \begin{bmatrix}
a_{ij} \\
b_{ij}
\end{bmatrix}
= \begin{bmatrix}
1234 \times 10^1 & -567 \times 10^0 \\
-89 \times 10^1 & 3102 \times 10^2
\end{bmatrix},
\]

where \( e_{11} - t_{11} = 5-4 = 1 \), \( e_{12} - t_{12} = 3-3 = 0 \), \( e_{21} - t_{21} = 1-2 = -1 \),

and \( e_{22} - t_{22} = 2-4 = -2 \).

Given a rational matrix \( A \) as expressed in Equation 6.2.2, we want to find some integer \( cf_A \) to satisfy Equation 6.2.1. The best choice for \( cf_A \) is \( cf_A = \ell \text{cm}(b_{ij}) \). Let
\[
e = \begin{cases} 
\min(e_{ij} - t_{ij}), & \text{if } \min(e_{ij} - t_{ij}) < 0 \\
0, & \text{if } \min(e_{ij} - t_{ij}) \geq 0
\end{cases}
\quad (6.2.3)
\]

then

\[\text{cf}_{A} = \frac{\text{lcm}(b_{ij})}{i,j} = \beta^{-e}.\]

In example 6.2.1, \(e = -2\), \(\text{cf}_{A} = 10^{2}\) and

\[
\text{cf}_{A} \cdot A = \begin{bmatrix}
1234 \times 10^{3} & 567 \times 10^{2} \\
89 \times 10^{1} & 3102
\end{bmatrix},
\]

which is an integral matrix. In the case where \(\min(e_{ij} - t_{ij}) \geq 0\), \(A\) is integral to start with and no scaling is necessary.

Now that we can find \(\text{cf}_{A}\), the next problem to solve is given a prime modulus \(p\), how to compute \(\lvert A^{*}\rvert_{p}\) for the matrix \(A^{*} = (a_{ij}^{*})\) of Equation 6.2.1. This can be done in fixed point arithmetic on the computer provided the prime \(p\) is such that arithmetic modulo \(p\) can be performed in the computer fixed-point arithmetic mode and provided \((\beta, p) = 1\) for the floating-point base \(\beta\). Recall from the general form of floating-point numbers the radix points \(f_{1}, f_{2}, \ldots, f_{t_{ij}}\) are integers. Now,

\[
a_{ij}^{*} = \text{cf}_{A} \frac{a_{ij}}{b_{ij}} = \beta^{-e}(\pm f_{1}f_{2} \ldots f_{t_{ij}} \times \beta^{e_{ij} - t_{ij}})
\]

\[
= \pm 1 \beta^{e_{ij} - t_{ij} - e}(f_{t_{ij}} + f_{t_{ij} - 1} \beta + f_{t_{ij} - 2} \beta^{2} + \ldots + f_{1} \beta^{t_{ij} - 1}).
\]
Let $B = |B|_p$, then

\[ |a_{ij}^*|_p = \left| \pm 1|_{p} |\beta |_{p}^{e_{ij}-t_{ij}-e} \right| |f_{t_{ij}}|_{p} + \left| f_{t_{ij}-1}|_{p}| \beta |_{p} + \left| f_{t_{ij}-2}|_{p}| \beta |_{p} + \right| + \cdots \]

\[ + \left| f_{1}|_{p}| \beta |_{p} \right| . \tag{6.2.4} \]

From this, it can be seen that with the values of $e$, as defined in Equation 6.2.3, the quantities $|a_{ij}^*|_p$ can be formed from the floating-point representation of $a_{ij}$ without constructing $a_{ij}^*$ first. From a computational point of view, $|a_{ij}^*|_p$, p-prime and $(\beta, p) = 1$, can be computed more efficiently using Horner's rule as opposed to using the expression for $|a_{ij}^*|_p$ given in Equation 6.2.4. Horner's rule yields

\[ |a_{ij}^*|_p = \left| \pm 1|_{p} \left| \cdots \left| f_{1}|_{p} |\beta |_{p} \right| + f_2|_{p} \beta + f_3|_{p} \beta \cdots + f_{t_{ij}}|_{p} \beta \right| + \right| + \cdots \]

where $\beta_p = |\beta|_p$. The next example computes $|A^*|_p$ using Horner's rule for the matrix $A$ of Example 6.2.1.

**Example 6.2.2** $\beta = 10$, $t = 4$

\[
A = \begin{bmatrix}
.1234 \times 10^5 & -.5670 \times 10^3 \\
-.8900 \times 10^1 & .3102 \times 10^2
\end{bmatrix}
\]

Let $p = 7$, $(\beta, p) = (10, 7) = 1$, $e = \min_{i,j}(e_{ij}-t_{ij}) = -2$, so $c_{f_A} = 10^2$. The matrix $|A^*|_7$ is formed as follows.
$$\beta_7 = |\beta|_7 = |10|_7 = 3 \quad \text{and}$$

$$|a^*_{11}|_7 = \left| \begin{array}{c}
|11|_7 \\
|1\cdot 3 + 2|_7 \\
|5|_7
\end{array} \right| \\
= 1 \cdot \left| \begin{array}{c}
|5\cdot 3 + 3|_7 \\
|3 + 4|_7 \\
|3^3|_7
\end{array} \right|_7 \\
= |4|_7$$

$$= \left| \begin{array}{c}
|5\cdot 3 + 4|_7 \\
|3^3|_7
\end{array} \right|_7 \\
= |16|_7$$

$$= 2 \cdot |3^3|_7_7$$

$$= |2 \cdot 6|_7$$

$$= 5.$$ 

$$|a^*_{12}|_7 = \left| \begin{array}{c}
|-1|_7 \\
|5\cdot 3 + 6|_7 \\
|7|_7
\end{array} \right| \\
= 6 \cdot \left| \begin{array}{c}
|0\cdot 3 + 7|_7 \\
|3^2|_7
\end{array} \right|_7 \\
= 6 \cdot 0 \cdot |3^2|_7$$

$$= 0.$$
\[ |a_{21}^*|_7 = \begin{vmatrix} -1 \end{vmatrix}_7 \begin{vmatrix} \begin{vmatrix} 8 + 3 \end{vmatrix}_7 + 9 \end{vmatrix}_7 \begin{vmatrix} 3 \end{vmatrix}_7 \\ \begin{vmatrix} 12 \end{vmatrix}_7 \end{vmatrix}_7 \\
= \begin{vmatrix} 6 \end{vmatrix}_7 \begin{vmatrix} 5 \end{vmatrix}_7 \begin{vmatrix} 3 \end{vmatrix}_7 \\ \begin{vmatrix} 15 \end{vmatrix}_7 \end{vmatrix}_7 \\
= \begin{vmatrix} 6 \end{vmatrix}_7 \\
= 6 \]

\[ |a_{22}^*|_7 = \begin{vmatrix} +1 \end{vmatrix}_7 \begin{vmatrix} \begin{vmatrix} 3 \end{vmatrix}_7 + 1 \end{vmatrix}_7 \begin{vmatrix} 3 \end{vmatrix}_7 + 2 \end{vmatrix}_7 \begin{vmatrix} 3 \end{vmatrix}_7 \\ \begin{vmatrix} 3 \end{vmatrix}_7 \end{vmatrix}_7 \\
= \begin{vmatrix} 1 \end{vmatrix}_7 \begin{vmatrix} \begin{vmatrix} 3 \end{vmatrix}_7 + 0 \end{vmatrix}_7 \begin{vmatrix} 3 \end{vmatrix}_7 + 2 \end{vmatrix}_7 \begin{vmatrix} 3 \end{vmatrix}_7 \\ \begin{vmatrix} 2 \end{vmatrix}_7 \end{vmatrix}_7 \\
= \begin{vmatrix} \begin{vmatrix} 2 \end{vmatrix}_7 + 2 \end{vmatrix}_7 \begin{vmatrix} 3 \end{vmatrix}_7 \\ \begin{vmatrix} 8 \end{vmatrix}_7 \end{vmatrix}_7 \\
= \begin{vmatrix} 1 \end{vmatrix}_7 \begin{vmatrix} 3 \end{vmatrix}_7 \\
= 1 \]

therefore
\[ |c_fA^*A|_7 = |A^*|_7 = \begin{bmatrix} 5 & 0 \\ 6 & 1 \end{bmatrix} \]
This chapter has illustrated that it is not unreasonable to assume, without loss of generality, the matrices we are working with are integral. However, this chapter also demonstrates that the solution of scaling the rational matrices is not trivial.
7. COMPUTER IMPLEMENTATION

Among the first people to implement on the computer, exact procedures for solving a full-rank system of linear equations were Borosh and Fraenkel (1966), Newman (1967), Fraenkel and Loewenthal (1971), Howell (1970), Howell and Gregory (1970), and Cabay and Lam (1977a,b). McClellan (1977a,b) gives a comparison of some of the different algorithms these people used to find the exact solution to linear equations. This chapter discusses the computer implementation of the multiple modulus residue arithmetic methods for finding exact generalized inverses and exact solutions to linear least squares problems presented in this research.

The basic multiple modulus residue arithmetic procedure can be broken into four pieces; computation of the product modulus bound, the choice of the prime numbers to be included in the multiple modulus base, the multiple modulus residue arithmetic solution, and the combining of the multiple modulus residue arithmetic solution to form the exact solution over the field of rational numbers. Section 7.1 will discuss the first two parts of the procedure and Section 7.2 will discuss the last two parts. Section 7.3 will give some sample output from the four implemented FORTRAN programs included in the Appendix. The four programs will be referred to as the Gaussian Elimination method (Algorithm 3.4.1), the Bordering method (Algorithm 3.4.2), the Moore-Penrose Inversion method (Algorithm 4.2.3), and the Minimum Euclidean Norm Least Squares
solution (Algorithm 5.2.1). Throughout this chapter we will once again only consider integral input. Even though the FORTRAN programs in the Appendix were implemented on an IBM 370, the discussion of computer implementation given in this chapter is very general.

7.1 Product Modulus Bound and the Multiple Modulus Base

The computation of the product modulus bound is done in double precision floating-point arithmetic. These bounds cannot be computed exactly using fixed-point arithmetic because they will exceed the permissible fixed-point number range. Consequently, there will be some accumulation of rounding error in their construction. All of the product modulus bounds presented in this research are formed by taking products and sums of positive quantities. The type of rounding errors that occur under those circumstances is due to truncation. One wonders if it is possible for enough rounding error to accumulate in the construction of the bounds such that the multiple modulus residue representations of the quantities in the exact solution will not be unique for the resulting multiple modulus base.

Based on the following two facts, we feel confident that the product modulus bounds computed in double precision floating-point arithmetic will be accurate enough to generate an appropriate multiple modulus base. First, the product modulus bounds used in this work are usually extremely conservative. Second, double precision floating-point arithmetic carries many significant digits (on the IBM 370, 16 equivalent significant decimal digits), so it is hard to imagine the
truncation error occurring here would effect the leading significant digits.

There is a more serious problem than loss of significant digits that can occur when trying to compute a product modulus bound using floating-point arithmetic. It is possible the floating-point exponent could overflow. On the IBM 370, for example, the largest number representable in floating-point arithmetic (single or double precision) is on the order of $10^{75}$. Since the problems to be solved will usually be the result of scaling rational data to integers, the size of these integers can be very large. When combinations of these are multiplied together, their magnitude could easily exceed $10^{75}$ causing the floating-point exponent to overflow.

Let $A = (a_{ij})$ be an $n \times s$ integral matrix and $y = X \tilde{y}$ be an integral linear model with $X = (x_{ij})$ an $n \times s$ integral matrix. The multiple modulus base, $\{p_1, p_2, ..., p_t\}$, and the corresponding product modulus, $M = \prod_{i=1}^{t} p_i$, in the FORTRAN programs are chosen such that

$$M > 2 \prod_{i=1}^{n} \max(R_i^+, R_i^-)$$

where $R_i^+ = \sum_{j=1}^{s} \max(a_{ij}, 0)$ and $R_i^- = -\sum_{j=1}^{s} \min(a_{ij}, 0)$ ;

$$M > 2 \max\{m, 2^n n^{-1}\}$$

where $m = \min\{\text{tr}(AA'), ||AA'||\}$ ;

or $$M > 2 \max\{m_x^n, n^2 n^{-1}\} \sum_{j=1}^{s} |w_i|$$

where $m_x = \min\{\text{tr}(X'X), ||X'X||\}$ and $\tilde{y} = X' \tilde{y}$.
The problem of overflowing the floating-point exponent can be handled by computing the logarithms of both sides of these inequalities. This means if a multiple modulus base, \( \{p_1, p_2, \ldots, p_t\} \), satisfies

\[
\sum_{i=1}^{t} \log(p_i) \geq \sum_{i=1}^{n} \log\{\max(R_i^+, R_i^-)\} + \log(2),
\]

or

\[
\sum_{i=1}^{t} \log(p_i) \geq \left\{ \begin{array}{ll}
\frac{n \log(m)}{21og(n)+(n-1)\log(m)}, & m < n^2 \\
\frac{\log(m)}{21og(n)+(n-1)\log(m)} + \log(2), & m \geq n^2
\end{array} \right\} + \log(2),
\]

then the product moduluses, \( M = \prod_{i=1}^{t} p_i \), satisfies the corresponding inequality from Inequalities 7.1.1.

The Gaussian Elimination program does not use logarithms when computing the product modulus bound. This is because this program uses Johnson and Newman's bound,

\[
\det(A_{r \times r}) = \prod_{i=1}^{r} \max(R_i^+, R_i^-) - \prod_{i=1}^{r} \min(R_i^+, R_i^-),
\]

where \( A_{r \times r} \) is a r\times r nonsingular minor of \( A \) with \( r = \text{rank}(A) \), to refine the product modulus bound once a largest nonsingular minor has been found. Taking the logarithm of this bound would not help the potential floating-point exponent overflow problem because the bound would have to be formed before the logarithm could be taken. The
Bordering program uses Schinzel's bound (first inequality of Inequalities 7.1.1) throughout the entire program. Consequently, the first logarithmic inequality of Inequalities 7.1.2 is used in this program. The product modulus bound portions of the Gaussian Elimination program could be replaced by those of the Bordering program giving the Gaussian Elimination program a wider range of problems to solve. The goal of the computer implementation portion of this research was to demonstrate several different methods, some possibly better than others. For both the Moore-Penrose inversion method and the Least Squares program, the logarithms are used in the computation of the product modulus bound and in choosing the multiple modulus base as indicated by the last two inequalities of Inequalities 7.1.2.

In all four of the programs given here, the possible prime numbers to be included in the different multiple modulus bases are input into the program through a FORTRAN DATA statement and stored in an array. An alternative to inputing a table of primes into the program is to use an algorithm which generates prime numbers in a specified number range. Pri chard (1981) or Buhler, Crandall and Penk (1982) gives such algorithms. Using algorithms such as theirs could be advantageous. This is because when a table of primes is input into the program it contains a limited number of primes, say \( p_1, p_2, \ldots, p_{1000} \). It is not realistic to think of inputing into an array every prime number from 3 to the largest prime number, \( p^* \), for which residue arithmetic modulo \( p^* \) can be performed in fixed-point arithmetic. Consequently,
the program is limited to solving only those problems for which a
product modulus of \( M = \prod_{i=1}^{1000} p_i \) is large enough to exceed the computed
product modulus bound. An algorithm which could generate all the
prime numbers in the interval \([3, p^*]\) would not restrict the program
in this way.

This concludes our discussion of the first two parts of the basic
multiple modulus residue arithmetic procedure. The documentation in
the FORTRAN programs indicates more directly, with respect to the
FORTRAN code, how the multiple modulus bound is computed and how the
multiple modulus base is chosen.

7.2 Implementing Multiple Modulus Residue Arithmetic
and the Symmetric Mixed Radix Representation

Once the product modulus has been determined and the multiple
modulus base has been chosen, the multiple modulus residue arithmetic
process can begin. This section will discuss how to construct the
unique multiple modulus residue representations for the component
parts of the exact generalized inverse or the exact least squares
solution we are trying to find. Then the problem of extracting these
exact solutions over the field of rational numbers from their unique
multiple modulus residue representations will be addressed.

Let the multiple modulus base be \( \{p_1, p_2, \ldots, p_t\} \). In each of
the four FORTRAN programs, residue arithmetic modulo \( p_i \), \( i = 1, 2, \ldots, t \),
is performed in the computer fixed-point arithmetic mode utilizing the
built-in FORTRAN MOD function. The function is
MOD(x, p) = x - \left\lfloor \frac{x}{p} \right\rfloor

where x, p \in \mathbb{Z} and \left\lfloor \frac{x}{p} \right\rfloor is the greatest integer not exceeding the magnitude of \frac{x}{p}. The sign of the MOD function will be, by definition, the sign of x. With the aid of the MOD function, the residue of x modulo p_1 is

\[ |x|_{p_1} = \begin{cases} 
    \text{MOD}(x, p_1), & \text{if } x \geq 0 \\
    \text{MOD}(x, p_1) + p_1, & \text{if } x < 0
  \end{cases} \]

The only quantity which cannot be found in FORTRAN by manipulating the built-in FORTRAN MOD function algebraically is the inverse of x modulo p_1, x^{-1}(p_1). This is found by using Euclid's Extended Algorithm. Gregory (1980) discusses and flow charts this algorithm. Each of our programs contain a FORTRAN function subprogram called EUCLID. EUCLID computes x^{-1}(p), p > 1 prime, according to Euclid's Extended Algorithm as follows.

**Function 7.2.1** This function EUCLID (x, p) computes x^{-1}(p).

Set \((u_1, u_2) = (1, x)\) and \((v_1, v_2) = (0, p)\).

1) Set \(q = \frac{u_2}{v_2}\) and \((t_1, t_2) = (u_1, u_2) - q(v_1, v_2),\)
\((u_1, u_2) = (v_1, v_2),\)
and
\((v_1, v_2) = (t_1, t_2).\)
ii) If $v_2 \neq 1$ go to (i). Else EXIT with

$$EUCLID(x,p) = x^{-1}(p) = \begin{cases} v_1 & , \text{if } v_1 > 0 \\ v_1 + p & , \text{if } v_1 < 0 \end{cases}.$$ 

With the use of the built-in FORTRAN MOD function and this function subprogram EUCLID, residue arithmetic modulo $p_i$ can be performed in the computer fixed-point arithmetic mode for each prime modulus, $p_i$, in the multiple modulus base.

This brings us to the last stage of the multiple modulus residue arithmetic problem solving process. We now have a unique multiple modulus residue representation (with respect to the given base, $\{p_1, p_2, \ldots, p_t\}$) for each of the component parts of the exact solution we are seeking. That is, we have unique multiple modulus residue representations for $cf \in \mathbb{Z}$ and $A^* = (a^*_{ij})$ integral where

$$A^- = \frac{1}{cf} A^* ,$$

or for $cf_b \in \mathbb{Z}$ and $\hat{b}^*$ integral where

$$\hat{b} = X^y = \frac{1}{cf_b} \hat{b}^* .$$

The multiple modulus residue representations must be combined to form the symmetric residue of these quantities modulo $M = \prod_{i=1}^{t} p_i$. In this text, the suggested way to do this was to use the Chinese Remainder Theorem. However, this presents a problem when it comes to computer implementation. The problem is that the Chinese Remainder Theorem requires performing arithmetic modulo $M$. For even the smallest problem,
arithmetic modulo M may not be able to be done in the computer fixed-point arithmetic mode because the modulus M will be too large. Recall that the reason for using multiple modulus residue arithmetic rather than single modulus residue arithmetic to find exact solutions was because the size of the modulus necessary to solve the problem exactly would be too large for residue arithmetic modulo M to be performed on the computer using fixed-point arithmetic.

There is an alternative to using the Chinese Remainder Theorem for finding the symmetric residue of a quantity modulo $M = \prod_{i=1}^{t} p_i$ from its multiple modulus residue representation with respect to the base $\{p_1, p_2, ..., p_t\}$. Consider the following theorem. A proof of this theorem can be found in Lindamood (1964).

**Theorem 7.2.1** If $M = \prod_{i=1}^{t} p_i$, then any integer $y$ in the interval $\left[ -\frac{M}{2}, \frac{M}{2} \right]$ has a unique representation of the form

$$y = \alpha_1 + \alpha_2 p_1 + \alpha_3 p_1 p_2 + \cdots + \alpha_t p_1 p_2 \cdots p_{t-1}$$

where $\langle \alpha_1, \alpha_2, ..., \alpha_t \rangle$ are the unique symmetric mixed-radix digits satisfying

$$\frac{-p_i}{2} < \alpha_i < \frac{p_i}{2},$$

for all $i = 1, 2, ..., t$. 
According to Theorem 7.2.1, given a symmetric mixed-radix representation \( \langle a_1, a_2, \ldots, a_t \rangle \) with respect to the base \( \{p_1, p_2, \ldots, p_t\} \), the unique integer in the interval \( \left\{ \frac{-M}{2}, \frac{M}{2} \right\} \),

\[ M = \prod_{i=1}^{t} p_i, \]

that this represents can be determined by forming the product-sum of Theorem 7.2.1. The problem now is how to determine \( \langle a_1, a_2, \ldots, a_t \rangle \) from the unique multiple modulus residue representation, \( \{ |y|_{p_1}, |y|_{p_2}, \ldots, |y|_{p_t} \} \), of an integer \( y \). Howell and Gregory (1970) show this can be done in the following way. From Theorem 7.2.1

\[ /y_{p_1} = a_1 \]

hence, \( p_1 | (y - a_1) \). Let \( y_1 = y \) and

\[ y_2 = \frac{y_1 - a_1}{p_1} = a_2 + a_3p_2 + a_4p_2p_3 + \ldots + a_tp_2p_3 \ldots p_{t-1}, \]

therefore \( y_2 \) is an integer and

\[ /y_{p_2} = a_2. \]

In general for \( i = 2, 3, \ldots, t, \) \( p_{i-1} | (y_{i-1} - a_{i-1}) \) and

\[ y_i = \frac{y_{i-1} - a_{i-1}}{p_{i-1}} = a_i + a_{i+1}p_i + a_{i+2}p_ip_{i+1} + \ldots + a_tp_ip_{i+1} \ldots p_{t-1}, \]

therefore, \( y_i \) is an integer and

\[ /y_{p_i} = a_i. \]
Thus, given the unique symmetric multiple modulus residue representation \(\{y/p_1, y/p_2, \ldots, y/p_t\}\), the unique symmetric mixed-radix representation \(<a_1, a_2, \ldots, a_t>\) can be constructed. Let

\[ y_1 \sim \{y/p_1, y/p_2, \ldots, y/p_t\}, \]

this can be written as

\[ y_1 \sim \{\alpha_1, y/p_2, \ldots, y/p_t\}. \]

By definition

\[ \alpha_1 \sim \{\alpha_1, y/p_2, \ldots, y/p_t\}. \]

Now,

\[ p_1y_2 = y_1 - \alpha_1 \sim \{0, y_1y_1 - y_2, y_1 - y_2/p_2, \ldots, y_1 - y_2/p_t\}. \]

Since the moduli are prime, \(p_1^{-1}(p_i)\) exists for all \(i \neq j\).

So,

\[ y_2/p_1 = (y_1y_1 - y_2/p_1) - \alpha_1/p_1 \]

for \(i = 2, 3, \ldots, t\). Let

\[ y_2 \sim \{y_2/p_2, y_2/p_3, \ldots, y_2/p_t\}, \]

this can be rewritten as

\[ y_2 \sim \{\alpha_2, y_2/p_3, \ldots, y_2/p_t\}. \]

In general for \(j = 2, 3, \ldots, t-1,\)

\[ \alpha_j = y_j/p_j \sim \{\alpha_j, y_j/p_{j+1}, \ldots, y_j/p_t\}, \]
\[ p_j y_{j+1} = y_j - \alpha_j \sim \{0, /y_j - \alpha_j/p_{j+1}, \ldots, /y_j - \alpha_j/p_t \} \]

and for \( i = j+1, \ldots, t \)
\[ /y_{j+1}/p_i = (y_j - \alpha_j) p_{j+1}^{-1}(p_i)/p_i, \]

then
\[ y_{j+1} \sim \{/y_{j+1}/p_{j+1}, /y_{j+1}/p_{j+2}, \ldots, /y_{j+1}/p_t \} \]
\[ \sim \{\alpha_{j+1}, /y_{j+1}/p_{j+2}, \ldots, /y_{j+1}/p_t \} . \]

The next example illustrates this process.

**Example 7.2.1** Let the multiple modulus base be \( \{7, 11, 13\} \), then
\[ M = \prod_{i=1}^{3} p_i = 1001. \]
Given the multiple modulus residue representation \( \{4, 2, 4\} \), find \( \langle \alpha_1, \alpha_2, \alpha_3 \rangle \) and compute the unique integer, \( y \), in the interval \( \left[ \frac{-1001}{2}, \frac{1001}{2} \right] \) this represents.

\[ y_1 \sim \{/y_1/7, /y_1/11, /y_1/13\} \]
\[ \sim \{-3, 2, 4\} \]
\[ \alpha_1 = /y_1/7 = -3 \sim \{-3, /-3/11, /-3/13\} \]
\[ \sim \{-3, -3, -3\} \]
\[ y_1 - \alpha_1 \sim \{0, /5/11, /7/13\} \]
\[ \sim \{0, 5, -6\} \]
\[ y_2 \sim \{(y_1 - \alpha_1)^{-1}(11)/11, (y_1 - \alpha_1)^{-1}(13)/13\} \]
\[ \sim \{/5.8/11, /-6.2/13\} \]
\[ \sim \{-4, 1\} \]
\[ \alpha_2 = \frac{y_2}{11} = -4 \sim \{-4, \ -4/13\} \]
\[ \sim \{-4, -4\} \]
\[ y_2 - \alpha_2 \sim \{0, \ 5/13\} \]
\[ \sim \{0, 5\} \]
\[ y_3 = \alpha_3 = \frac{(y_2 - \alpha_2)11^{-1}(13)}{13} \]
\[ = \frac{5\cdot6}{13} \]
\[ = 4 \]

Therefore,
\[ \langle \alpha_1, \alpha_2, \alpha_3 \rangle = \langle -3, -4, 4 \rangle, \]
and
\[ y = -3 + (-4)(7) + (4)(7)(11) \]
\[ = 277. \]

In terms of computer implementation, the unique symmetric mixed-radix representation can be determined from the unique multiple modulus residue representation using fixed-point arithmetic. This can be done in FORTRAN with the aid of the built-in FORTRAN MOD function and the function subprogram EUCLID, Function 7.2.1. The product-sum of Theorem 7.2.1 cannot be constructed in the computer fixed-point arithmetic mode because it will overflow the permissible fixed-point number range. This product-sum is formed using double precision floating-point arithmetic.

In Section 7.1, the potential problem of overflowing the floating-point exponent in construction of the product modulus bound was discussed.
If in the computation of the product modulus bound (without using logarithms) the floating-point exponent could overflow, then it could also overflow in the formation of the product-sum of Theorem 7.2.1. This is because the products \( \prod_{i=1}^{t-1} p_i, \prod_{i=1}^{t-2} p_i, \ldots, p_1p_2, p_1 \) are included in the product-sum. Unfortunately, the use of logarithms will not rectify this problem. One solution to this problem is to deflate this floating-point exponent.

Let \( \{p_1, p_2, \ldots, p_h\} \) be the multiple modulus base and let \( \langle a_1, a_2, \ldots, a_h \rangle \) be a symmetric mixed-radix representation with respect to this base. The moduli \( p_i, i = 1, 2, \ldots, h \), have the floating-point number representations

\[
p_i = \pm f_{p_i} \times \beta^{e_{p_i}}.
\]

The symmetric mixed-radix digits \( a_i, i = 1, 2, \ldots, h \), have the floating-point number representations

\[
a_i = \pm f_{a_i} \times \beta^{e_{a_i}}.
\]

The product-sum of Theorem 7.2.1 has the following floating-point number representation, (disregarding the possibility of overflowing the exponent),

\[
y = a_1 + a_2p_1 + \ldots + a_hp_1p_2 \cdots p_h-1 = \pm f_{y_1}f_{y_2} \cdots f_{y_t} \times \beta^{e_y}.
\]

For \( i = 1, 2, \ldots, h \), let
\[ \alpha_i^* = \alpha_i \cdot \beta^{-e}, \]

where \( e \in \mathbb{Z}^+ \), then the floating-point number representation of \( \alpha_i^* \) is

\[ \alpha_i^* = \pm .f_1 f_2 \ldots f_r \alpha_i \beta^e \]

We see that the floating-point fixed-radix digits of \( \alpha_i \) and \( \alpha_i^* \) are the same. The product-sum of Theorem 7.2.1 formed using \( \langle \alpha_1^*, \alpha_2^*, \ldots, \alpha_h^* \rangle \) is

\[ y^* = \alpha_1^* + \alpha_2^* p_1 + \ldots + \alpha_h^* p_1 p_2 \ldots p_{h-1} \]

\[ = (\alpha_1 + \alpha_2 p_1 + \ldots + \alpha_h p_1 p_2 \ldots p_{h-1}) \beta^{-e} \]

\[ = y \beta^{-e} \]

\[ = \pm .f_1 f_2 \ldots f_r \beta^{-e} \]

The floating-point fixed-radix points, \( f_{1_y}, f_{2_y}, \ldots, f_{t_y} \), are not affected when \( \langle \alpha_1^*, \alpha_2^*, \ldots, \alpha_h^* \rangle \) is used in this product-sum instead of \( \langle \alpha_1, \alpha_2, \ldots, \alpha_h \rangle \). The only thing that is affected is the floating-point exponent, it has been deflated. If the exponent is deflated enough so that \( e_y \beta^{-e} \) does not overflow the permissible floating-point exponent range, then the correct floating-point fixed-radix digits can be determined from the product-sum based on the \( \langle \alpha_1^*, \alpha_2^*, \ldots, \alpha_h^* \rangle \).

Recall that all of the elements of the exact solutions we are seeking are of the form \( \frac{x}{y}, x, y \in \mathbb{Z} \), (i.e., \( \hat{A} = \frac{1}{cf} A^*, cf \in \mathbb{Z} \) and \( A^* \) integral or \( \hat{b} = X^+ \frac{1}{cf} \hat{b}^*, cf \in \mathbb{Z} \) and \( \hat{b}^* \) integral). Consequently,
if we compute \( x^* \) and \( y^* \) by deflating the floating-point exponents in the product-sums formed with respect to the symmetric mixed-radix representations of \( x \) and \( y \) then,

\[
\begin{align*}
\hat{x}^* &= x^* \beta^{-e}, \\
\hat{y}^* &= y^* \beta^{-e},
\end{align*}
\]

and

\[
\frac{x}{y} = \frac{\hat{x}^*}{\hat{y}^*}
\]

is the exact solution.

If the deflation exponent, \( e \), is chosen such that

\[
0 \leq e \leq |m|,
\]

(7.2.2)

where \( m \in \mathbb{Z}^+ \) is the smallest permissible floating-point exponent, then it is not possible to overdeflate the floating-point exponent from the product-sum of Theorem 7.2.1. That means the deflated floating-point exponent will never underflow its permissible range. This is because the product-sum of Theorem 7.2.1 represents an integer; therefore, its floating-point exponent, \( e_y \), must always be positive. If \( e \) satisfies Equation 7.2.2 then,

\[
m < e_y - e.
\]

The best choice for \( e \) is \( e = |m| \) since this results in the maximum possible amount of deflation. Without deflating the exponent only exact solutions involving integers of magnitude less than \( \beta^m \), where \( m^* \) is the largest permissible floating-point exponent, could be computed. With deflation exact solutions involving integers on the order of \( \beta^{m^*} + |m| \) can be constructed.
Even after deflating the floating-point exponent it is still theoretically possible for the computation of product-sum of Theorem 7.2.1 to overflow the permissible floating-point number range. If this happened then the magnitude of the integers represented by \( <a_1, a_2, \ldots, a_h> \) is greater than \( \beta^\text{m} + |m| \). On the IBM 370 this would be integers greater than \( 10^{128} \). Given the product modulus

\[
M = \prod_{i=1}^{h} p_i
\]

the largest possible magnitude of the integers in the exact solution resulting from the multiple modulus residue representations with respect to the base \( \{p_1, p_2, \ldots, p_h\} \) is \( \frac{M}{2} \). The multiple modulus base, hence the product modulus, was chosen to exceed some bound based on the input to the problem. Therefore, the product modulus, \( M \), reflects the magnitude of the data and the size of the problem.

Consider the integral linear model \( y = Xb \). For a product modulus bound to exceed \( 10^{128} = \beta^m + |m| \), the magnitude of the entries in the \( X'X \) matrix would have to be huge or the dimensions of the \( X'X \) matrix (i.e., the number of parameters to be estimated) would have to be large. We do not feel that this particular limitation of the multiple modulus residue arithmetic methods is any more severe than limitation imposed by straight floating-point arithmetic methods.

This basically concludes the discussion of implementation of the multiple modulus residue arithmetic methods. There is an important fact that needs to be pointed out. The unique symmetric mixed-radix representation \( <a_1, a_2, \ldots, a_h> \) with respect to the multiple modulus
base \{p_1, p_2, \ldots, p_h\} is in itself an exact representation of some integer y in the range \( \frac{-M}{2}, \frac{M}{2} \), \( M = \prod_{i=1}^{h} p_i \). The final answer we look at is the deflated exponent product-sum

\[ y = \beta^{-e}(\alpha_1 + \alpha_2p_1 + \alpha_3p_1p_2 + \ldots + \alpha_hp_1p_2 \ldots p_{h-1}). \]

This product-sum is computed in double precision floating-point arithmetic. Consequently, rounding error will accumulate. The amount of deviation from the exact solution caused by accumulation of rounding error in the formation of this product-sum is not known. All of the numerical results thus far indicate that this is almost surely not a potential problem. However, it is a problem that needs to be researched further.

7.3 Examples

This section contains computer output for four examples. The first example is using a sample matrix from Rao et al. (1976). The second example is the example Stallings and Boullion (1972) used to illustrate the single modulus residue arithmetic application of Decell's algorithm. The third example can be found in Howell and Gregory (1970), it is a classical form of the types of matrices used to test matrix inversion procedures. For the first three examples, for the input matrix A, \( A^{-} \) is computed using the Gaussian Elimination program, \((A'A)^{-}\) is found using the Bordering program, and \( A^{+} \) is formed using the Moore-Penrose program. In these first three examples the floating-point
exponents from the formation of the product-sums as given in Theorem 7.2.1 are not deflated. The fourth example is a linear least squares example. The data used are the Longley (1967) data. In the Longley data the independent variables are very highly correlated with each other; consequently, standard regression packages may have difficulty computing the correct linear least squares solution. In example four, we will compute the exact linear least squares solution for these data by using the Minimum Euclidean Norm Least Squares program. The data have been scaled to integers by multiplying both the \( X \) matrix and the \( y \) vector by 10. In this example, the floating-point exponents from the formation of the product-sums as given in Theorem 7.2.1 have been deflated.
Example 7.3.1a  Gaussian Elimination Method

THE INPUT MATRIX A IS AS FOLLOWS:

\[
\begin{bmatrix}
22 & 14 & -1 & -3 & 1 & 9 & 2 & 4 \\
10 & 7 & 13 & -2 & 8 & 1 & -6 & 5 \\
2 & 10 & -1 & 13 & 1 & -7 & 6 & 0 \\
3 & 0 & -11 & -2 & 5 & 5 & -2 \\
7 & 8 & 3 & 4 & 4 & -1 & 1 & 2
\end{bmatrix}
\]

THE NUMBER OF PRIME MODULI USED RANK(A), P, AND Q = 2

THE RANK OF A = 4

\[
P = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\quad Q = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

THE NUMBER OF PRIME MODULI USED TO INVERT THE NONSINGULAR MINOR = 2

\[
\tilde{A} = \left( 1 / -10856.0 \right) ^* \\
\begin{bmatrix}
-529.0 & -52.0 & 777.0 & -84.0 & 0.0 \\
-69.0 & 260.0 & -1171.0 & 420.0 & 0.0 \\
-391.0 & 372.0 & 287.0 & 1436.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
1357.0 & -2124.0 & -413.0 & -2596.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0
\end{bmatrix}
\]

\[
= \begin{bmatrix}
0.0487288 & 0.0047900 & -0.0715733 & 0.0077377 & 0.0 \\
0.0063559 & -0.0239499 & 0.1078666 & -0.0386883 & 0.0 \\
0.0360169 & -0.0342668 & -0.0264370 & -0.1322771 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
-0.1250000 & 0.1956522 & 0.0380435 & 0.2391304 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0
\end{bmatrix}
\]
Example 7.3.1.b  Bordering Method

THE INPUT MATRIX A'A IS AS FOLLOWS:

\[
\begin{pmatrix}
646 & 454 & 94 & -38 & 126 & 202 & 18 & 146 \\
454 & 409 & 91 & 106 & 112 & 55 & 54 & 107 \\
94 & 91 & 301 & -2 & 136 & -47 & -138 & 89 \\
-38 & 106 & -2 & 202 & 14 & -134 & 78 & -10 \\
126 & 112 & 136 & 14 & 86 & -4 & -46 & 56 \\
202 & 55 & -47 & -134 & -4 & 157 & -6 & 29 \\
18 & 54 & -138 & 78 & -46 & -6 & 102 & -30 \\
146 & 107 & 89 & -10 & 56 & 29 & -30 & 49 \\
\end{pmatrix}
\]

THE NUMBER OF PRIME MODULI USED IN THE BORDERING ALGORITHM = 5
RANK(A) = 4
THE NUMBER OF PRIME MODULI USED TO INVERT THE NONSINGULAR MINOR = 3

\[
P = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{pmatrix}
\]

\[
A'A = \left( \frac{1}{183128448.0} \right) \begin{pmatrix}
1306824.0 & -1362888.0 & 234888.0 & 0.0 & -511176.0 & 0.0 & 0.0 & 0.0 \\
-1362888.0 & 2456936.0 & 792856.0 & 0.0 & -2456760.0 & 0.0 & 0.0 & 0.0 \\
234888.0 & 792856.0 & 3213416.0 & 0.0 & -6458376.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
-511176.0 & -2456760.0 & -6458376.0 & 0.0 & 16291080.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
\end{pmatrix}
\]

\[
A' A^{-1} = \begin{pmatrix}
0.0071361 & -0.0047423 & 0.0012826 & 0.0 & -0.0027913 & 0.0 & 0.0 & 0.0 \\
-0.0047423 & 0.0134165 & 0.0043295 & 0.0 & -0.0134155 & 0.0 & 0.0 & 0.0 \\
0.0012826 & 0.0043295 & 0.0175473 & 0.0 & -0.0352669 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
-0.0027914 & -0.0134155 & -0.0352669 & 0.0 & 0.08895985 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
\end{pmatrix}
\]
Example 7.3.1c  Moore–Penrose Inversion Method

THE INPUT MATRIX A IS AS FOLLOWS:

\[
\begin{bmatrix}
22 & 14 & -1 & -3 & 1 & 9 & 2 & 4 \\
10 & 7 & 13 & -2 & 8 & 1 & -6 & 5 \\
2 & 10 & -1 & 13 & 1 & -7 & 6 & 0 \\
3 & 0 & -11 & -2 & 5 & 5 & -2 \\
7 & 8 & 3 & 4 & 4 & -1 & 1 & 2
\end{bmatrix}
\]

NUMBER OF PRIME MODULI USED = 4
THE RANK OF A = 4

\[
+ A = (1 / -1831600128.0) * \\
\begin{bmatrix}
-60604416.0 & 7004160.0 & 1908736.0 & 19517440.0 & 10424320.0 \\
-43769856.0 & 18567168.0 & -40361984.0 & 42115072.0 & 4956160.0 \\
-50503680.0 & 17952768.0 & -1032192.0 & 163246080.0 & 58601472.0 \\
6733824.0 & 18161664.0 & -62865408.0 & 22659072.0 & -10960896.0 \\
228950016.0 & -240648192.0 & 28409856.0 & -369328128.0 & -225607680.0 \\
-10100736.0 & -13455360.0 & 39010304.0 & -49795072.0 & -5214208.0 \\
20201472.0 & -663552.0 & -24682496.0 & -69128192.0 & -32116736.0 \\
-23568384.0 & 5369856.0 & 827392.0 & 41992192.0 & 15941632.0
\end{bmatrix}
\]

= \[
\begin{bmatrix}
0.033088235 & -0.003824066 & -0.001042114 & -0.010655950 & -0.005691373 \\
0.023897059 & -0.010137130 & 0.022036461 & -0.022993595 & -0.002705918 \\
0.027573529 & -0.009801685 & 0.000563547 & -0.089127576 & -0.031994687 \\
0.003676471 & -0.009915736 & 0.034322671 & -0.012371189 & 0.005984328 \\
-0.125000000 & 0.131386861 & -0.015510949 & 0.201642336 & 0.123175182 \\
0.005514706 & 0.007346232 & -0.021298483 & 0.027186650 & 0.002846805 \\
-0.011029412 & 0.000362280 & 0.013475920 & 0.037741967 & 0.017534797 \\
0.012867647 & -0.002931784 & -0.000451732 & -0.022926506 & -0.008703664
\end{bmatrix}
\]
Example 7.3.2a Gaussian Elimination Method

THE INPUT MATRIX A IS AS FOLLOWS:

\[
\begin{bmatrix}
1 & 0 & 50 & 0 \\
0 & 2 & 0 & 1 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

THE NUMBER OF PRIME MODULI USED RANK(A), P, AND Q = 1
THE RANK OF A = 3

\[
P = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\quad
Q = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

THE NUMBER OF PRIME MODULI USED TO INVERT THE NONSINGULAR MINOR = 1

\[
A^{-1} = \frac{1}{-100.0} \times
\begin{bmatrix}
0.0 & 0.0 & -100.0 \\
0.0 & -50.0 & 0.0 \\
-2.0 & 0.0 & 2.0 \\
0.0 & 0.0 & 0.0
\end{bmatrix}
\]

\[
= \begin{bmatrix}
0.0 & 0.0 & 1.00 \\
0.0 & 0.50 & 0.0 \\
0.02 & 0.0 & -0.02 \\
0.0 & 0.0 & 0.0
\end{bmatrix}
\]
Example 7.3.2b  Bordering Method

THE INPUT MATRIX $A^TA$ IS AS FOLLOWS:

\[
\begin{bmatrix}
2 & 0 & 50 & 0 \\
0 & 4 & 0 & 2 \\
50 & 0 & 2500 & 0 \\
0 & 2 & 0 & 1 \\
\end{bmatrix}
\]

THE NUMBER OF PRIME MODULI USED IN THE BORDERING ALGORITHM = 2

$RANK(A) = 3$

THE NUMBER OF PRIME MODULI USED TO INVERT THE NONSINGULAR MINOR = 2

$P = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}$

$A^TA = \left( \frac{1}{10000.0} \right) \ast$

\[
\begin{bmatrix}
10000.0 & 0.0 & -200.0 & 0.0 \\
0.0 & 2500.0 & 0.0 & 0.0 \\
-200.0 & 0.0 & 8.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 \\
\end{bmatrix}
\]

= \[
\begin{bmatrix}
1.0000 & 0.0 & -0.0200 & 0.0 \\
0.0 & 0.2500 & 0.0 & 0.0 \\
-0.0200 & 0.0 & 0.0008 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 \\
\end{bmatrix}
\]
Example 7.3.2c  Moore-Penrose Inversion Method

The input matrix A is as follows:

\[
\begin{pmatrix}
1 & 0 & 50 & 0 \\
0 & 2 & 0 & 1 \\
1 & 0 & 0 & 0
\end{pmatrix}
\]

Number of prime moduli used = 3
The rank of A = 3

\[
A = \left( \frac{1}{12500.0} \right) \times
\begin{pmatrix}
0.0 & 0.0 & 0.0 & 12500.0 \\
0.0 & 5000.0 & 0.0 & 0.0 \\
250.0 & 0.0 & 0.0 & -250.0 \\
0.0 & 2500.0 & 0.0 & 0.0
\end{pmatrix}
\]

= 
\[
\begin{pmatrix}
0.0 & 0.0 & 1.00 \\
0.0 & 0.40 & 0.0 \\
0.02 & 0.0 & -0.02 \\
0.0 & 0.20 & 0.0
\end{pmatrix}
\]
### Example 7.3.3a

**Gaussian Elimination Method**

The input matrix $A$ is as follows:

$$
\begin{bmatrix}
10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 28147 \\
9 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\
8 & 8 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\
7 & 7 & 7 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\
6 & 6 & 6 & 6 & 6 & 5 & 4 & 3 & 2 & 1 \\
5 & 5 & 5 & 5 & 5 & 5 & 4 & 3 & 2 & 1 \\
4 & 4 & 4 & 4 & 4 & 4 & 4 & 3 & 2 & 1 \\
3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 2 & 1 \\
2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{bmatrix}
$$

The number of prime moduli used $\text{rank}(A)$, $P$, and $Q = 4$

The rank of $A = 10$

$P = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}$

$Q = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}$

The number of prime moduli used to invert the nonsingular minor $= 4$

$A^- = \left( \frac{1}{1.0} \right) *$

$$
\begin{bmatrix}
1.0 & -1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 28146.0 & -56292.0 \\
-1.0 & 2.0 & -1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -28146.0 & 56292.0 \\
0.0 & -1.0 & 2.0 & -1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & -1.0 & 2.0 & -1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & -1.0 & 2.0 & -1.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.0 & 2.0 & -1.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.0 & 2.0 & -1.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.0 & 2.0 & -1.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.0 & 2.0 \\
\end{bmatrix}
$$
Example 7.3.3b Bordering Method

The input matrix $A'A$ is as follows:

$$
\begin{bmatrix}
375 & 366 & 348 & 322 & 289 & 250 & 206 & 158 & 107 & 253368 \\
356 & 348 & 332 & 308 & 277 & 240 & 198 & 152 & 103 & 225220 \\
329 & 322 & 308 & 287 & 259 & 225 & 186 & 143 & 97 & 197071 \\
295 & 289 & 277 & 259 & 235 & 205 & 170 & 131 & 89 & 168921 \\
255 & 250 & 240 & 225 & 205 & 180 & 150 & 116 & 79 & 140770 \\
210 & 206 & 198 & 186 & 170 & 150 & 116 & 79 & 53 & 112618 \\
161 & 158 & 152 & 143 & 131 & 116 & 98 & 77 & 53 & 84465 \\
109 & 107 & 103 & 97 & 89 & 79 & 67 & 53 & 37 & 56311 \\
\end{bmatrix}
$$

The number of prime moduli used in the bordering algorithm = 12

$\text{RANK}(A) = 10$

The number of prime moduli used to invert the nonsingular minor = 12

$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
\end{bmatrix}$

$A'A^- = (1 / 1.0) \times$

$$
\begin{bmatrix}
3960986582 & -3960986583 & 1. & 0. & 0. & 0. & 0. & -28146 & 112584 & -140730 \\
-3960986583 & 3960986586 & -4. & 1. & 0. & 0. & 0. & 28146 & -112584 & 140730 \\
1 & -4. & 6. & -4. & 1. & 0. & 0. & 0. & 1. & 0. \\
0 & 1. & -4. & 6. & -4. & 1. & 0. & 0. & 1. & 0. \\
0 & 0. & 1. & -4. & 6. & -4. & 1. & 0. & 1. & 0. \\
0 & 0. & 0. & 1. & -4. & 6. & -4. & 1. & 1. & 0. \\
-28146 & 28146 & 0. & 0. & 1. & -4. & 6. & -4. & 1. & 1. \\
112584 & -112584 & 0. & 0. & 0. & 1. & -4. & 6. & -4. & 1. \\
-140730 & 140730 & 0. & 0. & 0. & 0. & 1. & -4. & 5. & 1. \\
\end{bmatrix}
$$

= $\begin{bmatrix}
3960986582 & -3960986583 & 1. & 0. & 0. & 0. & 0. & -28146 & 112584 & -140730 \\
-3960986583 & 3960986586 & -4. & 1. & 0. & 0. & 0. & 28146 & -112584 & 140730 \\
1 & -4. & 6. & -4. & 1. & 0. & 0. & 0. & 1. & 0. \\
0 & 1. & -4. & 6. & -4. & 1. & 0. & 0. & 1. & 0. \\
0 & 0. & 1. & -4. & 6. & -4. & 1. & 0. & 1. & 0. \\
0 & 0. & 0. & 1. & -4. & 6. & -4. & 1. & 1. & 0. \\
-28146 & 28146 & 0. & 0. & 1. & -4. & 6. & -4. & 1. & 1. \\
112584 & -112584 & 0. & 0. & 0. & 1. & -4. & 6. & -4. & 1. \\
-140730 & 140730 & 0. & 0. & 0. & 0. & 1. & -4. & 5. & 1. \\
\end{bmatrix}$
Example 7.3.3c  
**Moore-Penrose Inversion Method**

**THE INPUT MATRIX A IS AS FOLLOWS:**

\[
\begin{array}{ccccccccccc}
10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 28147 \\
9 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\
8 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\
7 & 7 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\
6 & 6 & 6 & 6 & 5 & 4 & 3 & 2 & 1 \\
5 & 5 & 5 & 5 & 5 & 4 & 3 & 2 & 1 \\
4 & 4 & 4 & 4 & 4 & 4 & 3 & 2 & 1 \\
3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 2 & 1 \\
2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

**NUMBER OF PRIME MODULI USED = 20**

**THE RANK OF A = 10**

\[
A = \left( \frac{1}{-1.0} \right) \times
\begin{array}{cccccccccccc}
-1.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -28146.0 & 56292.0 \\
1.0 & -2.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 28146.0 & -56292.0 \\
0.0 & 1.0 & -2.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 1.0 & -2.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 1.0 & -2.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 1.0 & -2.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & -2.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & -2.0 & 1.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & -2.0 & 1.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & -2.0 & 1.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & -2.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & -2.0 \\
\end{array}
\]

=  
\[
\begin{array}{cccccccccccc}
1.0 & -1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 28146.0 & -56292.0 \\
-1.0 & 2.0 & -1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -28146.0 & 56292.0 \\
0.0 & -1.0 & 2.0 & -1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & -1.0 & 2.0 & -1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & -1.0 & 2.0 & -1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.0 & 2.0 & -1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.0 & 2.0 & -1.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.0 & 2.0 & -1.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.0 & 2.0 & -1.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & -2.0 & 0.0 \\
\end{array}
\]
Minimum Euclidean Norm Least Squares Method with Floating-Point Exponent Deflation.

The input data for the integral linear model $Y = XB$ are as follows:

<table>
<thead>
<tr>
<th>Y</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>603230</td>
<td>10 830 2342890 23560 15900</td>
</tr>
<tr>
<td>611220</td>
<td>10 885 2594260 23250 14560</td>
</tr>
<tr>
<td>601710</td>
<td>10 882 2580540 36820 16160</td>
</tr>
<tr>
<td>611870</td>
<td>10 895 2843990 33510 16500</td>
</tr>
<tr>
<td>632210</td>
<td>10 962 3289750 20990 30990</td>
</tr>
<tr>
<td>636390</td>
<td>10 981 3469990 19320 35940</td>
</tr>
<tr>
<td>649890</td>
<td>10 990 3653850 18700 35470</td>
</tr>
<tr>
<td>637610</td>
<td>10 1000 3631120 35780 33500</td>
</tr>
<tr>
<td>660190</td>
<td>10 1012 3974690 29040 30480</td>
</tr>
<tr>
<td>678570</td>
<td>10 1046 4191800 28220 28570</td>
</tr>
<tr>
<td>681690</td>
<td>10 1084 4427690 29360 27980</td>
</tr>
<tr>
<td>665130</td>
<td>10 1108 4445460 46810 26370</td>
</tr>
<tr>
<td>686550</td>
<td>10 1126 4827040 38130 25520</td>
</tr>
<tr>
<td>695640</td>
<td>10 1142 5026010 39310 25140</td>
</tr>
<tr>
<td>693310</td>
<td>10 1157 5181730 48060 25720</td>
</tr>
<tr>
<td>705510</td>
<td>10 1169 5548940 40070 28270</td>
</tr>
</tbody>
</table>

The number of prime moduli used = 19
The rank of $A = 7$

$XY = \left( \begin{array}{c}
\frac{1}{0.212285084989D-28} \\
-0.739231570199D-22 \\
-0.319741083522D-27 \\
-0.760387752037D-30 \\
-0.428864655601D-28 \\
-0.219338653311D-29 \\
-0.108486394120D-29 \\
0.388301574123D-25 \\
\end{array} \right)$

$= -3482258.634596$
$15.061872271373$
$-0.035819179293$
$-2.020229803817$
$-1.033226867174$
$-0.051104105654$
$1829.1514646136$
8. BIBLIOGRAPHY


Longley, James W. 1967. An appraisal of least squares problems for
the electronic computer from the point of view of the user,

Lavoie, J. L. 1980. A determinantal inequality involving the Moore-

University of Maryland Computer Science Center Report TR-64-7,
College Park, Maryland.

McClellan, Michael T. 1973. The exact solution of systems of linear
equations with polynomial coefficients. Journal of the Association

McClellan, Michael T. 1977a. The exact solution of linear equations
with rational function coefficients. ACM Transactions on Mathematical

McClellan, Michael T. 1977b. A comparison of algorithms for the exact
solution of linear equations. ACM Transactions on Mathematical
Software, 3(2):147-158.

of the Association for Computing Machinery, 22(2):291-308.

Musser, David R. 1978. On the efficiency of a polynomial irreducibility
test. Journal of the Association for Computing Machinery, 25(2):
271-282.

of the National Bureau of Standards-B. Mathematics and Mathe­

application to statistics. Bulletin of Mathematical Statistics,
18(3):45-49.

O'Keefe, Kenneth H. 1975. A note on fast base extension for residue
number systems with three moduli. IEEE Transactions on Computers,

Okeke, C. C. 1979. Exact solutions for linear matrix equations. The
Matrix and Tensor Quarterly, 30(2):51-54.

Pearl, Martin H. 1968. Generalized inverses of matrices with entries
taken from an arbitrary field. Linear Algebra and its Applications,
1:571-587.


9. ACKNOWLEDGMENTS

I wish to acknowledge and thank Professor William J. Kennedy for suggesting this area of research, for his encouragement, and for his valuable advice.

I also wish to acknowledge the support given by Professor J. J. Higgins throughout the years of my graduate study.

I wish to thank Mrs. Darlene Wicks for her patience and skill in the typing of this manuscript.

Finally, I wish to express my sincere thanks to my parents, Leo and Liz Keller, for their understanding, encouragement, and moral support throughout the years of my education.
10. APPENDIX

10.1 The Gaussian Elimination Method FORTRAN Program

/1 EXEC FORTG
IMPLICIT INTEGER (A-Z)
DIMENSION B(5,8),PRIME(20),WF1(8),WF2(8),WF3(8),WF4(8),
*BMOD(5,8),WMAT1(5,5),WMAT2(8,8),P(5,5),Q(8,8),RANK(20),
*W(8),ADJF(8,5),ADJ(5,5,20),DET(20)
REAL*8 WF1,WF2,WF3,WF4,ADJF,DET

THIS PROGRAM COMPUTES A REFLEXIVE INVERSE OF AN (N X S) MATRIX B
USING A MODIFIED GAUSSIAN ELIMINATION METHOD BASED ON
MULTIPLE MODULUS RESIDUE ARITHMETIC.

THIS PARTICULAR "MAIN" PROGRAM IS SETUP TO COMPUTE A REFLEXIVE
GENERALIZED INVERSE FOR THE MATRIX OF EXAMPLE 7.3.1.

THE ARRAY PRIME CONTAINS THE POSSIBLE PRIME MODULI THAT WILL
BE USED.

DATA PRIME/45233,45247,45259,45263,45281,45289,
145293,45307,45317,45319,45329,45337,45341,45343,45361,
245377,45389,45403,45413,45427/
N=5
S=8

INPUT N X S MATRIX B

DO 1 I=1,N
READ(5,100) (B(I,J),J=1,S)
1 CONTINUE
WRITE(6,300)
DO 2 I=1,N
2 WRITE(6,200) (B(I,J),J=1,S)

COMPUTE THE INITIAL PRODUCT MODULUS BOUND.

CALL SCHINZ(N,S,M,PRIME,B,WFl,WF2)
WRITE(6,400) M

COMPUTE THE REFLEXIVE INVERSE OF B.

CALL PBQ(N,S,M,PRIME,W,B,WFl,WF2,WF3,WF4,ADJF,B,WMAT1,WMAT2,P,Q,
*BMOD,ADJ,DET,RANK,R,DETF)
CGC THE FORMATS USED HERE FOR OUTPUT HAVE BEEN SPECIFIED WITH
CGC EXAMPLE 7.3.1 IN MIND.

WRITE(6,500) R
WRITE(6,600) (P(1,J),J=1,N),(Q(1,J),J=1,S)
DO 3 I=2,N
3 WRITE(6,700) (P(I,J),J=1,N),(Q(I,J),J=1,S)
DO 17 I=6,8
17 WRITE(6,906) (Q(I,J),J=1,S)
WRITE(6,901) M
WRITE(6,902) DETF
DO 5 I=1,S
5 WRITE(6,903) (ADJF(I,J),J=1,N)
WRITE(6,904)
DO 4 I=1,S
DO 6 J=1,N
6 ADJF(I,J)=ADJFCI,J)/DETF
WRITE(6,905) (ADJF(I,J),J=1,N)
4 CONTINUE
100 FORMAT(8I4)
200 FORMAT(' ',13X,8I5)
300 FORMAT('1',/',13X,' THE INPUT MATRIX A IS AS FOLLOWS: ')
400 FORMAT(' ',13X,' THE NUMBER OF PRIME MODULI USED RANK(A), P, AND, '
* 'Q = ',13X)
500 FORMAT(' ',13X,' THE RANK OF A = ',13X)
600 FORMAT('0',13X,' P = ',513,' Q = ',813)
700 FORMAT(' ',13X,3X,513,6X,813)
906 FORMAT(' ',13X,3X,15X,6X,813)
901 FORMAT(' ',13X,' THE NUMBER OF PRIME MODULI USED TO INVERT THE ',
* ' NONSINGULAR MINOR = ',13X)
902 FORMAT(' ',13X,' A = ( 1/',13X,'.F9.1,' ) *')
903 FORMAT(' ',13X,4X,4(F10.1,2X),F4.1)
904 FORMAT(' ',13X,3X,' = ')  
905 FORMAT(' ',13X,4X,4(F10.7,2X),F4.1)
STOP

SUBROUTINE SCHINZ(CN,S,M,PRIME,B,RPLUS,RMIN)
IMPLICIT INTEGER (A-Z)
DIMENSION B(N,S),PRIME(20),RPLUS(N),RMIN(N)
REAL*R8 RPLUS,RMIN,BDD

THE ARRAY RPLUS(N) CONTAINS THE SUM OF THE POSITIVE B(I,J) ELEMENTS ACROSS EACH ROW(I), I=1,N.

THIS SUBROUTINE COMPUTES A MODIFIED SCHINZEL'S BOUND FOR THE DETERMINANT OF ANY MINOR OF THE N X S (N<=S) MATRIX B.
THE ARRAY RMIN(N) CONTAINS THE SUM OF THE ABSOLUTE VALUE OF THE NEGATIVE B(I,J) ELEMENTS ACROSS EACH ROW(I), I=1,N.

BDD=MAX(RPLUS(1),RMIN(1))*MAX(RPLUS(2),RMIN(2))*...*MAX(RPLUS(N),RMIN(N))

IT IS ASSUMED THAT B HAS NO NULL ROWS, IF A ROW OF B IS NULL

RPLUS(N) AND RMIN(N) ARE COMPUTED HERE

DO 1 I=1,N
    RPLUS(I)=0.D0
    RMIN(I)=0.D0
    DO 2 J=1,S
        RPLUS(I)=RPLUS(I)+AMAX0(0,B(I,J))
        RMIN(I)=RMIN(I)-AMIN0(0,B(I,J))
    2 CONTINUE

BDD IS FORMED HERE.

BDD=BDD*DMAX1(RPLUS(I),RMIN(I),1.D0)

THE SUBROUTINE SELECT IS THEN CALLED TO CHOOSE THE NUMBER OF PRIMES (M) NEEDED FROM THE TABLE OF PRIMES TO GENERATE A LARGEST NONSINGULAR MINOR USING MULTIPLE MODULUS RESIDU ARITHMETIC.

BDD=2*BDD
CALL SELECT(BDD,M,PRIME)
RETURN
END

SUBROUTINE SELECT(BDD,M,PRIME)
IMPLICIT INTEGER (A-Z)
DIMENSION PRIME(20)
REAL*8 BDD,PPROD

This subroutine computes the number of prime moduli needed to solve the given problem. The possible prime moduli are stored in the array PRIME. SELECT computes the number M which is the number of prime numbers that need to be multiplied.
TOGETHER TO INSURE A CORRECT RATIONAL SOLUTION TO THE
MULTIPLE MODULUS PROBLEM CAN BE FOUND.

IF FOR EXAMPLE $M=4$, \( \text{PRIME}(1) \times \text{PRIME}(2) \times \text{PRIME}(3) \times \text{PRIME}(4) > \text{BDD} \).

C

PPROD=1
DO 1 I=1,10
PPROD=PPROD*PRIME(I)
IF(PPROD.GE.BDD) GO TO 2
1 CONTINUE
2 M=I
RETURN
END

SUBROUTINE PBQ(N,S,M,PRIME,W,WF1,WF2,WF3,WF4,ADJF,B,WP,WQ,PP,QQ,
* BMOD,ADJ,DETF)
IMPLICIT INTEGER (A-Z)
DIMENSION PRIME(20),B(N,S),WP(N,N),WQ(S,S),BMOD(N,S),W(S),
*PP(N,N),QQ(S,S),RANK(M),WF1(S),WF2(N),WF3(N),WF4(N),ADJF(S,N),
*ADJ(N,N,M),DETF(N)
REAL*8 WF1,WF2,WF3,WF4,ADJF,DETF

THIS SUBROUTINE FIRST FINDS PERMUTATION MATRICES PP AND QQ
SUCH THAT

\[
\begin{pmatrix}
B(1,1) & B(1,2) \\
B(2,1) & B(2,2)
\end{pmatrix}
\]

WHERE \( \text{ORDER}(B(1,1))=\text{RANK}(B(1,1))=\text{RANK}(B)=R \) USING MULTIPLE
MODULUS RESIDUE ARITHMETIC.

IF \( M \) THE NUMBER OF PRIMES BEING USED HAS BEEN CHOSEN CORRECTLY,
THEN THERE EXISTS AT LEAST ONE PRIME, \( P \), SUCH THAT
\( \text{RANK}(B \text{ MOD}(P))=\text{RANK}(B)=R \).

THE PERMUTATION MATRICES WP AND WQ ARE SUCH THAT

\[
\begin{pmatrix}
B(1,1) \text{ MOD}(P) & B(1,2) \text{ MOD}(P) \\
B(2,1) \text{ MOD}(P) & B(2,2) \text{ MOD}(P)
\end{pmatrix}
\]

HENCE BY SETTING \( PP=W, QQ=W \) AND \( R=\text{RANK}(B \text{ MOD}(P)) \) THE FIRST
STEP OF THE PROBLEM IS COMPLETED.
CONTINUE

NEXT ADJ(B(1,1)) AND THE DET(B(1,1)) IS COMPUTED USING MULTIPLE MODULUS RESIDUE ARITHMETIC.

FINALLY THE REFLEXIVE INVERSE OF B IS FORMED AS

\[
B^{-1} = \begin{bmatrix}
1 & \text{ADJ}(B(1,1)) & 0 \\
\text{DET}(B(1,1)) & 0 & 0
\end{bmatrix}
\]

WP, WQ, AND RANK(B MOD(P)) = RANK(P) ARE FOUND FOR EACH PRIME.

DO 100 P = 1, M

INITIALIZE WP = I

DO 1 I = 1, N
  DO 1 J = 1, N
    WP(I,J) = 0
    IF (I.EQ.J) WP(I,J) = 1
  1 CONTINUE

INITIALIZE WQ = I

DO 2 I = 1, S
  DO 2 J = 1, S
    WQ(I,J) = 0
    IF (I.EQ.J) WQ(I,J) = 1
  2 CONTINUE

SET RANK(P) = N, RANK(P) WILL BE DECREASED AS RANK DEFICIENCIES ARE LOCATED.

RANK(P) = N

B MOD(P) WILL BE STORED IN BMOD

DO 3 I = 1, N
  DO 3 J = 1, S
    BMOD(I,J) = MOD(B(I,J), PRIME(P))
    IF (BMOD(I,J).LT.0) BMOD(I,J) = BMOD(I,J) + PRIME(P)
  3 CONTINUE
THE SEARCH FOR B(1,1) MOD(P) STARTS HERE

DO 4 I=1,N

IF THE (1,1) DIAGONAL ELEMENT OF B MOD(P) IS ZERO, ROW I IS
SEARCHED FOR A NONZERO ELEMENT.

IF(BMOD(I,I).NE.0) GO TO 8
DO 5 J=I,N
  DO 6 L=I,S
    IF(BMOD(I,L).EQ.0) GO TO 6

IF A NONZERO ELEMENT IN ROW I IS FOUND, SAY IN COLUMN K, THEN
COLUMNS I AND K OF BOTH B MOD(P) AND WQ ARE INTERCHANGED.

DO 7 K=1,N
  W(K)=BMOD(K,I)
  BMOD(K,I)=BMOD(K,L)
  BMOD(K,L)=W(K)
7 CONTINUE

DO 77 K=1,S
  W(K)=WQ(K,I)
  WQ(K,I)=WQ(K,L)
  WQ(K,L)=W(K)
77 CONTINUE

GO TO 8
6 CONTINUE

IF NO NONZERO ELEMENT IS FOUND IN ROW I OF B MOD(P) THEN
ROW I IS INTERCHANGED WITH A NONNULL ROW OF B MOD(P) IF
ONE EXISTS.

THE FIRST TIME THIS STEP IS EXECUTED ROW I AND ROW RANK(P)=N
ARE INTERCHANGED IN BOTH B MOD(P) AND WP. RANK(P) IS THEN
DECREASED BY 1.

IN GENERAL ROWS I AND RANK(P) ARE INTERCHANGED AND RANK(P) IS
DECREASED BY 1.

IF RANK(P) = I NO NONNULL ROWS ARE LEFT AND
RANK(P)=RANK(P)-1=RANK(B MOD(P)).

IF(I.NE.RANK(P)) GO TO 9
RANK(P)=RANK(P)-1
GO TO 10
9 DO 11 K=1,S
  BMOD(I,K)=BMOD(RANK(P),K)
  BMOD(RANK(P),K)=0
11 CONTINUE
DO 111 K=1,N
   W(K)=WP(I,K)
   WP(I,K)=WP(RANK(P),K)
   WP(RANK(P),K)=W(K)
111 CONTINUE
   RANK(P)=RANK(P)-1
   CONTINUE

CCC IF RANK(P) DOES NOT EQUAL RANK(B MOD(P)) A PIVOT IS EXECUTED
CCC
   8 IF(I.EQ.RANK(P)) GO TO 10
      INV=EUCLID(BMOD(I,I),PRIME(P))
CCC
   INV = BMOD(I,I) INVERSE MOD(P)
CCC
      DO 12 K=1,S
         BMOD(I,K)=MOD( BMOD(I,K)*INV, PRIME(P))
12    CONTINUE
      IP1=I+1
CCC
   ELEMENTS BELOW BMOD(I,I) ARE ANNIHILATED
CCC
      DO 13 I=IP1,N
         IF(BMOD(L,I).EQ.0) GO TO 13
         PIV=PRIME(P)-BMOD(L,I)
         DO 13 K=1,S
            BMOD(L,K)=MOD( PIV*BMOD(I,K)+BMOD(L,K), PRIME(P))
13     CONTINUE
CCC
   PIVOT IS COMPLETE AND A NEW ROW TO PIVOT ON IS SEARCHED FOR
CCC
   4 CONTINUE
CCC
   INITIALLY RANK(B)=R IS SET TO ZERO.
CCC
   THE PRIME THAT GENERATES THE LARGEST RANK(B MOD(P)) IS ACTUALLY
   GENERATING RANK(B)=R. NOTE: MORE THAN ONE PRIME MAY GENERATE
   THIS RANK.
CCC
   IF RANK(P) IS GREATER THAN THE PREVIOUS R,
   R=MAX(RANK(1),RANK(2),...,RANK(P-1)),
   THEN R IS SET TO RANK(P), PP IS SET TO WP AND QQ IS SET TO WQ.
CCC
   10 IF(RANK(P).LE.R) GO TO 100
      R=RANK(P)
      DO 14 I=1,N
      DO 14 J=1,N
         PP(I,J)=WP(I,J)
14    CONTINUE
DO 15 I=1,S
DO 15 J=1,S
  QQ(I,J)=WQ(I,J)
15  CONTINUE
100  CONTINUE

CCC
THE SEARCH FOR B(1,1) MOD(P) HAS BEEN COMPLETED FOR EACH PRIME.
CCC
PP, QQ AND R HAVE BEEN FOUND.
CCC
THE PRODUCT PP*B*QQ IS NOW FORMED AND B(1,1) IS STORED IN THE
CCC UPPER R X R LEFT HAND CORNER OF BMOD.
CCC
DO 16 I=1,R
DO 16 J=1,S
  WQ(I,J)=0
DO 16 K=1,N
  WQ(I,J)=PP(I,K)*B(K,J)+WQ(I,J)
16  CONTINUE
DO 18 I=1,R
DO 18 J=1,R
  BMOD(I,J)=0
DO 18 K=1,S
  BMOD(I,J)=WQ(I,K)*QQ(K,J)+BMOD(I,J)
18  CONTINUE

CCC
B(1,1) VIA BMOD IS SENT IN THE SUBROUTINE JHNEW. JHNEW WILL
CCC COMPUTE JOHNSON AND NEWMAN'S BOUND FOR THE DET(B(1,1)) AND
CCC FOR THE COFACTORS OF B(1,1).
CCC
A SUBSET OF THE ORIGINAL SET OF PRIME MODULI (NOT NECESSARILY
CCC A PROPER SUBSET) WILL BE CHosen TO BE USED IN THE
CCC MULTIPLE MODULUS INVERSION OF B(1,1).
CCC
CALL JHNEW(N,S,R,M,PRIME,WF1,WF2,WF3,WF4,BM0D,ADJF)
CCC
FOR EACH PRIME, ADJ(B(1,1)) MOD(P) AND THE PRODUCT OF THE
CCC NONZERO PIVOTS MOD(P) IS COMPUTED AND STORED IN ADJ(I,J,P) AND
CCC DET(P), I=1,R AND J=1,R.
CCC
IF RANK(P) < R-1 THEN ADJ(B(1,1),P) = 0 AND DET(P)=0.
CCC
IF RANK(P) = R-1 THEN DET(P) = 0 AND EXTENDED GAUSS JORDAN
CCC ELIMINATION IS PERFORMED OF B(1,1) MOD(P) TO
CCC FIND ADJ(B(1,1),P).
CCC
IF RANK(P) = R GAUSS JORDAN ELIMINATION IS PERFORMED ON
CCC B(1,1) MOD(P) AND DET(P)= PRODUCT OF THE PIVOTS.
CCC
DO 200 P=1,M
CCC
THE EXTENDED AND REGULAR GAUSS JORDAN ELIMINATION MOD(P) IS COMPLETED IN THE SUBROUTINE EXGAUS.

```
NULL = 0
IF(RANK(P) .LT. R - 1) NULL = 1
CALL EXGAUS(N, S, R, PRIME(P), NULL, BMOD, WP, W, WQ, DET(P))
IF(RANK(P) .LT. R) DET(P) = 0
DO 201 I = 1, R
   DO 201 J = 1, R
      ADJ(I, J, P) = WQ(I, J)
201 CONTINUE
200 CONTINUE
```

THE INTEGER PORTION OF THE REFLEXIVE INVERSE OF PP*B*QQ WILL BE STORED IN ADJF(I, J) I = 1, S AND J = 1, N. AT THIS TIME THE APPROPRIATE ROWS AND COLUMNS OF ADJF ARE ZEROED OUT.

```
RP1 = R + 1
IF(R .EQ. N .AND. N .EQ. S) GO TO 222
IF(R .EQ. N) GO TO 202
DO 203 I = RP1, N
   DO 203 J = 1, S
      ADJF(J, I) = 0. DO
203 CONTINUE
202 DO 204 I = 1, N
   DO 204 J = RP1, S
      ADJF(J, I) = 0. DO
204 CONTINUE
222 DO 300 P = 1, M
```

THE DET(B(1,1)) MOD(P) AND ADJ(B(1,1), P) ARE CONVERTED TO THEIR SYMMETRIC MULTIPLE MODULUS RESIDUE REPRESENTATION.

```
IF(DET(P) .GT. (PRIME(P)/2)) DET(P) = DET(P) - PRIME(P)
DO 300 I = 1, R
   DO 300 J = 1, R
      IF(ADJ(I, J, P) .GT. (PRIME(P)/2)) ADJ(I, J, P) = ADJ(I, J, P) - PRIME(P)
300 CONTINUE
```

THE SYMMETRIC MULTIPLE MODULUS RESIDUE REPRESENTATIONS ARE COMBINED TO FORM DET(B(1,1)) AND ADJ(B(1,1)). THESE ARE STORED IN DETF AND THE UPPER R X R LEFT HAND CORNER OF ADJF.

```
call ration(detf, det, m, prime)
do 301 i = 1, r
   do 301 j = 1, r
      det(p) = adj(i, j, p)
302 continue
   call ration(adjf(i, j), det, m, prime)
301 continue
```
DO 303 I=1,S
   DO 304 J=1,N
      WF1(J)=0.DO
   DO 304 I=1,N
      WF1(J)=ADJF(I,L)*PP(L,J)+WF1(J)
   304 CONTINUE
   DO 303 J=1,N
      ADJF(I,J)=WF1(J)
   303 CONTINUE
   DO 305 I=1,N
      DO 306 J=1,S
         WF1(J)=0.DO
      DO 306 I=1,S
         WF1(J)=QQ(J,L)*ADJF(L,I)+WF1(J)
      306 CONTINUE
   305 CONTINUE
   DO 303 J=1,N
      ADJF(J,I)=WF1(J)
   303 CONTINUE

B, PP, QQ, DETF, ADFJ AND BMOD=PP*B*QQ ARE PASSED BACK TO
THE MAIN PROGRAM.

RETURN
END

SUBROUTINE EXGAUS(N,S,K,P,NULL,A,WA,W,ADJ,PIVOT)
IMPLICIT INTEGER (A-Z)
DIMENSION A(N,S),WA(K,K),W(K),ADJ(S,S)
CCC
CCC THIS SUBROUTINE PERFORMS GAUSS - JORDAN ELIMINATION MOD(P)
CCC INVOKING EXTENDED ELIMINATION WHEN A NULL ROW IS ENCOUNTERED.
CCC THE ELIMINATION IS PERFORMED ON THE UPPER K X K LEFT HAND
CCC SUBMATRIX OF A, A(K,K).
The adjoint of this submatrix and the product of the nonzero pivots are returned from this subroutine.

Adj(A(K,K)) mod(P) is stored in the matrix Adj and the product of the nonzero pivots mod(P) is stored in PIVOT.

NULL = 1 if the rank(A(K,K)) mod(P) < K-1, Adj = 0 and PIVOT = 0.

INTER = 2
PIVOT = 1

Initialize Adj = I and store A(K,K) mod(P) in WA.

Do 1 I = 1, K
   Do 1 J = 1, K
      Adj(I,J) = 0
      If(I .EQ. J) Adj(I,J) = 1
      Wa(I,J) = Mod( Ac(I,J), P)
      If(Wa(I,J).LT.0) Wa(I,J) = Wa(I,J) + P
   1 Continue

Start to form the upper triangular matrix.

If a nonzero pivot can be found in column I perform the pivot, else go to the next column.

Do 2 I = 1, K
   If(Wa(I,I).NE.0) Go to 3
   Do 4 J = I, K
      If(Wa(J,I).NE.0) Go to 5
   4 Continue
   Go to 2
5  Do 6 L = 1, K
      W(L) = Wa(J,L)
      Wa(J,L) = Wa(I,L)
      Wa(I,L) = W(L)
      W(L) = Adj(J,L)
      Adj(J,L) = Adj(I,L)
      Adj(I,L) = W(L)
   6 Continue

Iter counts the number of row interchanges. The product of the nonzero pivots will be multiplied by (-1)**Iter or a facsimile thereof.

Iter = Iter + 1
3 Inv = Euclid(Wa(I,I), P)
PIVOT = MOD(P, PIVOT*WA(I,I), P)
DO 7 I=1,K
   WA(I,L) = MOD(INV*WA(I,L), P)
   ADJ(I,L) = MOD(INV*ADJ(I,L), P)
7 CONTINUE
IF(I.EQ.K) GO TO 20
   IP1 = I+1
   DO 8 L=IP1,K
      IF(WA(L,I).EQ.0) GO TO 8
      PIV = MOD(P-WA(L,I), P)
      DO 8 M=1,K
         IF(M.LT.I) GO TO 9
         WA(L,M) = MOD(WA(I,M)*PIV+WA(L,M), P)
      9 ADJ(L,M) = MOD(ADJ(I,M)*PIV+ADJ(L,M), P)
8 CONTINUE
2 CONTINUE

CCC
CCC WA IS NOW IN UPPER TRIANGULAR FORM.
CCC
CCC STARTING WITH THE (K,K) ELEMENT, BEGIN TO ANNIHILATE THE UPPER
CCC TRIANGULAR PORTION OF WA.
CCC
CCC IF AT STEP I THE (I,I) ELEMENT OF WA IS ZERO THEN ROW I OF
CCC WA IS NULL AND EXTENDED ELIMINATION MUST BE PERFORMED.
CCC
CCC FOR EXTENDED ELIMINATION A 1 IS PLACE IN THE (I,I) POSITION
CCC OF WA AND ALL ROWS BUT ROW I ARE ZEREO OUT IN ADJ. THEN
CCC THE ELEMENTS ABOVE WA(I,I) ARE ANNIHILATED. CONSEQUENTLY,
CCC IF ROW I OF ADJ IS NULL AT THE START OF THIS STEP, THEN
CCC RANK(A(K,K)) < K-1 AND ADJ = 0.
CCC
20 DO 10 I=1,K
   II = K-I+1
   IF(WA(II,II).NE.0) GO TO 11
CCC
CCC CHECK ROW I OF ADJ TO SEE IF IT IS NULL.
CCC
   WA(II,II) = 1
   DO 12 L=1,K
      IF(ADJ(II,L).NE.0) GO TO 13
   12 CONTINUE
CCC
CCC SET ADJ = 0
CCC
90 DO 14 L=1,K
   DO 14 M=1,K
      ADJ(L,M) = 0
   14 CONTINUE
NULL = 1
GO TO 999
CGC EXTENDED ELIMINATION IS PERFORMED HERE

13 DO 15 L=1,K
   DO 15 M=1,K
   IF(L.EQ.II) GO TO 15
   ADJ(L,M)=0
15 CONTINUE

11 IF(I.EQ.K) GO TO 99
IIM1=II-1
   DO 16 L=1,IIM1
      LIr=II-L
      IF(WA(LL,II).EQ.O) GO TO 16
      PIV=MOD( P-WA(LL,II), P)
      DO 16 M=1,K
         ADJ(LL,M)=MOD( ADJ(II,M)*PIV+ADJ(LL,M), P)
16 CONTINUE

10 CONTINUE

ELIMINATION PROCESS IS COMPLETED

PRODUCT OF NONZERO PIVOTS*(-1**ITER) MOD(P) IS FORMED.

99 PIVOT=MOD( PIVOT*(1-2*MOD(INTER,2)), P)
   IF(PIVOT.LT.0) PIVOT=PIVOT+P

ADJ(A(K,K)) = PIVOT*ADJ IS FORMED AND RESTORED IN ADJ.

DO 17 I=1,K
   DO 17 J=1,K
      ADJ(I,J)=MOD( PIVOT*ADJ(I,J), P)
17 CONTINUE

999 RETURN

SUBROUTINE RATION(SOL,X,M,PRIME)
IMPLICIT INTEGER (A-Z)
DIMENSION X(M),PRIME(M)
REAL*8 SOL
C RATION IS A SUBROUTINE THAT COMPUTES THE SYMMETRIC MIXED RADIX C
C VALUE OF AN ARRAY X WHICH CONTAINS THE SYMMETRIC RESIDUE C
C REPRESENTATION OF SOME INTEGER. PRIME(P) CONTAINS THE BASE. C
C RATION THEN CALLS A SUBROUTINE CONVER THAT CONVERTS THE SYMMETRIC C
C MIXED RADIX REPRESENTATION TO THE INTEGER VALUE IT REPRESENTS. C
C SOL=X(1)+X(2)*PRIME(1)+X(3)*PRIME(2)*PRIME(1)+...+ C
C X(M)*PRIME(M-1)*PRIME(M-2)*...*PRIME(1) C
C
DO 1 Q=2,M
  DO 2 P=Q,M
    X(P)=MOD(MOD(X(P)-X(Q-1),PRIME(P))*
    EUCLID(PRIME(Q-1),PRIME(P)),PRIME(P))
    IF(IABS(X(P)).GT.(PRIME(P)/2)) X(P)=X(P)-ISIGN(PRIME(P),X(P))
  2 CONTINUE
1 CONTINUE
CALL CONVER(SOL,X,M,PRIME)
RETURN
END

SUBROUTINE CONVER(SOL,X,M,PRIME)
IMPLICIT INTEGER (A-Z)
DIMENSION X(M),PRIME(M)
REAL*8 SOL,RADIX

CONVER CONVERTS THE SYMMETRIC MIXED RADIX REPRESENTATION TO THE
INTEGER VALUE IT REPRESENTS.

SOL=X(1)+X(2)*PRIME(1)+X(3)*PRIME(2)*PRIME(1)+ . . . +(M)*PRIME(M-1)*PRIME(M-2)* . . . *PRIME(1)

THESE COMPUTATIONS ARE DONE IN DOUBLE PRECISION FLOATING POINT
ARITHMETIC.

SOL=X(1)
DO 1 Q=2,M
  RADIX=X(M-Q+2)
  DO 2 P=Q,M
    RADIX=RADIX*PRIME(M-P+1)
  2 CONTINUE
SOL=SOL+RADIX
1 CONTINUE
RETURN
END

SUBROUTINE JHNEW(N,S,R,M,PRIME,RPLUS,RMIN,RADJP,RADJM,B,BDADJ)
IMPLICIT INTEGER (A-Z)
DIMENSION B(N,S),PRIME(20),RPLUS(R),RMIN(R),RADJP(R),
*RADJM(R),BDADJ(R,R)
REAL*8 RPLUS,RMIN,RADJP,RADJM,BDADJ,BDDET,BDD,BDMAX,BDMIN

THIS SUBROUTINE COMPUTES JOHNSON AND NEWMAN'S BOUND FOR THE
DETERMINANT OF THE R X R NONSINGULAR MINOR OF B, B(R,R), AND
ITS' COFACTORS, B(K,L).

IN MANY CASES THIS BOUND WILL BE MUCH SMALLER THAN THE ORIGINAL
MODIFIED SCHINZEL'S BOUND. CONSEQUENTLY, FEWER PRIMES IN
THE MULTIPLE MODULUS BASE WILL BE SUFFICIENT TO SOLVE THE GGC PROBLEM.

THE R X R NONSINGULAR MINOR IS STORED IN THE UPPER R X R LEFT CORNER OF B.

THE ARRAY RPLUS(N) CONTAINS THE SUM OF THE POSITIVE B(I,J) ELEMENTS ACROSS EACH ROW(I), I=1,N.

THE ARRAY RMIN(N) CONTAINS THE SUM OF THE ABSOLUTE VALUE OF THE NEGATIVE B(I,J) ELEMENTS ACROSS EACH ROW(I), I=1,N.

ABS(DET(B(R,R)))<=MAX(RPLUS(1),RMIN(1))*...*MAX(RPLUS(R),RMIN(R))*

-MIN(RPLUS(1),RMIN(1))*...*MIN(RPLUS(R),RMIN(R))

ABS(DET(B(K,L)))<= MAX - MIN AS FORMED ABOVE ONLY REMOVING THE (RPLUS(K),RMIN(K)) PAIR AND THE (I,L) ELEMENTS FROM THE OTHER (RPLUS(I),RMIN(I)) PAIR.

COMPUTE BOUND FOR DET(B(R,R)) AND STORE IN BDDET.

DO 1 I=1,R
   RPLUS(I)=0.DO
   RMIN(I) =0.DO
   DO 2 J=1,R
      RPLUS(I)=RPLUS(I)+AMAXO(0,B(I,J))
      RMIN(I) =RMIN(I) -AMIN0(0,B(I,J))
   2 CONTINUE
   BDMAX=BDMAX*DMAX1(RPLUS(I),RMIN(I),1.DO)
IF(BDMIN.EQ.O.DO) GO TO 1
   BDMIN=BDMIN*DMIN1(RPLUS(I),RMIN(I))
1 CONTINUE
   BDDET=BDMAX-BDMIN

COMPUTE BOUNDS FOR DET(B(K,L)) AND STORE THESE IN BADJ(K,L)

DO 3 K=1,R AND L=1,R.
3 DO 2 I=1,R
   BDMAX=1.DO
   BDMIN=1.DO
   DO 5 J=1,R
      IF(I.EQ.K) GO TO 5
      RADJP(I)=RPLUS(I)-AMAXO(0,B(I,L))
      RADJM(I)=RMIN(I) +AMINO(0,B(I,L))
      BDMAX=BDMAX*DMAX1(RADJP(I),RADJM(I),1.DO)
      IF(BDMIN.EQ.0.DO) GO TO 5
   5 CONTINUE
BDMIN = BDMIN * DMIN1(RADJP(I), RADJM(I))

CONTINUE

BDADJ(K,L) = BDMAX - BDMIN

CONTINUE

CCC

CCC FIND MAX(BDDET, MAX(BADJ(K,L)) = BDD

CCC

K, L

BDD = 1.0
DO 6 I = 1, R
    DO 6 J = 1, R
        BDD = DMAX1(BDD, BDADJ(I, J))
    6 CONTINUE

CCC

CCC MULTIPLE BDD BY 2 AND CALL SUBROUTINE SELECT TO FIND THE
CCC MULTIPLE MODULUS BASE NEEDED TO INVERT B(R,R).
CCC
CCC NOTE: THIS BASE IS A SUBSET OF THE BASE FOUND VIA
CCC SCHINZEL'S BOUND AT THE BEGINNING OF THIS
CCC PROBLEM.

BDD = 2 * DMAX1(BDD, BDDET)
CALL SELECT(BDD, M, PRIME)
RETURN

END

FUNCTION EUCLID(K, M)
IMPLICIT INTEGER (A-Z)

FUNCTION EUCLID IS THE STANDARD EXTENDED EUCLIDEAN ALGORITHM.
EUCLID COMPUTES THE INVERSE OF K MOD(M).

U1 = 0
U2 = M
V1 = 1
V2 = K

1 IF(V2 .EQ. 1) GO TO 99
    IF(V2 .EQ. 0) GO TO 98
    Q = U2 / V2
    T1 = U1 - Q * V1
    T2 = U2 - Q * V2
    U1 = V1
    U2 = V2
    V1 = T1
    V2 = T2
    GO TO 1

98 EUCLID = 0
RETURN

99 IF(V1 .LT. 0) V1 = M + V1
EUCLID = V1
RETURN

END

/G0.SYSIN DD *
10.2 The Bordering Method FORTRAN Program

//S1 EXEC FORTG
IMPLICIT INTEGER (A-Z)
DIMENSION W1(8),W2(38),W3(38),WF1(8),WF2(8),WF3(8),WF4(8),
*WMAT1(8,8),WMAT2(8,8),WMAT3(8,38),WMAT4(8,38),WMAT5(8,38),
*PP(8,8),A(8,8),ADJ(8,8,38),ADJF(8,8),PRIME(38)
REAL*8 WF1,WF2,WF3,WF4,DET,ADJF

THIS PROGRAM COMPUTES A REFLEXIVE INVERSE OF AN (N X N) SYMMETRIC
POSITIVE SEMI-DEFINITE MATRIX A'A USING A BORDERING METHOD
BASED ON MULTIPLE MODULUS RESIDUE ARITHMETIC.
THIS PARTICULAR "MAIN" PROGRAM IS SETUP TO COMPUTE A REFLEXIVE
GENERALIZED INVERSE FOR THE A'A MATRIX OF EXAMPLE 7.3.1.
THE ARRAY PRIME CONTAINS THE POSSIBLE PRIME MODULI THAT WILL
BE USED.
DATA PRIME/ 45233,45247,45259,45263,45281,45289
1,45293,45307,45317,45319,45329,45337,45341,45343,45361
2,45377,45389,45403,45413,45427,8117,8123,8147,8161,8171,8179
3,8191,8209,8219,8231,47,53,59,61,67,71,73,83/

INPUT (N X N) SYMMETRIC POSITIVE SEMI-DEFINITE MATRIX A'A.
N=8
DO 1 I=1,N
1 READ(5,100) (A(I,J),J=1,N)
WRITE(6,300)
DO 2 I=1,N
2 WRITE(6,200) (A(I,J),J=1,N)

-COMPUTE THE INITIAL PRODUCT MODULUS BOUND.
CALL SCHINZ(N,N,M,PRIME,A,WF1,WF2)
WRITE(6,400) M

-COMPUTE THE REFLEXIVE INVERSE OF A'A.
CALL BORDER(N,M,PRIME,W1,W2,W3,A,WMAT1,PP,WMAT3,WMAT4,WMAT5,
*WMAT2,ADJ,ADJF,WF1,WF2,WF3,WF4,DET,RANK)

THE FORMATS USED HERE FOR OUTPUT HAVE BEEN SPECIFIED WITH
EXAMPLE 7.3.1 IN MIND.
WRITE(6,500) RANK
WRITE(6,901) M
WRITE(6,904) (PP(1,J),J=1,N)
4 WRITE(6,905) (PP(I,J),J=1,N)
WRITE(6,902) DET
DO 3 I=1,N
3 WRITE(6,903) (ADJF(I,J),J=1,N)
WRITE(6,906)
DO 5 I=1,N
DO 6 J=1,N
ADJF(I,J)=ADJF(I,J)/DET
6 CONTINUE
WRITE(6,907) (ADJF(I,J),J=1,N)
5 CONTINUE
100 FORMAT(8I4)
200 FORMAT(' ',13X,8I7)
300 FORMAT('1',////,13X,38H THE INPUT MATRIX A' A IS AS FOLLOWS:/)
400 FORMAT(' ',13X,'THE NUMBER OF PRIME MODULI USED IN THE ',
* 'BORDERING ALGORITHM = ',I3)
500 FORMAT(' ',13X,'RANK(A) = ',I2)
901 FORMAT(' ',13X,3X,\',/13X,1X,12H A' A = ( 1 /,F12.1,') *'
902 FORMAT(' ',13X,3X,\_',/13X,1X,12H A = ( 1 /,F12.1,') *'
903 FORMAT(' ',13X,\',3X,F10.1,1X,F4.1,1X,F11.1,1X,3(F4.1,1X))
904 FORMAT(' ',13X,\',P = ',10I3)
905 FORMAT(' ',13X,3X,10I3)
906 FORMAT('0',13X,4X,')')
907 FORMAT(' ',13X,\',5X,3(F10.7,1X),F4.1,1X,F11.8,1X,3(F4.1,1X))
STOP
END

SUBROUTINE BORDER(N,M,PRIME,W,NULL,AKK,A,WA,PP,B,DET,V,WADJ,
* ADJ,ADJF,WF1,WF2,WF3,WF4,DT,RANK)
IMPLICIT INTEGER (A-Z)
DIMENSION PRIME(38),W(N),NULL(M),AKK(M),A(N,N),WA(N,N),PP(N,N),
*WADJ(N,N),B(N,N),DET(N,N),V(N,N),ADJ(N,N,M),ADJF(N,N),
*WF1(N),WF2(N),WF3(N),WF4(N)
REAL*8 WF1,WF2,WF3,WF4,ADJF,DT
CCC THIS SUBROUTINE WILL SIMULTANEOUSLY SEARCH OUT A LARGEST
CCC NONSINGULAR MINOR OF THE (N X N) SYMMETRIC POSITIVE SEMI -
CCC DEFINITE MATRIX A AND INVERT THIS MINOR.
CCC THE BORDERING METHOD IS USED TO INVERT THE MINOR.
CCC
IF

\[
A(K) = \begin{bmatrix}
A(K-1) & B(K) \\
B(K)' & A(K,K)
\end{bmatrix}
\]

IS A (K X K) SYMMETRIC NONSINGULAR MINOR OF A THEN

\[
\text{ADJ}(A(K)) = \text{ADJ}(K) = \text{ADJ}(K-1)
\]

\[
\begin{bmatrix}
(\text{ADJ}(K-1) + \text{ADJ}(K-1) \cdot B(K) \cdot B(K)' \cdot \text{ADJ}(K-1) / \text{DET}(K-1)) & -\text{ADJ}(K-1) \cdot B(K) \\
-B(K)' \cdot \text{ADJ}(K-1) & \text{DET}(K-1)
\end{bmatrix}
\]

AND \( \text{DET}(A(K)) = \text{DET}(K) = \text{DET}(K-1) \cdot A(K,K) - B(K)' \cdot \text{ADJ}(K-1) \cdot B(K) \)

CONTINUE

AT EACH STEP, K, THE MATRIX A IS SEARCHED TO FIND A PARTIAL ROW AND COLUMN OF A TO ADJOIN TO A(K-1) SUCH THAT A(K) IS A (K X K) NONSINGULAR MINOR. THIS IS DETERMINED BY FIRST COMPUTING \( \text{DET}(K) \) AND CHECKING TO SEE IF IT IS NONZERO. A PERMUTATION MATRIX PP IS FORMED TO KEEP TRACK OF THE ROW AND COLUMN INTERCHANGES IN A.

IF \( \text{RANK} = \text{RANK}(A) \) THEN

\[
\text{PP} \cdot A \cdot \text{PP}' = \begin{bmatrix}
\text{A}(\text{RANK}) \\
* \\
* \\
*
\end{bmatrix}
\]

AND THE REFLEXIVE INVERSE OR A IS CONSTRUCTED AS

\[
A = \begin{bmatrix}
1 & \text{ADJ}(\text{RANK}) & 0 \\
\text{DET}(\text{RANK}) & \text{ADJ}(\text{RANK}) & 0 \\
* & 0 & 0
\end{bmatrix}
\]

THE BORDERING PROCESS IS DONE USING MULTIPLE MODULUS RESIDUE ARITHMETIC.

IF THE PRODUCT MODULUS BOUND IS CHOSEN CORRECTLY THE \( \text{DET}(K) = 0 \) IF AND ONLY IF \( \text{DET}(K) \mod (P) = 0 \) FOR ALL PRIMES, P, IN THE BASE.
CONTINUE

A FEW NOTES:

--ONE PORTION OF ADJ(K) INVOLVES INVERTING DET(K-1) MOD(P). CONSEQUENTLY, DET(K-1) MOD(P) MUST BE NONZERO TO DO THIS STEP USING THE BORDERING FORMULA.

--IF DET(K-1) MOD(P)=0 THIS DOES NOT NECESSARILY MEAN ADJ(K) MOD(P) IS NULL. IN THIS CASE WE CAN'T USE THE BORDERING FORMULA TO FIND ADJ(K) MOD(P).

EXTENDED GAUSS JORDAN ELIMINATION MOD(P) IS USED ON A(K) MOD(P) TO FIND ADJ(K) MOD(P).

--IF ADJ(K) MOD(P) IS NULL THEN ADJ(K+1) MOD(P),..., ADJ(RANK) MOD(P) WILL ALSO BE NULL AND DET(K) MOD(P)=DET(K+1) MOD(P)=...=DET(RANK) MOD(P)=0.

AT STEP K IF NO PARTIAL ROW OR COLUMN CAN BE LOCATED TO ADJOIN TO A(K-1) SUCH THAT DET(K) MOD(P) IS NONZERO OF AT LEAST ONE PRIME, P, THEN THE PROCESS STOPS AND RANK=RANK(A)=K-1.

INITIALIZE PP=I

DO 999 I=1,N
DO 999 J=1,N
PP(I,J)=0
IF(I.EQ.J) PP(I,J)=1
999 CONTINUE

FIND A NONZERO DIAGONAL ELEMENT OF A, THIS WILL BE A(1).

DO 1 I=1,N
IF(A(I,I).NE.0) GO TO 3
1 CONTINUE

IF THE (1,1) ELEMENT OF A IS THE FIRST NONZERO DIAGONAL ELEMENT INTERCHANGE ROW 1 AND I IN BOTH A AND PP, THEN INTERCHANGE COLUMNS 1 AND I IN A.

3 IF(I.EQ.1) GO TO 4
DO 2 I=1,N
   W(L)=PP(1,L)
   PP(1,L)=PP(I,L)
   PP(I,L)=W(L)
   W(L)=A(1,L)
   A(1,L)=A(I,L)
   A(I,L)=W(L)
2 CONTINUE
DO 22 L=1,N
   W(L)=A(L,1)
   A(L,1)=A(L,I)
   A(L,I)=W(L)
22 CONTINUE
CCC DET(1) = A(1) MOD(P)
CCC ADJ(1) = 1
CCC
   4 DO 5 P=1,M
      DET(1,P)=MOD( A(1,1), PRIME(P))
      ADJ(1,1,P)=1
      NULL(P)=0
5 CONTINUE
CCC STEP K STARKS HERE
CCC
   DO 100 K=2,N
CCC
      KM1=K-1
      OK=0
CCC AT STEP K NULL(P)=1 IF ADJ(K-2) MOD(P) IS NULL. IF NULL(P)=1
CCC THEN ADJ(K) MOD(P) IS ZEROED OUT AND THE PROCESS PROCEEDS TO
CCC THE NEXT PRIME.
CCC IF NULL(P)=0 THEN A CHECK IS DONE TO SEE IF ADJ(K-1) MOD(P) IS
CCC NULL. IF ADJ(K-1) MOD(P) IS NULL NULL(P) IS SET TO 1.
CCC
   DO 9 P=1,M
      IF(NULL(P).EQ.1) GO TO 9
      DO 8 I=1,KM1
         DO 8 J=1,KM1
            IF(ADJ(I,J,P).NE.0) GO TO 9
      8 CONTINUE
      NULL(P)=1
9 CONTINUE
CCC THE SEARCH FOR A PARTIAL ROW AND COLUMN TO ADJOIN TO A(K-1)
CCC STARTS HERE. R KEEPS TRACKK OF WHICH PARTIAL ROWS AND COLUMNS
CCC HAVE BEEN TRIED.
DO 200 R=K,N

DET(K) MOD(P) IS COMPUTED FOR EACH PRIME.

DO 6 P=1,N

IF(NULL(P).EQ.1) GO TO 6

AKK(P)=MOD( A(K,K), PRIME(P))

IF(AKK(P).LT.0) AKK(P)=AKK(P)+PRIME(P)

DO 7 I=1,KM1

B(I,P)=MOD( A(I,K), PRIME(P))

IF(B(I,P).LT.0) B(I,P)=B(I,P)+PRIME(P)

7 CONTINUE

CONSTRUCT V=ADJ(K-1)*B(K) MOD(P)

DO 10 I=1,KM1

V(I,P)=0

DO 10 J=1,KM1

V(I,P)=MOD( ADJ(I,J,P)*B(J,P)+V(I,P), PRIME(P))

10 CONTINUE

SUM=0

DET(K)=DET(K-1)*AKK(K)-B(K)'*V MOD(P)

DO 11 I=1,KM1

SUM=MOD( B(I,P)*V(I,P)+SUM, PRIME(P))

11 CONTINUE

IF(DET(K,P).NE.0) OK=1

6 CONTINUE

IF(OK.EQ.1) GO TO 12

IF(R.EQ.N) GO TO 200

IF R DOES NOT EQUAL N THEN ROWS R+1 AND K OF BOTH A AND PP ARE
INTERCHANGED AND COLUMNS R+1 AND K OF A ARE INTERCHANGED.
THEN STEP K IS STARTED OVER.

IF OK=0 AFTER DET(K) MOD(P) HAS BEEN COMPUTED FOR ALL THE PRIMES
IN THE BASE, THEN A DIFFERENT PARTIAL ROW AND COLUMN IS
ADJOINED TO A(K-1) AND THE PROCESS IS REPEATED.

IF DET(K) MOD(P) IS NONZERO FOR AT LEAST ONE PRIME, P, THEN THE
FLAG "OK" IS SET TO 1.

IF R=N THEN THERE IS NO (K X K) NONSINGULAR MINOR THAT CAN
BE FOUND. RANK(A)=K-1.
DO 13 I=1,N
   W(I)=PP(R+1,I)
   PP(R+1,I)=PP(K,I)
   PP(K,I)=W(I)
13 CONTINUE
DO 23 I=1,N
   W(I)=A(I,R+1)
   A(I,R+1)=A(I,K)
   A(I,K)=W(I)
23 CONTINUE
RANK=K-1
GO TO 90

IF RANK(A)>K-1 THEN ADJ(K) MOD(P) IS FORMED FOR EACH PRIME.

DO 300 P=1,M
   IF(NULL(P).EQ.0.AND.DETCK-1,P).NE.0) GO TO 14
   IF(NULL(P).EQ.0) GO TO 15
   IF NULL(P)=1 THEN ADJ(K) MOD(P) IS NULL.
   DO 16 I=1,K
      DO 16 J=1,K
         ADJ(I,J,P)=0
16 CONTINUE
   DET(K,P)=0
   GO TO 300

IF DET(K-1) MOD(P)=0 BUT ADJ(K-1) MOD(P) IS NONNULL THEN
   EXTENDED GAUSS JORDAN ELIMINATION IS USED ON A(K) MOD(P) TO
   FORM ADJ(K) MOD(P). THE ELIMINATION IS PERFORMED IN THE
   SUBROUTINE EXGAUS.
   CALL EXGAUS(N,K,PRIME(P),NULL(P),A,WA,W,ADJ)
   DO 20 I=1,K
      DO 20 J=1,K
         ADJ(I,J,P)=ADJ(I,J)
20 CONTINUE
   GO TO 300

BORDERING FORMULA IS USED HERE TO FORM ADJ(K) MOD(P).
INV=EUCLID(DET(K-1,P),PRIME(P))
DO 17 I=1,KM1
DO 17 J=1,KM1
ADJ(I,J,P)=MOD(MOD(DET(K,P)*ADJ(I,J,P), PRIME(P)) +
* MOD(V(I,P)*V(J,P), PRIME(P)), PRIME(P))
ADJ(I,J,P)=MOD(INV*ADJ(I,J,P), PRIME(P))
17 CONTINUE
ADJ(K,K,P)=DET(K-1,P)
DO 18 I=1,KM1
ADJ(I,K,P)=MOD(PRIME(P)-V(I,P), PRIME(P))
ADJ(K,I,P)=ADJ(I,K,P)
18 CONTINUE
300 CONTINUE
100 CONTINUE
RANK=N
CCC
CCC AT THIS POINT THE RANK(A) HAS BEEN DETERMINED AND THE
CCC (RANK X RANK) UPPER LEFT HAND CORNER SUBMATRIX OF A CONTAINS
CCC THE SYMMETRIC NO SINGULAR MINOR FOR WHICH ADJ MOD(P) AND
CCC DET MOD(P) HAS BEEN COMPUTED.
CCC
CCC THIS MINOR IS SENT INTO THE SUBROUTINE SCHINZ WHERE A SUBSET
CCC OF THE ORIGINAL BASE OF PRIME MODULI IS FOUND BY COMPUTING
CCC A PRODUCT MODULUS BOUND BASED OF THIS MINOR.
CCC
CCC THE SOLUTIONS FOR THE PRIME MODULI IN THIS NEW BASE WILL BE
CCC THE SOLUTIONS COMBINED TO FORM TO REFLEXIVE INVERSE OF A.
CCC
CCC NOTE: THIS SUBSET IS NOT NECESSARILY A PROPER SUBSET.
CCC
90 CALL SCHINZ(N,RANK,M,PRIME,A,WF1,WF2)
CCC
CCC DET MOD(P) AND ADJ MOD(P) ARE CONVERTED TO THEIR
CCC SYMMETRIC MULTIPLE MODULUS RESIDUE REPRESENTATION.
CCC
DO 111 P=1,M
NULL(P)=DET(RANK,P)
IF(NULL(P).GT.(PRIME(P)/2)) NULL(P)=NULL(P)-PRIME(P)
DO 111 I=1,RANK
DO 111 J=1,RANK
IF(ADJ(I,J,P).GT.(PRIME(P)/2)) ADJ(I,J,P)=ADJ(I,J,P)-PRIME(P)
111 CONTINUE
CALL RATION(DT,NULL,M,PRIME)
CCC
CCC THE SYMMETRIC MULTIPLE MODULUS RESIDUE REPRESENTATIONS ARE
CCC COMBINED TO FORM DET(RANK) AND ADJ(RANK). THESE ARE STORED
CCC IN DT AND THE UPPER (RANK X RANK) LEFT HAND CORNER OF ADJF.
DO 112 I=1,RANK
DO 112 J=1,RANK
   DO 113 P=1,M
      NULL(P)=ADJ(I,J,P)
   CONTINUE
   CALL RATION(ADJF(I,J),NULL,M,PRIME)
   CONTINUE
IF(RANK.EQ.N) GO TO 222
CCC
CCC THE APPROPRIATE ROW AND COLUMNS OF ADJF ARE ZEROED OUT.
CCC
   RP1=RANK+1
   DO 114 I=1,N
      DO 114 J=RP1,N
         ADJF(I,J)=0.D0
         ADJF(J,I)=0.D0
   CONTINUE
222 DO 211 I=1,N
   DO 212 J=1,N
      WF1(J)=0.D0
   CONTINUE
   DO 212 L=1,N
      WF1(J)=ADJF(I,L)*PP(L,J)+WF1(J)
   CONTINUE
   DO 211 J=1,N
      ADJF(I,J)=WF1(J)
   CONTINUE
   CCC
   CCC THE FOLLOWING PRODUCT IS FORMED
   CCC
   PP'*ADJF*PP = PP'*
   PP
   PP
   = DET(A(RANK)) * A
   CCC
DO 311 I=1,N
   DO 312 J=1,N
      WF1(J)=0.D0
   CONTINUE
   DO 312 L=1,N
      WF1(J)=PP(L,J)*ADJF(L,I)+WF1(J)
   CONTINUE
   DO 311 J=1,N
      ADJF(J,I)=WF1(J)
   CONTINUE
   CCC
   CCC PP, ADJF, DT=DET(A(RANK)), RANK AND A=PP*A*PP' ARE PASSED
   CCC BACK TO THE MAIN PROGRAM
   CCC
SUBROUTINE SCHINZ(D,N,M,PRIME,B,RPLUS,RMIN)
IMPLICIT INTEGER (A-Z)
DIMENSION B(D,D),PRIME(38),RPLUS(N),RMIN(N)
REAL*R8 RPLUS,RMIN,BDD

THIS SUBROUTINE COMPUTES A MODIFIED SCHINZEL'S BOUND FOR THE DETERMINANT OF ANY MINOR OF THE N X N MATRIX B.

THE ARRAY RPLUS(N) CONTAINS THE SUM OF THE POSITIVE B(I,J) ELEMENTS ACROSS EACH ROW(I), I=1,N.

THE ARRAY RMIN(N) CONTAINS THE SUM OF THE ABSOLUTE VALUE OF THE NEGATIVE B(I,J) ELEMENTS ACROSS EACH ROW(I), I=1,N.

BDD=MAX(RPLUS(1),RMIN(1))*MAX(RPLUS(2),RMIN(2))* ... MAX(RPLUS(N),RMIN(N))

>= DETERMINANT OF ANY MINOR

IT IS ASSUMED THAT B HAS NO NULL ROWS, IF A ROW OF B IS NULL MAX(RPLUS(I),RMIN(I))=1

BDD=O.DO

RPLUS(N) AND RMIN(N) ARE COMPUTED HERE

DO 1 I=1,N
  RPLUS(I)=O.DO
  RMIN(I)=O.DO
  DO 2 J=1,N
    RPLUS(I)=RPLUS(I)+AMAX0(0,B(I,J))
    RMIN(I) =RMIN(I) -AMIN0(0,B(I,J))
  2 CONTINUE

BDD IS FORMED HERE.

ACTUALLY THE LOG(BDD) IS COMPUTED SO THAT THE FLOATING-POINT EXPONENT WILL NOT OVERFLOW.

BDD=BDD+DLOG(DMAX1(RPLUS(I),RMIN(I),1.DO))

1 CONTINUE

BDD IS MULTIPLIED BY 2 SO THAT BDD>= 2(DET(ANY MINOR))
THE SUBROUTINE SELECT IS THEN CALLED TO CHOOSE THE NUMBER OF
PRIMES (M) NEEDED FROM THE TABLE OF PRIMES TO GENERATE A LARGEST
NONSINGULAR MINOR USING MULTIPLE MODULUS RESIDU ARITHMETIC.

SINCE WE ARE COMPUTING THE LOG(BDD) RATHER THAN MULTIPLYING
BY 2 WE ADD LOG(2.0) TO LOG(BDD) TO COMPUTE LOG(2*BDD).

BDD=BDD+DLOG(2.DO)
CALL SELECT(BDD,M,PRIME)
RETURN
END

SUBROUTINE SELECT(BDD,M,PRIME)
IMPLICIT INTEGER (A-Z)
DIMENSION PRIME(38)
REAL*8 BDD,PPROD,PP

DO 1 1=1,38
   PP=PRIME(I)
   PPROD=PPROD+DLOG(PP)
   IF(PPROD.GE.BDD) GO TO 2
1 CONTINUE
   M=I
2 RETURN
END

SUBROUTINE EXGAUS(N,K,P,NULL,A,W,A,W,ADJ)
IMPLICIT INTEGER (A-Z)
DIMENSION A(N,N),W(A,K),W(K),ADJ(N,N)

THIS SUBROUTINE PERFORMS GAUSS-JORDAN ELIMINATION MOD(P) INVOKING EXTENDED ELIMINATION WHEN A NULL ROW IS ENCOUNTERED.
THE ELIMINATION IS PERFORMED ON THE UPPER K X K LEFT HAND SUBMATRIX OF A, A(K,K).

THE ADJOINT OF THIS SUBMATRIX AND THE PRODUCT OF THE NONZERO PIVOTS ARE RETURNED FROM THIS SUBROUTINE.

ADJ(A(K,K)) MOD(P) IS STORED IN THE MATRIX ADJ AND THE PRODUCT OF THE NOZERO PIVOTS MOD(P) IS STORED IN PIVOT.

NULL=1 IF THE RANK(A(K,K)) MOD(P) < K-1, ADJ=0 AND PIV0T=0.

INTER=2
PIV0T=1

INITIALIZE ADJ=I AND STORE A(K,K) MOD(P) IN WA.

DO 1 I=1,K
  DO 1 J=1,K
    ADJ(I,J)=0
    IF(I.EQ.J) ADJ(I,J)=1
    WA(I,J)=MOD(A(I,J), P)
    IF(WA(I,J).LT.0) WA(I,J)=WA(I,J)+P
  1 CONTINUE

START TO FORM THE UPPER TRIANGULAR MATRIX.

IF A NONZERO PIVOT CAN BE FOUND IN COLUMN I PERFORM THE PIVOT,
ELSE GO TO THE NEXT COLUMN.

DO 2 I=1,K
  IF(WA(I,I).NE.0) GO TO 3
  DO 4 J=1,K
    IF(WA(J,I).NE.0) GO TO 5
  4 CONTINUE
  GO TO 2
  5 DO 6 L=1,K
    W(L)=WA(J,L)
    WA(J,L)=WA(I,L)
    WA(I,L)=W(L)
    W(L)=ADJ(J,L)
    ADJ(J,L)=ADJ(I,L)
    ADJ(I,L)=W(L)
  6 CONTINUE

ITER COUNTS THE NUMBER OF ROW INTERCHANGES. THE PRODUCT OF THE NOZERO PIVOTS WILL BE MULTIPLIED BY (-1**INTER) OR A FACSIMILE THEREOF.
INTER = INTER + 1

INV = EUCLID(WA(I,I), P)
Pivot = MOD( Pivot * WA(I,I), P)
DO 7 L = 1, K
   WA(I,L) = MOD( INV * WA(I,L), P)
   ADJ(I,L) = MOD( INV * ADJ(I,L), P)
7 CONTINUE

IF (I.EQ.K) GO TO 20
IP1 = I + 1
DO 8 L = IP1, K
   IF (WA(L,I).EQ.0) GO TO 8
   Pivot = MOD( P - WA(L,I), P)
   DO 8 M = 1, K
      IF (M.LT.I) GO TO 9
      WA(L,M) = MOD( WA(L,M) * Pivot - WA(L,M), P)
9   ADJ(L,M) = MOD( ADJ(I,M) * Pivot + ADJ(L,M), P)
8 CONTINUE
2 CONTINUE

CCC
CCC WA IS NOW IN UPPER TRIANGULAR FORM.
CCC
CCC STARTING WITH THE (K,K) ELEMENT, BEGIN TO ANNIHILATE THE UPPER
CCC TRIANGULAR PORTION OF WA.
CCC
CCC IF AT STEP I THE (1,1) ELEMENT OF WA IS ZERO THEN ROW I OF
CCC WA IS NULL AND EXTENDED ELIMINATION MUST BE PERFORMED.
CCC
CCC FOR EXTENDED ELIMINATION A 1 IS PLACE IN THE (1,1) POSITION
CCC OF WA AND ALL ROWS BUT ROW I ARE ZEROED OUT IN ADJ. THEN
CCC THE ELEMENTS ABOVE WA(I,I) ARE ANNIHILATED. CONSEQUENTLY,
CCC IF ROW I OF ADJ IS NULL AT THE START OF THIS STEP, THEN
CCC RANK(A(K,K)) < K-1 AND ADJ = 0.
CCC
20 DO 10 I = 1, K
   II = K - I + 1
   IF (WA(II,II).NE.0) GO TO 11
CCC
CCC CHECK ROW I OF ADJ TO SEE IF IT IS NULL.
CCC
   WA(II,II) = 1
   DO 12 L = 1, K
      IF (ADJ(II,L).NE.0) GO TO 13
12 CONTINUE
CCC
CCC SET ADJ = 0
CCC
   DO 14 L = 1, K
      DO 14 M = 1, K
         ADJ(L,M) = 0
14 CONTINUE
NULL=1
GO TO 999

CGC
CGC EXTENDED ELIMINATION IS PERFORMED HERE
CGC
13       DO 15 L=1,K
         DO 15 M=1,K
            IF(L.EQ.II) GO TO 15
            ADJ(L,M)=0
15      CONTINUE
11      IF(I.EQ.K) GO TO 99
       IIM1=II-1
       DO 16 L=1,IIM1
          LL=II-L
          IF(WA(LL,II).EQ.0) GO TO 16
          PIV=MOD( P-WA(LL,II), P)
          DO 16 M=1,K
             ADJ(LL,M)=MOD( ADJ(II,M)*PIV+ADJ(LL,M), P)
16    CONTINUE
10   CONTINUE

CGC
CGC ELIMINATION PROCESS IS COMPLETED
CGC
CGC PRODUCT OF NONZERO PIVOTS*(-1**ITER) MOD(P) IS FORMED.
CGC
99      PIVOT=MOD( PIVOT*(1-2*MOD(INTER,2)), P)
       IF(PIVOT.LT.0) PIVOT=PIVOT+P

CGC
CGC ADJ(A(K,K)) = PIVOT*ADJ IS FORMED AND RESTORED IN ADJ.
CGC
DO 17 1=1,K
DO 17 J=1,K
   ADJ(I,J)=MOD( PIVOT*ADJ(I,J), P)
17 CONTINUE
999 RETURN

END

SUBROUTINE RATION(SOL,X,M,PRIME)
IMPLICIT INTEGER (A-Z)
DIMENSION X(M),PRIME(M)
REAL*8 SOL

CGC
CGC RATION IS A SUBROUTINE THAT COMPUTES THE SYMMETRIC MIXED RADIX C
CGC VALUE OF AN ARRAY X WHICH CONTAINS THE SYMMETRIC RESIDUE C
CGC REPRESENTATION OF SOME INTEGER. PRIME(P) CONTAINS THE BASE. C
CGC
CGC RATION THEN CALLS A SUBROUTINE CONVER THAT CONVERTS THE SYMMETRIC C
CGC MIXED RADIX REPRESENTATION TO THE INTEGER VALUE IT REPRESENTS. C
CGC
CGC
C SOL=X(1)+X(2)*PRIME(1)+X(3)*PRIME(2)*PRIME(1)+...+C
C X(M)*PRIME(M-1)*PRIME(M-2)*...*PRIME(1) C
C C

DO 1 Q=2,M
  DO 2 P=Q,M
    X(P)= MOD(MOD(X(P)-X(Q-1),PRIME(P)) *,
    EUCLID(PRIME(Q-1),PRIME(P)),PRIME(P))
    IF(IABS(X(P)).GT.(PRIME(P)/2)) X(P)=X(P)-ISIGN(PRIME(P),X(P))
  2 CONTINUE
  1 CONTINUE
CALL CONVER(SOL,X,M,PRIME)
RETURN
END

SUBROUTINE CONVER(SOL,X,M,PRIME)
IMPLICIT INTEGER (A-Z)
DIMENSION X(M),PRIME(M)
REAL*8 SOL,RADIX

SOL=X(1)
DO 1 Q=2,M
  RADIX=X(M-Q+2)
  DO 2 P=Q,M
    RADIX=RADIX*PRIME(M-P+1)
  2 CONTINUE
  SOL=SOL+RADIX
  1 CONTINUE
RETURN
END

FUNCTION EUCLID(K,M)
IMPLICIT INTEGER (A-Z)

FUNCTION EUCLID IS THE STANDARD EXTENDED EUCLIDEAN ALGORITHM.
EUCLID COMPUTES THE INVERSE OF K MOD(M).

FUNCTION EUCLID(K,M)
U1=0
U2=M
V1=1
V2=K
1 IF(V2.EQ.1) GO TO 99
   IF(V2.EQ.0) GO TO 98
   Q=U2/V2
   T1=U1-Q*V1
   T2=U2-Q*V2
   U1=V1
   U2=V2
   V1=T1
   V2=T2
   GO TO 1
98 EUCLID=0
   RETURN
99 IF(V1.LT.0) V1=M+V1
   EUCLID=V1
   RETURN
END
//GO.SYSIN DD *
646 454 94 -38 126 202 18 146
454 409 91 106 112 55 54 107
94 91 301 -2 136 -47 -138 89
-38 106 -2 202 14 -134 78 -10
126 112 136 14 86 -4 -46 56
202 55 -47 -134 -4 157 -6 29
18 54 -138 78 -46 -6 102 -30
146 107 89 -10 56 29 -30 49
10.3 The Moore–Penrose Inversion Method
FORTRAN Program

//S1 EXEC FORTG
IMPLICIT INTEGER (A-Z)
DIMENSION A(5,8),WMAT1(5,5),WMATF(5,5),WMAT2(5,5),
* AA(5,5),WMAT3(5,5,38),WORK1(38),WORK2(38),ADAG(8,5),
* PRIME(38)
REAL*8 ADAG,WMATF,QK

C THIS PROGRAM COMPUTES THE EXACT MOORE PENROSE INVERSE OF AN
C (N X S) MATRIX A.
C
C THIS PARTICULAR "MAIN" PROGRAM IS SET UP TO COMPUTE AN EXACT
C MOORE-PENROSE INVERSE FOR THE MATRIX A IN EXAMPLE 7.3.1.
C
C THE ARRAY PRIME CONTAINS THE POSSIBLE PRIME MODULI THAT WILL
C BE USED.
C
DATA PRIME/ 45233,45247,45259,45263,45281,45289
1,45293,45307,45317,45319,45329,45337,45341,45343,45361
2,45377,45389,45403,45413,45427,8117,8123,8147,8161,8171,8179
3,8191,8209,8219,8231,47,53,59,61,67,71,73,83/

C INPUT THE MATRIX A.
C
N=5
S=8
DO 1 I=1,N
READ(5,200) (A(I,J),J=1,S)
1 CONTINUE
WRITE(16,300)
DO 2 I=1,N
WRITE(16,100) (A(I,J),J=1,S)
2 CONTINUE
C
C COMPUTE THE PRODUCT MODULUS BOUND.
C
CALL BOUND(A,WMATF,N,S,M,PRIME)

C COMPUTE THE EXACT MOORE PENROSE INVERSE.
C
CALL ALGO(A,AA,ADAG,WMAT1,WMATF,WMAT2,WMAT3,N,S,M,PRIME,WORK1,
* WORK2,QK,R)
C
C THE FORMATS USED HERE FOR OUTPUT HAVE BEEN SPECIFIED WITH EXAMPLE
C 7.3.1 IN MIND.
WRITE(16,900) M
WRITE(16,500) R
WRITE(16,400) QK
DO 3 I=1,S
WRITE(16,600) (ADAG(I,J),J=1,N)
3 CONTINUE
150 FORMAT('0',13X,3X,'=')
WRITE(16,150)
DO 4 J=1,N
   ADAG(I,J)=ADAG(I,J)/QK
4 CONTINUE
WRITE(16,601) (ADAG(I,J),J=1,N)
4 CONTINUE
900 FORMAT(' ',/13X,' NUMBER OF PRIME MODULI USED = ',I3)
100 FORMAT(' ',13X,8I5)
600 FORMAT(' ',13X,4X,5(F12.1,IX))
601 FORMAT(' ',13X,4X,5(F12.9,IX))
200 FORMAT(8I8)
300 FORMAT('1','///',13X,' THE INPUT MATRIX A IS AS FOLLOWS:',/)
400 FORMAT('0',13X,'+',/13X,' A = ( 1 /',F13.1,') *')
500 FORMAT('0',13X,' THE RANK OF A =',I4)
STOP
END

SUBROUTINE BOUND(A,AA,N,S,M,PRIME)
IMPLICIT INTEGER (A-Z)
DIMENSION PRIME(38),A(N,S),AA(N,N)
REAL*8 TR,NORM,BDD,MIN,WORK,AA

THIS SUBROUTINE DETERMINES THE SIZE OF THE PRODUCT MODULUS
TO INSURE WE CAN EXTRACT THE MOORE PENROSE INVERSE OVER THE
FIELD OF RATIONAL NUMBERS FROM THE MULTIPLE MODULUS SOLUTION
COMPUTED USING A MODIFIED VERSION OF DECELL'S ALGORITHM.

THE BOUND FOR AND (N X S) MATRIX A IS COMPUTED AS FOLLOWS:
LET MIN = MIN(TR(AA'),INFINITY NORM(AA'))
THEN BDD= 2*(N**2) (MIN**(N-1)) IF N**2 > MIN
OTHERWISE BDD = 2*(MIN**N)
BOUND THEN CALLS A SUBROUTINE SELECT WHICH CHOOSES ENOUGH PRIME
MODULI FROM THE ARRAY OF PRIME NUMBER, PRIME(I). SELECT
RETURNS THE NUMBER OF PRIME NUMBERS,M, NEEDED TO EXCEED BDD.

AA' IS BEING COMPUTED HERE.
DO 1 I=1,N
   DO 1 J=1,N
      AA(I,J)=0
      DO 1 K=1,S
         AA(I,J)=AA(I,J)+A(I,K)*AC(K,J)
      1 CONTINUE

C THE TRACE OF AA' IS BEING COMPUTED HERE.
C
   TRI=0
   DO 2 I=1,N
      TRI=TRI+AA(I,I)
   2 CONTINUE

C THE INFINITY NORM OF AA' IS BEING COMPUTED HERE.
C
   NORM=0
   DO 4 I=1,N
      WORK=0
      DO 3 J=1,N
         WORK=WORK+DABS(AA(I,J))
      3 CONTINUE
      NORM=NORM+DABS(AA(I,J))
   4 CONTINUE

C BDD IS BEING DETERMINED NOW.
C
   MIN=DMIN1(TR,NORM)
   BDD=N*N
   IF(MIN.GE.BDD) BDD=M

C ACTUALLY THE LOG(BDD) IS COMPUTED SO THAT THE FLOATING-POINT
C EXPONENT WILL NOT OVERFLOW.
C
C SINCE WE ARE COMPUTING LOG(BDD) RATHER THAN MULTIPLYING BY 2
C WE NEED TO ADD LOG(2.0) TO LOG(BDD) THEN LOG(2*BDD).
C
   BDD=DL0G(2.DO)+N*DLOG(BDD)
C
C SUBROUTINE SELECT IS BEING CALLED TO FIND M THE NUMBER OF
C PRIME MODULI NEEDED TO SOLVE THE PROBLEM.
C
   CALL SELECT(BDD,M,PRIME)
   RETURN
END
THIS SUBROUTINE COMPUTES THE NUMBER OF PRIME MODULI NEEDED TO SOLVE THE GIVEN PROBLEM. THE POSSIBLE PRIME MODULI STORED IN THE ARRAY PRIME. SELECT COMPUTES THE NUMBER M WHICH IS THE NUMBER OF PRIME NUMBERS THAT NEED TO BE MULTIPLIED TOGETHER TO INSURE A CORRECT RATIONAL SOLUTION TO THE MULTIPLE MODULUS PROBLEM CAN BE FOUND.

IF FOR EXAMPLE M=4, PRIME(1)*PRIME(2)*PRIME(3)*PRIME(4) > BDD.

SINCE WE ARE USING LOG(BDD) RATHER THAN BDD, WE NEED TO ACTUALLY FORM THE SUM OF THE LOG(PRIME(I)) RATHER THEN THE PRODUCT. IF THE SUM OF THE LOG(PRIME(I)) EXCEEDS THE LOG(BDD) THEN THE PRODUCT OF THE PRIME(I) EXCEEDS BDD.

PPROD=0
DO 1 I=1,38
PP=PRIME(I)
PPROD=PPROD+DLG(PP)
IF(PPROD.GE.BDD) GO TO 2
1 CONTINUE
2 M=I
RETURN
END

SUBROUTINE ALGO(A,AA,ADAG,W,WF,B,BB,N,S,M,PRIME,RANK,QRANK,QK,R)
IMPLICIT INTEGER (A-Z)
DIMENSION A(N,S),AA(N,N),ADAG(S,N),WF(N,N),B(N,N),RANK(M),QRANK(M),PRIME(M),W(N,N),BB(N,N,M)
REAL*8 ADAG,WF,QK

THIS ALGORITHM USES A MODIFIED VERSION OF DECELL'S ALGORITHM FOR COMPUTING THE MOORE PENROSE INVERSE OF AN (N X S) MATRIX A USING MULTIPLE MODULUS ARITHMETIC. NO PRIOR KNOWLEDGE OF THE RANK(A) IS NECESSARY. (NOTE: N <= TO S)
THE ALGORITHM IS CALLED ALGO, THE ITERATION PROCESS IS AS FOLLOWS.

ITER=0 :
W(0)=NULL Q(0)=-1 B(0)=I
ITER=1 :
W(1)=AA' Q(1)=TR(W(1)) B(1)=W(1)-Q(1)*I
ITER=2 :
W(2)=AA'B(1) Q(2)=(1/2)TR(W(2)) B(2)=W(2)-Q(2)*I
ITER=K-1:
W(K-1)=AA'B(K-2) Q(K-1)=(1/K-1)TR(W(K-1)) B(K-1)=W(K-1)-Q(K-1)*I

ITER=K:
W(K)=AA'B(K-1) Q(K)=(1/K)TR(W(K)) B(K)=W(K)-Q(K)*I

IF THE RANK(A)=K THE W=NULL AT ITER=K+1 AND THE MOORE PENROSE INVERSE OF A IS
ADAG=(1/Q(K))A'B(K-1)
WHERE B IS THE B COMPUTED AT ITER=K-1.

AT THE TIME ALGO IS CALLED M PRIME NUMBERS HAVE ALREADY BEEN CHOOSEN TO BE USED AS THE PRIME MODULI.

THE FOLLOWING IS A LIST OF THE ARRAYS AND MATRICES USED IN ALGO.

PRIME(P) - AN ARRAY OF PRIME MODULI, P=1,M.
RANK(P) - AN ARRAY CONTAINING THE NUMBER OR ITERATIONS COMPLETED FOR THE MODULUS PRIME(P), P=1,M.
QRANK(P) - AN ARRAY CONTAINING THE Q VALUES COMPUTED FOR EACH PRIME(P), P=1,M.
A(I,J) - THE INPUT MATRIX FOR WHICH THE MOORE PENROSE INVERSE IS BEING COMPUTED, I=1,N AND J=1,S.
AA(I,J) - THE WORK MATRIX WHICH STORES (AA')MOD(PRIME(P)) FOR A GIVEN PRIME(P), I=1,N AND J=1, N.
W(I,J) - THE WORK MATRIX USED TO COMPUTE THE MATRICES A(K) IN THE ALGORITHM, I=1,N AND J=1,N. AT THE KTH ITERATION W(I,J) CONTAINS A(K+1).
B(I,J) - THE WORK MATRIX USED TO COMPUTE THE MATRICES B(K) IN THE ALGORITHM, I=1,N AND J=1,N. AT THE KTH ITERATION B(I,J) CONTAINS B(K).
BB(I,J,P) - A MATRIX CONTAINING THE B MATRIX NEED FOR THE MOORE PENROSE INVERSE IN THE ALGORITHM, I=1,N J=1,N AND P=1,M. AT THE KTH ITERATION B(I,J,P) CONTAINS B(K-1).
WF(I,J) - A DOUBLE PRECISION FLOATING POINT MATRIX THAT CONTAINS THE COMBINED SYMMETRIC MIXED RADIX VALUES OF B(K-1), I=1,N AND J=1,N.
ADAG(I,J) - A DOUBLE PRECISION FLOATING POINT MATRIX THAT CONTAINS THE A'B(K-1) PORTION OF THE MOORE PENROSE INVERSE, I=1,S AND J=1,N.

DO 99 P=1,M
COMPUTING (AA')MOD(PRIME(P)) AND STORING IT IN W(I,J). ALSO CHECKING TO SEE IF (AA')MOD(PRIME(P)) IS NULL, IF SO SET RANK(P)=0 IF NOT, SET RANK(P)=1.
NOTE: IF RANK(P)=0 NO ITERATIONS WILL BE PERFORMED FOR THIS PRIME.

RANK(P)=0
DO 1 I=1,N
DO 1 J=1,N
AA(I,J)=0
DO 2 K=1,S
   AA(I,J)=MOD( MOD(MOD(A(I,K),PRIME(P))*MOD(A(J,K),
                           * PRIME(P)), PRIME(P))+AA(I,J), PRIME(P))
   CONTINUE
IF(AA(I,J).LT.0) AA(I,J)=AA(I,J)+PRIME(P)
IF(AA(I,J).NE.0) RANK(P)=1
W(I,J)=AA(I,J)
1 CONTINUE

ITER=0
IF(RANK(P).EQ.0) GO TO 99

DO 3 I=1,N
DO 3 J=1,N
B(I,J)=0
IF(I.EQ.J) B(I,I)=1
3 CONTINUE

ITER=ITER+1
RANK(P)=ITER

DO(500, ITER=ITER+1)
RANK(P)=ITER
500

COMPUTE Q(ITER) = (INVERSE OF ITER)MOD(PRIME(P) * TR(A(ITER))
THIS WHOLE EXPRESSION IS REDUCED MOD(PRIME(P)) AND STORED IN
QRANK(P).

THE INVERSE OF ITER MOD(PRIME(P)) IS FOUND VIA THE EXTENDED
EUCLIDEAN ALGORITHM WHICH IS INCLUDED IN THIS PROGRAM AS A
FORTRAN SUB FUNCTION.
Q = 0
DO 4 I = 1, N
   Q = MOD(W(I, I) + Q, PRIME(P))
   CONTINUE
4
Q = MOD(EUCLID(ITER, PRIME(P)) \* Q, PRIME(P))
IF(Q.LT.0) Q = Q + PRIME(P)
QRANK(P) = Q

C
STORE B(ITER-1) IN BB(I, J, P).
C
DO 22 J = 1, N
   BB(I, J, P) = B(I, J)
22
CONTINUE

C
IF ITER = N THEN A HAS FULL COLUMN RANK AND THE ITERAION
PROCESS STOPS.
C
IF(ITER.EQ.N) GO TO 99
C
DO 7 J = 1, N
   B(I, J) = W(I, J)
   IF(I.EQ.J) B(I, J) = MOD(B(I, I) - Q, PRIME(P))
   IF(B(I, J).LT.0) B(I, J) = B(I, J) + PRIME(P)
7
CONTINUE

C
DO 8 J = 1, N
   W(I, J) = 0
   DO 8 K = 1, N
      W(I, J) = MOD(MOD(AA(I, K) \* B(K, J), PRIME(P)) + W(I, J),
      PRIME(P))
8
CONTINUE

C
CHECK TO SEE IF A(ITER+1) = NULL. IF SO WE EXIT WITH Q(ITER) AND C
B(ITER-1) FOR PRIME(P) AND THEN GO TO THE NEXT PRIME(P). IF C
NOT WE CONTINUE TO THE NEXT ITERATION. C

DO 21 I=1,N
   DO 21 J=1,N
      IF(W(I,J).NE.0) GO TO 500
21   CONTINUE

MAXIM WILL COMPUTE THE LARGEST ELEMENT IN THE ARRAY RANK(P). C
THE LARGEST ELEMENT IN THE ARRAY RANK(P) REPRESENTS THE MAXIMUM C
NUMBER OF ITERATIONS COMPLETED FOR THE DIFFERENT MODULI. THIS C
MAXIMUM NUMBER OF ITERATIONS IS ACTUALLY THE RANK(A) = R. C

IF FOR A GIVEN PRIME(P) RANK(P) < R THIS MEANS QRANK(P)=0 FOR C
THIS PRIME(P). IF RANK(P) < R-1 THIS MEANS B(I,J,P)=NULL FOR C
THIS PRIME(P). IF THESE CONDITIONS HOLD THESE VALUES ARE ASSIGNED C
to RANK(P) AND B(I,J,P) FOR THE PRIME(P). C

CALL MAXIM(RANK,M,R)
   DO 10 P=1,M
      IF(RANK(P).EQ.R) GO TO 10
      QRANK(P)=0.
      IF(RANK(P).EQ.R-1) GO TO 10
      DO 10 I=1,N
         DO 10 J=1,N
            BB(I,J,P)=0
10    CONTINUE

COMPUTE THE SYMMETRIC RESIDUE REPRESENTATION FOR Q(K) AND B(K-1). C

DO 11 P=1,M
   IF(QRANK(P).GT.(PRIME(P)/2)) QRANK(P)=QRANK(P)-PRIME(P)
   IF(RANK(P).LT.R-1) GO TO 11
   DO 11 I=1,N
      DO 11 J=1,N
         IF(BB(I,J,P).GT.(PRIME(P)/2)) BB(I,J,P)=BB(I,J,P)-PRIME(P)
11    CONTINUE
RATION IS A SUBROUTINE THAT COMPUTES THE SYMMETRIC MIXED RADIX VALUE OF AN ARRAY WHICH CONTAINS THE SYMMETRIC RESIDUE REPRESENTATION OF SOME INTEGER. PRIME(P) CONTAINS THE BASE. RATION THEN COMPUTES THE INTEGER REPRESENTED BY THIS NOTATION.

These computations are done in double precision floating point arithmetic.

First Q(K) is computed from the array QRANK(P) and stored in QK.

Next B(K-1) is computed from the matrix BB(I,J,P). The array QRANK(P) is now being used as a work vector so that for a given (I,J) the values of BB(I,J,P), P=1,M are placed in QRANK(P) and then the symmetric mixed radix value of the (I,J)th entry of B(K-1) is computed. The integer values are then stored in WF(I,J).

A multiple of the Moore Penrose inverse is being computed, and being stored in ADAG.

\[
ADAG = A'B(K-1) = A'WF
\]

A multiple of the Moore Penrose inverse of A, and A(K) are passed back to the main program. (Note: A is still intact.)
RETURN
END

C*------------------------------------------------------------------------------------------------*
SUBROUTINE RATION(SOL,X,M,PRIME)
IMPLICIT INTEGER (A-Z)
DIMENSION X(M),PRIME(M)
REAL*8 SOL

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
C RATION IS A SUBROUTINE THAT COMPUTES THE SYMMETRIC MIXED RADIX C
C VALUE OF AN ARRAY X WHICH CONTAINS THE SYMMETRIC RESIDUE C
C REPRESENTATION OF SOME INTEGER. PRIME(P) CONTAINS THE BASE. C
C C RATION THEN CALLS A SUBROUTINE CONVER THAT CONVERTS THE SYMMETRIC C
C MIXED RADIX REPRESENTATION TO THE INTEGER VALUE IT REPRESENTS. C
C
C SOL=X(1)+X(2)*PRIME(1)+X(3)*PRIME(2)*PRIME(1)+ . . . + C
C X(M)*PRIME(M-1)*PRIME(M-2)*...*PRIME(1) C
C
CCcccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
DO 1 P=2,M
DO 2 Q=Q,M
X(P)= MOD(MOD(X(P)-X(Q-1),PRIME(P))*,
* EUCLID(PRIME(Q-1),PRIME(P)),PRIME(P))
IF(IABS(X(P)).GT. (PRIME(P)/2)) X(P)=X(P)-ISIGN(PRIME(P),X(P))
2 CONTINUE
1 CONTINUE
CALL CONVER(SOL,X,M,PRIME)
RETURN

C*------------------------------------------------------------------------------------------------*
SUBROUTINE CONVER(SOL,X,M,PRIME)
IMPLICIT INTEGER (A-Z)
DIMENSION X(M),PRIME(M)
REAL*8 SOL

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
DO 1 P=2,M
SOL=SOL*PRIME (M-P+1 ) + X(M-P+1)
1 CONTINUE
SUBROUTINE MAXIM(IW,N,LIM)
DIMENSION IW(N)
LIM=IW(1)
DO 1 I=1,N
   LIM=MAXO(LIM,IW(I))
1 CONTINUE
RETURN
END

FUNCTION EUCLID(K,M)
IMPLICIT INTEGER (A-Z)
U1=0
U2=M
V1=1
V2=K
1 IF(V2.EQ.1) GO TO 99
   IF(V2.EQ.0) GO TO 98
   Q=U2/V2
   T1=U1-Q*V1
   T2=U2-Q*V2
   U1=V1
   U2=V2
   V1=T1
   V2=T2
   GO TO 1
98 EUCLID=0
   RETURN
99 IF(V1.LT.0) V1=M+V1
   EUCLID=V1
   RETURN
END

//GO.SYSIN DD *
10.4 The Minimum Euclidean Norm Least Squares Method
FORTRAN Program

//S1 EXEC FORTG
IMPLICIT INTEGER (A-Z)
DIMENSION A(7,16),WMAT1(7,7),WMATF(7,7),WMAT2(7,7),
* AA(7,7),WMAT3(7,7,20),WORK1(20),WORK2(20),WLS(7,16),
* PRIME(20),BHAT(7),WBETA(7,20),Y(16)
REAL*8 WHATF,QK,BHAT,BDIV
C
C THIS PROGRAM COMPUTES THE EXACT MINIMUM EUCLIDEAN NORM LEAST
C SQUARES SOLUTION TO THE LINEAR MODEL Y=XB. THE SOLUTION IS
C +
C B = X Y.
C
C THIS IS DONE USING MULTIPLE MODULUS RESIDUE ARITHMETIC.
C
C THE MATRIX A IN THIS PROGRAM CONTAINS THE MATRIX X' FROM THE
C LINEAR MODEL Y=XB.
C
C THIS PARTICULAR "MAIN" PROGRAM IS SETUP TO COMPUTE AN EXACT LEAST
C SQUARES SOLUTION FOR THE LINEAR MODEL OF EXAMPLE 7.3.4.
C
C THE ARRAY PRIME CONTAINS THE POSSIBLE PRIME MODULI THAT WILL
C BE USED.
C
DATA PRIME/ 45233,45247,45259,45263,45281,45289
1,45293,45307,45317,45319,45329,45337,45341,45343,45361
2,45377,45389,45403,45413,45427/
C
C INPUT THE DATA FOR THE LINEAR MODEL
C
N=7
S=16
DO 1 J=1,S
READ(5,200) (A(I,J),I=1,N),Y(J)
1 CONTINUE
WRITE(6,300)
DO 2 J=1,S
WRITE(6,100) Y(J),(A(I,J),I=1,N)
2 CONTINUE
C
C COMPUTE THE PRODUCT MODULUS BOUND.
C
CALL BOUND(A,WMATF,N,S,M,PRIME,Y)
THE EXACT LEAST SQUARES SOLUTION WILL BE COMPUTED IN SUBROUTINE
LSSQ AS

\[ B = \frac{1}{QK} \cdot B_{\text{HAT}}. \]

CALL LSSQ(A, AA, WLS, WMAT1, WBETA, WMAT2, WMAT3, N, S, M, PRIME, WORK1, 
\* WORK2, QK, R, B_{\text{HAT}}, Y)

THE FORMATS USED HERE FOR OUTPUT HAVE BEEN SPECIFIED WITH 
EXAMPLE 7.3.4 IN MIND.

WRITE(6,900) M
WRITE(6,500) R
WRITE(6,400) QK
DO 3 I=1,N
WRITE(6,600) B_{\text{HAT}}(I)
B_{\text{HAT}}(I)=B_{\text{HAT}}(I)/QK
3 CONTINUE
WRITE(6,601)
WRITE(6,602) (B_{\text{HAT}}(I), I=1,N)

THE INPUT DATA FOR THE INTEGRAL LINEAR MODEL \( Y = X B \) ARE AS '
+'FOLLOWS:', /, 13X, 'Y', 4X, 23X, 'X')

\[ Y = C \left( \frac{1}{E20.12} \right). \]

STOP
END

SUBROUTINE BOUND(A, AA, N, S, M, PRIME, Y)
IMPLICIT INTEGER (A-Z)
DIMENSION PRIME(20), Y(S), A(N.S), AA(N, N)
REAL*8 TR, NORM, BDD, MIN, WORK, AA, LSQ

THIS SUBROUTINE DETERMINES THE SIZE OF THE PRODUCT MODULUS 
TO INSURE WE CAN EXTRACT THE LEAST SQUARES SOLUTION OVER THE 
FIELD OF RATIONAL NUMBERS FROM THE MULTIPLE MODULUS SOLUTION 
COMPUTED USING A MODIFIED VERSION OF DECELL'S ALGORITHM TO FIND 
\[ A = (X'). \]
THE BOUND FOR AND (N X S) MATRIX A IS COMPUTED AS FOLLOWS:

LET MIN = MIN(TR(AA'), INFINITY NORM(AA'))

THEN BDD = 2*(N**2)(MIN**(N-1)) IF N**2 > MIN

OTHERWISE BDD = 2*(MIN**N)

THE BOUND FOR THE LEAST SQUARES PROBLEM IS COMPUTED AS FOLLOWS:

BDD = BDD*ABS(X'Y)

BDD = BDD*ABS(AY).

BOUND THEN CALLS A SUBROUTINE SELECT WHICH CHOSES ENOUGH PRIME MODULI FROM THE ARRAY OF PRIME NUMBER, PRIME(I). SELECT RETURNS THE NUMBER OF PRIME NUMBERS, M, NEEDED TO EXCEED THIS BOUND.

AA' IS BEING COMPUTED HERE.

\[
\begin{align*}
&\text{DO 1 } I=1,N \\
&\quad \text{DO 1 } J=1,N \\
&\quad \text{AA(I,J)=0} \\
&\quad \text{DO 1 } K=1,S \\
&\quad \quad \text{AA(I,J)=A(I,K)*A(J,K)+AA(I,J)} \\
&1 \text{ CONTINUE}
\end{align*}
\]

THE TRACE OF AA' IS BEING COMPUTED HERE.

\[
\begin{align*}
&\text{TR=0} \\
&\text{DO 2 } I=1,N \\
&\quad \text{TR=AA(I,I)+TR} \\
&2 \text{ CONTINUE}
\end{align*}
\]

THE INFINITY NORM OF AA' IS BEING COMPUTED HERE.

\[
\begin{align*}
&\text{NORM}=0 \\
&\text{DO 4 } I=1,N \\
&\quad \text{WORK}=0 \\
&\quad \text{DO 3 } J=1,N \\
&\quad \quad \text{WORK=DABS(AA(I,J))+WORK} \\
&\quad \text{3 CONTINUE} \\
&\quad \text{NORM=DMAX1(NORM,WORK)} \\
&4 \text{ CONTINUE}
\end{align*}
\]

BDD IS BEING DETERMINED NOW.

\[
\begin{align*}
&\text{MIN=DMIN1(TR,NORM)} \\
&\text{BDD=N**N} \\
&\text{IF(MIN.GE.BDD) BDD=MIN} \\
&\text{WORK=0}
\end{align*}
\]
DO 5 I=1,N
   LSQ=0
   DO 55 K=1,S
      LSQ=A(I,K)*Y(K)+LSQ
   55 CONTINUE
   WORK=DABS(LSQ)+WORK
5 CONTINUE
   I=M+1
   
   C ACTUALLY THE LOG(BDD) IS COMPUTED SO THAT THE FLOATING-POINT
   C EXPONENT WILL NOT OVERFLOW.
   C
   C SINCE WE ARE COMPUTING LOG(BDD) RATHER THAN MULTIPLYING BY 2
   C WE NEED TO ADD LOG(2.0) TO LOG(BDD) THE FORM LOG(2*BDD).
   C
   BDD=DLOG(2.0)+DLOG(WORK)+N*DLOG(BDD)
   C
   SUBROUTINE SELECT IS BEING CALLED TO CHOOSE THE NUMBER OF
   C PRIMES (M) NEEDED FROM THE TABLE OF PRIMES TO GENERATE AN
   C EXACT SOLUTION.
   C
   CALL SELECT(BDD,M,PRIME)
   RETURN
   END

C********************************************************************
SUBROUTINE SELECT(BDD,M,PRIME)
IMPLICIT INTEGER (A-Z)
DIMENSION PRIME(20)
REAL*8 BDD,PPROD,PP

C C C THIS SUBROUTINE COMPUTES THE NUMBER OF PRIME MODULI NEEDED C
C TO SOLVE THE GIVEN PROBLEM. THE POSSIBLE PRIME MODULI
C ARE STORED IN THE ARRAY PRIME. SELECT COMPUTES THE NUMBER
C M WHICH IS THE NUMBER OF PRIME NUMBERS THAT NEED TO BE MULTIPLIED
C TOGETHER TO INSURE A CORRECT RATIONAL SOLUTION TO THE
C MULTIPLE MODULUS PROBLEM CAN BE FOUND.
C
C IF FOR EXAMPLE M=4, PRIME(1)*PRIME(2)*PRIME(3)*PRIME(4) > BDD.
C
C SINCE WE ARE USING LOG(BDD) RATHER THAN BDD, WE NEED TO
C ACTUALLY FORM THE SUM OF THE LOG(PRIME(I)) RATHER THEN
C THE PRODUCT. IF THE SUM OF THE LOG(PRIME(I)) EXCEEDS THE
C THE LOG(BDD) THEN THE PRODUCT OF THE PRIME(I) EXCEEDS BDD.
C
PPRD0=0
   DO 1 I=1,20
      PP=PRIME(I)
      PPROD=PPRD0+DLOG(PP)
1 C

C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

IF(PPROD.GE.BDD) GO TO 2
1 CONTINUE
2 M=I
RETURN
END

SUBROUTINE LSSQ(A,AA,WLS,W,BETA,B,BB,N,S,M,PRIME,RANK,QRANK,QK, 
R,BHAT,Y)
IMPLICIT INTEGER (A-Z)
DIMENSION A(N,S),AA(N,N),WLS(N,S),BETA(N,M),B(N,N),RANK(M), 
Q(RANK(M),PRIME(M),BHAT(N),W(N,N),BB(N,N,M),Y(S)
REAL*8 BHAT,QK

THIS SUBROUTINE COMPUTE THE EXACT LEAST SQUARES SOLUTION FOR THE 
LINEAR MODEL Y=X'B USING MULTIPLE MODULUS RESIDUE ARITHMETIC WITH 
a MULTIPLE MODULUS BASE OF M-PRIMES. THE SUBROUTINE'S MATRIX A=X' 

THE ALGORITHM USES A MODIFIED VERSION OF DECELL'S ALGORITHM FOR 
COMPUTING THE MOORE PENROSE INVERSE OF AN (N X S) MATRIX A USING 
MULTIPLE MODULUS ARITHMETIC. NO PRIOR KNOWLEDGE OF THE RANK(A) 
is necessary. (NOTE: N <= TO S)

THE ALGORITHM IS CALLED ALGO, THE ITERATION PROCESS IS AS FOLLOWS.

ITER=0 :
W(0)=NULL Q(0)=-1 B(0)=I
ITER=1 :
W(1)=AA' Q(1)=TR(W(1)) B(1)=W(1)-Q(1)*I
ITER=2 :
W(2)=AA'B(1) Q(2)=(1/2)TR(W(2)) B(2)=W(2)-Q(2)*I
. 
ITER=K-1:
W(K-1)=AA'B(K-2) Q(K-1)=(1/K-1)TR(W(K-1)) B(K-1)=W(K-1)-Q(K-1)*I
ITER=K :
W(K)=AA'B(K-1) Q(K)=(1/K)TR(W(K)) B(K)=W(K)-Q(K)*I

IF THE RANK(A)=K THE W=NULL AT ITER=K+1 AND THE MOORE PENROSE 
INVERSE OF A IS 
A =A'/B(K-1) 
WHERE B IS THE B COMPUTED AT ITER=K-1.

THEN THE LEAST SQUARES SOLUTION IS 
B =A'/B(K-1)'AY. 
AT THE TIME ALGO IS CALLED M.

PRIME NUMBERS HAVE ALREADY BEEN

CHOSEN TO BE USED AS THE PRIME MODULI.

CONTINUE

THE FOLLOWING IS A LIST OF THE ARRAYS AND MATRICES USED IN ALGO.

PRIME(P) - AN ARRAY OF PRIME MODULI, P=1,M.
RANK(P) - AN ARRAY CONTAINING THE NUMBER OR ITERATIONS COMPLETED FOR THE MODULUS PRIME(P), P=1,M.
QRANK(P) - AN ARRAY CONTAINING THE Q VALUES COMPUTED FOR EACH PRIME(P), P=1,M.
A(I,J) - THE INPUT MATRIX FOR WHICH THE MOORE PENROSE INVERSE IS BEING COMPUTED, I=1,N AND J=1,S.
AA(I,J) - THE WORK MATRIX WHICH STORES (AA')MOD(PRIME(P)) FOR A GIVEN PRIME(P), I=1,N AND J=1,N.
W(I,J) - THE WORK MATRIX USED TO COMPUTE THE MATRICES A(K) IN THE ALGORITHM, I=1,N AND J=1,N. AT THE KTH ITERATION W(I,J) CONTAINS A(K+1).
B(I,J) - THE WORK MATRIX USED TO COMPUTE THE MATRICES B(K) IN THE ALGORITHM, I=1,N AND J=1,N. AT THE KTH ITERATION B(I,J) CONTAINS B(K).
BB(I,J,P) - A MATRIX CONTAINING THE B MATRIX NEED FOR THE MOORE PENROSE INVERSE IN THE ALGORITHM, I=1,N J=1,N AND P=1,M. AT THE KTH ITERATION B(I,J,P) CONTAINS B(K-1).
Y(J) - AN INPUT ARRAY OF DEPENDENT VARIABLE VALUES FOR THE LINEAR MODEL Y=XB, J=1,S.
WLS(I,J) - THE WORK MATRIX USED TO COMPUTE B(K-1)'A MODULO P, I=1,N AND J=1,S.
BETA(I,P) - AN ARRAY WHICH STORES THE BHAT VALUES MODULO P, I=1,S AND P=1,M.
BHAT(I) - A DOUBLE PRECISION FLOATING-POINT ARRAY THAT THE VALUES B(K-1)'AY OF THE LEAST SQUARES SOLUTION.

DO 99 P=1,M

RANK(P)=0

NOTE: IF RANK(P)=0 NO ITERATIONS WILL BE PREFORMED FOR THIS PRIME.
DO 1 I=1,N
DO 1 J=1,N
AA(I,J)=0
DO 2 K=1,S
   AA(I,J)=MOD(MOD(MOD(A(I,K),PRIME(P))*MOD(A(J,K),PRIME(P)),PRIME(P))+AA(I,J),PRIME(P))
2 CONTINUE
IF(AA(I,J).LT.O) AA(I,J)=AA(I,J)+PRIME(P)
IF(AA(I,J).NE.O) RANK(P)=1
W(I,J)=AA(I,J)
1 CONTINUE
C
ITER=0
C
IF(RANK(P).EQ.0) GO TO 99
C
DO 3 I=1,N
   DO 3 J=1,N
      B(I,J)=0
      IF(I.EQ.J) B(I,I)=1
3 CONTINUE
C
DO 4 I=1,N
   Q=MOD(W(I,I)+Q,PRIME(P))
4 CONTINUE
Q = MOD(EUCLID(ITER, PRIME(P)) * Q, PRIME(P))
IF(Q.LT.0) Q = Q + PRIME(P)
QRANK(P) = Q
C
STORE B(ITER-1) IN BB(I,J,P).
C
DO 22 I = 1, N
DO 22 J = 1, N
BB(I,J,P) = B(I,J)
22 CONTINUE
C
IF(ITER.EQ.N) GO TO 99
C
COMPUTE B(ITER), REDUCE THIS MOD(PRIME(P)) AND STORE IT IN B.
C
DO 7 I = 1, N
DO 7 J = 1, N
B(I,J) = W(I,J)
IF(I.EQ.J) B(I,J) = MOD(B(I,I) - Q, PRIME(P))
IF(B(I,J).LT.0) B(I,J) = B(I,J) + PRIME(P)
7 CONTINUE
C
COMPUTE A(ITER+1), REDUCE THIS MOD(PRIME(P)) AND STORE IT IN W.
C
DO 8 I = 1, N
DO 8 J = 1, N
W(I,J) = 0
DO 8 K = 1, N
W(I,J) = MOD(MOD(AA(I,K)*B(K,J), PRIME(P)) + W(I,J),
* PRIME(P))
8 CONTINUE
C
CHECK TO SEE IF A(ITER+1) = NULL. IF SO WE EXIT WITH Q(ITER) AND C
B(ITER-1) FOR PRIME(P) AND THEN GO TO THE NEXT PRIME(P). IF C
NOT WE CONTINUE TO THE NEXT ITERATION.
DO 21 I=1,N
DO 21 J=1,N
    IF(W(I,J) .NE. 0) GO TO 500
21   CONTINUE
C
C MAXIM WILL COMPUTE THE LARGEST ELEMENT IN THE ARRAY RANK(P).
C THE LARGEST ELEMENT IN THE ARRAY RANK(P) REPRESENTS THE MAXIMUM
C NUMBER OF ITERATIONS COMPLETED FOR THE DIFFERENT MODULI. THIS
C MAXIMUM NUMBER OF ITERATIONS IS ACTUALLY THE RANK(A) = R.
C
C IF FOR A GIVEN PRIME(P) RANK(P) < R THIS MEANS QRANK(P)=0 FOR
C THIS PRIME(P). IF RANK(P) < R-1 THIS MEANS B(I,J,P)=NULL FOR
C THIS PRIME(P). IF THESE CONDITIONS HOLD THEN QRANK(P) IS SET TO
C ZERO AND BETA(I,P) IS SET TO THE NULL VECTOR.
C
CALL MAXIM(RANK,M,R)
C
BETA(I,P)=B(K-1)'AY MODULO P IS COMPUTED HERE USING
C WLS(I,J) AS WORK SPACE.
C
DO 10 P=1,M
    IF(RANK(P).LT.R) QRANK(P)=0
    IF(RANK(P).LT.R-1) GO TO 110
    DO 100 I=1,N
    DO 100 J=1,S
        WLS(I,J)=0
        DO 100 K=1,N
            WLS(I,J)=MOD( MOD( BB(K,I,P)* MOD(A(K,J),PRIME(P)),
                        * PRIME(P)) + WLS(I,J), PRIME(P))
100   CONTINUE
110   DO 200 I=1,N
        BETA(I,P)=0
        IF(RANK(P).LT.R-1) GO TO 200
        DO 200 K=1,S
            BETA(I,P)=MOD( WLS(I,K)* MOD(Y(K),PRIME(P)),
                           * PRIME(P)) + BETA(I,P), PRIME(P))
200   CONTINUE
10   CONTINUE
C
COMPUTE THE SYMMETRIC RESIDUE REPRESENTATION FOR Q(K) AND BETA.
C
...
DO 11 P=1,M
   IF(QRANK(P).GT.(PRIME(P)/2)) QRANK(P)=QRANK(P)-PRIME(P)
   IF(RANK(P).LT.R-1) GO TO 11
   DO 11 I=1,N
      IF(BETA(I,P).GT.(PRIME(P)/2)) BETA(I,P)=BETA(I,P)-PRIME(P)
 11 CONTINUE

C RATION IS A SUBROUTINE THAT COMPUTES THE SYMMETRIC MIXED RADIX
C VALUE OF AN ARRAY WHICH CONTAINS THE SYMMETRIC RESIDUE
C REPRESENTATION OF SOME INTEGER. PRIME(P) CONTAINS THE BASE.
C RATION THEN COMPUTES THE INTEGER REPRESENTED BY THIS NOTATION.
C THESE COMPUTATIONS ARE DONE IN DOUBLE PRECISION FLOATING POINT
C ARITHMETIC.
C FIRST Q(K) IS COMPUTED FROM THE ARRAY QRANK(P) AND STORED IN QK.
C NEXT BHAT(I) IS COMPUTED FROM THE ARRAY BETA(I,P). THE ARRAY
C QRANK(P) IS NOW BEING USED AS A WORK VECTOR SO THAT FOR A
C GIVEN (I) THE VALUES OF BETA(I,P), P =1,M, ARE PLACED IN
C QRANK(P) AND THEN THE SYMMETRIC MIXED RADIX VALUE OF THE
C (I)TH ENTRY OF BHAT IS COMPUTED.
C
CALL RATION(QK,QRANK,M,PRIME)
   DO 12 I=1,N
      DO 13 P=1,M
         QRANK(P)=BETA(I,P)
      13 CONTINUE
   CALL RATION(BHAT(I),QRANK,M,PRIME)
 12 CONTINUE
C
RETURN
END

SUBROUTINE RATION(SOL,X,M,PRIME)
IMPLICIT INTEGER (A-Z)
DIMENSION X(M),PRIME(M)
REAL*8 SOL
C RATION IS A SUBROUTINE THAT COMPUTES THE SYMMETRIC MIXED RADIX
C VALUE OF AN ARRAY X WHICH CONTAINS THE SYMMETRIC RESIDUE
C REPRESENTATION OF SOME INTEGER. PRIME(P) CONTAINS THE BASE.
C RATION THEN CALLS A SUBROUTINE CONVER THAT CONVERTS THE SYMMETRIC
C MIXED RADIX REPRESENTATION TO THE INTEGER VALUE IT REPRESENTS.
C SOL=X(1)+X(2)*PRIME(1)+X(3)*PRIME(2)*PRIME(1)+...+X(M)*PRIME(M-1)*PRIME(M-2)*...*PRIME(1)
C

DO 1 Q=2,M
   DO 2 P=Q,M
      X(P)= MOD(MOD(X(P)-X(Q-1),PRIME(P))*,
            EUCLID(PRIME(Q-1),PRIME(P)),PRIME(P))
      IF(IABS(X(P)).GT.(PRIME(P)/2)) X(P)=X(P)-ISIGN(PRIME(P),X(P))
   2 CONTINUE
1 CONTINUE
CALL CONVER(SOL,X,M,PRIME)
RETURN
END

SUBROUTINE CONVER(SOL,X,M,PRIME)
IMPLICIT INTEGER (A-Z)
DIMENSION X(M),PRIME(M)
REAL*8 SOL,E
DATA E/Z0210000000000000/
SOL=X(M)*E
DO 1 P=2,M
   SOL=SOL*PRIME(M-P+1) + X(M-P+1)*E
1 CONTINUE
RETURN
END

SUBROUTINE MAXIM(IW,N,LIM)
DIMENSION IW(N)
SOL=X(1)+X(2)*PRIME(1)+X(3)*PRIME(2)*PRIME(1)+...+X(M)*PRIME(M-1)*PRIME(M-2)*...*PRIME(1)
C THESE COMPUTATIONS ARE DONE IN DOUBLE PRECISION FLOATING POINT
C ARITHMETIC USING HORNER'S RULE.
C THE FLOATING-POINT EXPONENT IN THIS PRODUCT-SUM COMPUTATION IS
C BEING DEFLATED BY E=16**(-63). THE VALUE OF E IS INPUT BY
C THE DATA STATEMENT ABOVE IN HEX NOTATION.
SOL=X(M)*E
DO 1 P=2,M
   SOL=SOL*PRIME(M-P+1) + X(M-P+1)*E
1 CONTINUE
RETURN
END
LIM=IW(1)
DO 1 I=1,N
   LIM=MAX0(LIM,IW(I))
1 CONTINUE
RETURN
END

FUNCTION EUCLID(K,M)
IMPLICIT INTEGER (A-Z)

FUNCTION EUCLID IS THE STANDARD EXTENDED EUCLIDEAN ALGORITHM.
EUCLID COMPUTES THE INVERSE OF K MOD(M).

U1=0
U2=M
V1=1
V2=K
1 IF(V2.EQ.1) GO TO 99
   IF(V2.EQ.0) GO TO 98
   Q=U2/V2
   T1=U1-Q*V1
   T2=U2-Q*V2
   U1=V1
   U2=V2
   V1=T1
   V2=T2
GO TO 1
98 EUCLID=0
RETURN
99 IF(V1.LT.0) V1=M+V1
   EUCLID=V1
RETURN
END

//GO.SYSIN DD *