Fraction-free RNS Algorithms for Solving Linear Systems

Peter R Turner
Mathematics Department, US Naval Academy, Annapolis MD 21402
prt@sma.usna.navy.mil

Abstract

This paper is concerned with overcoming the arithmetic problems which arise in the solution of linear systems with integer coefficients. Specifically, solving such systems using (integer) Gauss elimination or its variants usually results in severe growth in the dynamic range of the integers that must be represented. To alleviate this problem, a Residue Number System (RNS) can be utilized so that large integers can be represented by a vector of residues which require only short wordlengths. RNS arithmetic however cannot easily handle any divisions that are needed in the solution process. This paper presents fraction-free *algorithms for the solution of integer systems. This does involve divisions — but only divisions where the result is known to be an exact integer. The other principal contribution of this paper is the presentation of an RNS division algorithm for exact integer division which does not require any conversion to standard binary form. It uses entirely modular arithmetic, perhaps including a step equivalent to RNS base extension.*

1. Introduction

This paper is concerned with the computer arithmetic aspects of one of the fundamental problems of scientific computing, the solution of systems of linear equation. Specifically we are concerned here with systems in which all elements of the coefficient matrix and the right-hand side are integers. Such systems arise in many practical applications including signal processing tasks where data are often expressed as integers reflecting the resolution and quantization of the signals. This was discussed in the context of adaptive beamforming in [7].

The biggest single difficulty with solving integer systems is the growth in the integer dynamic range during (integer) Gauss elimination. (Of course, other algorithms can be used but they are typically less readily modified to integer computation.) This difficulty can be countered by the use of Residue Number Systems, RNS. This allows a large dynamic range to be accommodated without needing a very long integer wordlength.

The basic idea of a residue number system is that an integer is represented by its residues modulo each of a basis set of (usually prime) numbers. (See [3], [10], for example.) Using an RNS basis of L relatively prime integers $\{p_1, p_2, \ldots, p_L\}$, an integer a is represented by the vector $(a_1, a_2, \ldots, a_L)$ of residues defined by

$$a_k = a \bmod p_k = \langle a \rangle_{p_k} \qquad (k = 1, 2, \ldots, L) \quad (1)$$

The *dynamic range* of this system is the product of the basis elements: $M = \prod_{k=1}^{L} p_k$ which is to say that a set of M consecutive integers can be represented uniquely in this system. Often the range used is symmetric so that if $M = 2P + 1$ integers in the interval $[-P, P]$ can be represented. The integer a can be recovered by use of the Chinese Remainder Theorem, CRT.

RNS arithmetic has the additional advantage of a natural parallelism since addition, subtraction and multiplication can be performed entirely within the various modular arithmetics. Specifically, for two integers a, b represented as in (1),

$$a + b = \left(\langle a_1 + b_1 \rangle_{p_1}, \langle a_2 + b_2 \rangle_{p_2}, \ldots, \langle a_L + b_L \rangle_{p_L} \right)$$
$$a \times b = \left(\langle a_1 \times b_1 \rangle_{p_1}, \langle a_2 \times b_2 \rangle_{p_2}, \ldots, \langle a_L \times b_L \rangle_{p_L} \right)$$

The operations in the different moduli are independent and so can be performed simultaneously if parallel modular arithmetic channels are available.

Much of the recent literature on RNS arithmetic has been devoted to the development of RNS division algorithms. Division is not a natural RNS (or

1063-6889/97 $10.00 © 1997 IEEE

even integer) operation. Typically RNS division algorithms require use of the CRT or some extended RNS basis. See [2], [4], [6], [8], for example.

One of the more promising approaches to reducing the growth in dynamic range is to use *fraction-free* algorithms. The first significant progress in such algorithms for linear algebra problems was made by Bareiss [1]. His algorithm can be simplified and extended, for example, to fraction-free LU decomposition of a matrix. The difficulty with using this approach in RNS arithmetic is that it requires divisions of complete submatrices by a common factor. However, these divisions are not completely general. These common factors are known and are generated automatically by the algorithm. It follows that in all cases the results are known to be exact integers. Such divisions can be achieved within the RNS system using entirely modular arithmetic. The advantages of parallelism are thus retained with, perhaps, one additional step. This step is logically equivalent to base extension for any moduli for which the quotient takes the indeterminate form 0/0.

These last developments are the contribution of the present paper. In Section II, we describe the fraction-free algorithms for linear systems. The main novelty comes from the extension of the basic idea to a fraction-free equivalent of LU factorization which allows multiple systems with a common coefficient matrix to be solved more economically. The central role of exact integer division becomes apparent. Section III is primarily concerned with this problem. We present a simple algorithm for RNS division where the result is known to be an *exact integer* in advance. For algorithms such as those of Section II, this is an important development which may render RNS arithmetic a practical approach to the solution of integer linear systems.

2. Fraction-free algorithms for linear systems

We begin with a brief description of fraction-free Gauss elimination, FFGE. The algorithm presented here is a simplification of Bareiss's original algorithm. It remains valid for symbolic computation in more general ring settings but we shall concentrate here on its application for integer arithmetic. This version of the algorithm resulted from a comparison of the floating-point and integer divisionless forms of Gauss elimination [11].

The essence of the algorithm can be seen by considering a 3×3 matrix. From the comparative analysis mentioned above, we see that the result of divisionless Gauss elimination starting with the (integer) matrix

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

is an upper triangular matrix

$$\begin{bmatrix} a & b & c \\ & ae-db & af-dc \\ & & a*\det A \end{bmatrix}$$

The key observation here is that the final element has the factor a. If this factor is removed (by exact division) the diagonal will now consist of the principal minors of increasing size of the original matrix. Moreover for a larger matrix, *every* element below or to the right of the 3, 3 position will have this factor a at this stage since the computation for every such position is essentially identical. This factor is therefore removable form the entire active matrix. A similar comment applies at each subsequent stage of the elimination — except that the common factor "moves" down the diagonal. These factors can be systematically removed as the algorithm proceeds. The following algorithm (written using MATLAB subscript notation to help highlight potential parallel operations) describes this procedure for the augmented matrix $[A|b]$ so that the final column is identified with the right-hand side of the original system.

Algorithm FFGE Fraction-free Gauss elimination

Input $n \times (n+1)$ augmented matrix A
Compute
for $i = 1 : n-1$
  for $j = i+1 : n$
    $a_{j,i+1:n+1} := a_{i,i} * a_{j,i+1:n+1} - a_{j,i} * a_{i,i+1:n+1}$
  if $(i > 1)$
    $a_{i+1:n,i+1:n+1} := a_{i+1:n,i+1:n+1}/a_{i-1,i-1}$
  $a_{i+1:n,i} := 0$
Output (modified) augmented matrix A

It is immediately apparent that FFGE is not *division*-free but it is *fraction*-free since all divisors are exact factors of their dividends. The algorithm above has no mention of pivoting. If a zero pivot is encountered then the simplest pivoting strategy is to interchange this row with the first row having a nonzero entry in the pivot column. This is also the pivoting strategy recommended in both [1] and [11].

Example 1 *The above algorithm applied to the following system produces the result shown below. It is easy to check that the leading diagonal consists of the determinants of the principal minors of A.*

$$A = \begin{bmatrix} 2 & 9 & 0 & 0 \\ 0 & 3 & 0 & 3 \\ 6 & 5 & 5 & 0 \\ 6 & 8 & 6 & 4 \end{bmatrix} \quad \mathrm{b} = \begin{bmatrix} 11 \\ 6 \\ 16 \\ 24 \end{bmatrix}$$

gives the output

$$\begin{bmatrix} 2 & 9 & 0 & 0 & 11 \\ 0 & 6 & 0 & 6 & 12 \\ 0 & 0 & 30 & 132 & 162 \\ 0 & 0 & 0 & -102 & -102 \end{bmatrix}$$

The entries in the final triangular (or, strictly, echelon) matrix are all determinants of minors of the original augmented matrix. Specifically, the i,j-th entry is the determinant of the minor consisting of rows $1,2,\ldots,i$ and columns $1,\ldots,i-1,j$. This fact may perhaps be exploited in other settings such as using resultants to solve systems of polynomial equations.

It is clear that the removal of these common factors necessarily reduces the dynamic range requirement. Since we are concerned with integer arithmetic these factors all have magnitude of at least 1. What is not immediately apparent is the extent to which the dynamic range is reduced. General bounds are difficult to establish and are unrealistically pessimistic due to allowing for all worst cases. A more useful indication of the savings achieved can be obtained from a simple comparison of the final values for a_{nn} by the Algorithm FFGE and by the *divisionless* algorithm. This comparison was introduced in [11]. A simpler characterization of this is obtained in terms of the matrix elements in the fraction-free algorithm.

Let a_{ij} denote the values obtained from Algorithm FFGE. Denote by a^D_{nn} the final value obtained by divisionless Gauss elimination for the n,n element of A. Then we have $a_{nn} = \det A$ and

$$a^D_{nn} = \left(a_{11}^{n-2} a_{22}^{n-3} \cdots a_{n-2,n-2}\right) \det A \qquad (2)$$

This growth factor is a realistic estimate of the saving in dynamic range achieved by the fraction-free algorithm. Even for the very small dimensional example above, this represents a saving of a factor of 24 in the magnitude of a_{44}.

For a 12×12 matrix with initial entries represented by 8-bit integers, we may conjecture a "typical" magnitude of initial entries around 16 (approximately the square-root of the initial range). Even if we assume that no growth in the diagonal entries the ratio $a^D_{nn}/\det A$ given by (2) is $\prod_{k=1}^{10} 16^k = 16^{55} = 2^{220}$. That is some 220 bits have been saved from the effective wordlength demanded by this dynamic range growth!

The use of Gauss elimination in its basic form is greatly restricted by the fact that if subsequent systems with the same coefficient matrix are to be solved then the complete solution process must be repeated. In the general real arithmetic setting this problem is easily overcome by the simple modification of GE to a LU factorization algorithm by storing the multipliers in the subdiagonal positions. This is not so immediately available in the fraction-free setting: [1] includes variations on the original algorithm which do not include a fraction-free LU factorization algorithm. The principal reason for this difficulty is that the removal of the common factors has different effects on different parts of the matrix. This means that the resulting upper triangular system is not a simple factor of the original matrix.

However, the overriding objective of the LU factorization is not the *factorization* but the fact that subsequent systems can be solved without starting from scratch. In this sense, we can indeed obtain a fraction-free LU algorithm — and the forward and back substitution algorithms needed to complete the solution process. The modifications to Algorithm FFGE for the "factorization" are precisely equivalent to those for floating-point computation. The "multipliers" turn out to be just the corresponding matrix entries which can then be used to mimic the matrix operations for the right-hand side during the forward substitution. (This means that the subdiagonal entries in the pivot column are left *unchanged*.) The back substitution can then be completed precisely as for Gauss elimination itself. This algorithm is also described in [9]. Again, in the interest of simplicity, we omit any mention of pivoting from the algorithm description.

Algorithm FFLU Fraction-free "LU factorization"

Input $n \times n$ matrix A
Compute
for $i = 1 : n-1$
  for $j = i+1 : n$
    $a_{j,i+1:n} := a_{i,i} * a_{j,i+1:n} - a_{j,i} * a_{i,i+1:n}$
  if $(i > 1)$
    $a_{i+1:n,i+1:n} := a_{i+1:n,i+1:n}/a_{i-1,i-1}$
Output (modified) matrix A

The only significant changes from the FFGE algorithm are that the command to replace the "zeroed"

elements with 0 is omitted, and the row operations apply only to the coefficient matrix. The right-hand side is not mentioned in this algorithm. The two "factors" are stored together by overwriting the original matrix as usual. Unlike the conventional LU factorization both L and U have the same diagonal.

Example 2 *The above algorithm applied to the matrix of Example 1 produces the result shown below.*

$$A = \begin{bmatrix} 2 & 9 & 0 & 0 \\ 0 & 3 & 0 & 3 \\ 6 & 5 & 5 & 0 \\ 6 & 8 & 6 & 4 \end{bmatrix}$$

gives the output

$$\begin{bmatrix} 2 & 9 & 0 & 0 \\ 0 & 6 & 0 & 6 \\ 6 & -44 & 30 & 132 \\ 6 & -38 & 36 & -102 \end{bmatrix}$$

from which we obtain

$$L = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 6 & -44 & 30 & 0 \\ 6 & -38 & 36 & -102 \end{bmatrix}$$

$$U = \begin{bmatrix} 2 & 9 & 0 & 0 \\ 0 & 6 & 0 & 6 \\ 0 & 0 & 30 & 132 \\ 0 & 0 & 0 & -102 \end{bmatrix}$$

The product of these matrices is

$$\begin{bmatrix} 4 & 18 & 0 & 0 \\ 0 & 36 & 0 & 36 \\ 12 & -210 & 900 & 3696 \\ 12 & -174 & 1080 & 14928 \end{bmatrix}$$

which bears little apparent resemblance to the original matrix A.

To see the effect of this algorithm we need first to detail the forward and back substitution algorithms which yield an integer vector which is a scaled copy of the true solution (with a known scale factor) so that the exact solution can be obtained in either the rationals or the integers as appropriate. (The scale factor is the determinant of the original matrix.)

Algorithm FFsolve Fraction-free forward and back substitution

Input $n \times n$ matrix A (from FFLU), right-hand side $\mathbf{b}$
Compute (Forward substitution)
$\mathbf{r} := \mathbf{b}$
for $i = 1 : n-1$
  for $j = i+1 : n$
    $r_j := a_{i,i} * r_j - a_{j,i} * r_i$
    if $(i > 1)$ $r_j := r_j / a_{i-1,i-1}$
Compute (Back substitution)
$d := a_{nn}$
for $i = n-1 : -1 : 1$
  $r_i := d * r_i$
  for $j = n : -1 : i+1$
    $r_i := r_i - a_{ij} * r_j$
  $r_i := r_i / a_{ii}$
Output scaled solution $\mathbf{r}$, scale factor d.

It is easy to see from the lop structure that these forward and back substitution procedures require $O(n^2)$ operations compared with $O(n^3)$ for the FFLU phase. (The actual operation counts of course vary from the floating-point case due to the "cross-multiplications" and the removal of the common factors.) This is precisely the same comparison as is valid for the floating-point situation and so the same complexity benefits are derived from this method.

Example 3 *The above algorithm applied to the system of Example 1, using the factorization in Example 2, produces the results shown below.*

The "factors" found in Example 2 are

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 6 & -44 & 30 & 0 \\ 6 & -38 & 36 & -102 \end{bmatrix}, \begin{bmatrix} 2 & 9 & 0 & 0 \\ 0 & 6 & 0 & 6 \\ 0 & 0 & 30 & 132 \\ 0 & 0 & 0 & -102 \end{bmatrix}$$

The forward substitution phase yields the vector $\mathbf{r} = (11, 12, 162, -102)'$ and the back substitution, in turn, gives $\mathbf{r} = (-102, -102, -102, -102)'$ and $d = -102$ from which the integer solution is easily obtained. In the situation where there is no integer solution, $\mathbf{r}$ and d would yield the rational solution.

The solution for a different right-hand side $\mathbf{b} = (20, 18, 31, 56)'$ is obtained by simply recomputing Algorithm FFsolve. In this case we get $\mathbf{r} = (20, 36, 618, -408)'$ from the forward substitution and then $\mathbf{r} = (-102, -204, -306, -408)'$ still with $d = -102$ results from the back substitution. Again the exact solution is easily computed.

Just as with the FFGE algorithm we can easily obtain a comparison of the elements generated by this FFsolve and those that would be obtained form

a divisionless back substitution. The comparison is similar to that summarized in (2) - except that we should note that the matrix entries for the divisionless algorithm would already have suffered that first growth. The combined effect is therefore that the saving in dynamic range is much enhanced.

If the systems are to be solved on a parallel computer then it is easy to modify the FFGE algorithm to perform fraction-free Gauss-Jordan decomposition to take advantage of the ease of elimination both above and below the diagonal. The details are omitted here, see [9].

There is an obvious difficulty in implementing these algorithms in a residue number system. RNS arithmetic does not (in any simple way) admit division and, clearly, the removal of the common factors in these algorithms is division–intensive. However, these are not general RNS divisions but are known to be *exact* integer operations since the divisors are *known* factors of the various numerators. Such an operation is amenable to RNS arithmetic without the need to leave the naturally parallel environment of residue arithmetic. The next section is concerned with the implementation of exact integer division in RNS arithmetic.

3. Exact RNS integer division

We first recall our basic notation for a residue number system. For our exact integer division algorithm it is necessary that *all moduli are prime* so that we have an RNS basis $\{p_1, p_2, \ldots, p_L\}$ of L prime numbers. The reason that each modulus is required to be prime is simply that each ring $\mathcal{Z}_{p_k}$ is then a *field* which is to say that every element has a unique multiplicative inverse. The dynamic range of this system is $M = \prod_{k=1}^{L} p_k$ which would normally be used to represent either the set of integers $\{0, 1, \ldots, M-1\}$ or, for $M = 2P+1$, the set $\{-P, -P+1, \ldots, P\}$. For most linear algebra applications of RNS arithmetic the symmetric range is more suitable.

A second convention that we adopt throughout this section is that all divisions under discussion are known to have *exact* integer results. In this special context, division can be performed within the RNS system. The only complication that can arise is that a limited base-extension is sometimes needed. Strictly, the term base-*extension* is inappropriate here: what may really be needed is the residue of the quotient relative to one (or more) modulus for which the division is undefined. For this reason we shall refer to base-*completion* when such operations are needed.

The exact division algorithm is based on the fact that if all residues are nonzero and division is known to be exact, then modular division can be used. This is stated more precisely as part of the following elementary theorem on modular arithmetic.

Theorem 4 *Suppose m, n, q are integers, all representable in the RNS, and such that $q = m/n$. Then*

(1) For any k such that $n_k \neq 0$,

$$q_k = \langle q \rangle_{p_k} = \langle m_k / n_k \rangle_{p_k} \tag{3}$$

(2) If $n_k = 0$, then $m_k = 0$ also, and q_k is not defined by (3).

The first part follows immediately from the facts that multiplication is a modular operation and that $\mathcal{Z}_{p_k}$ is an algebraic field. The proof of the second part simply depends on the observation that if $n_k = 0$, then n is a multiple of p_k. Since m is an integer multiple of n, it follows that $m_k = 0$ as well.

Equation (3) also shows that if $n_k \neq 0$ for every k, then such exact divisions *are* modular operations. The primary significance of the second part is that for any modulus for which the denominator has 0 residue so does the numerator. That is, the only difficulty arises from a quotient of the form 0/0 in one or more moduli. (We cannot have $x/0$ where $x \neq 0$.)

Example 5 *As a first example we consider the RNS system using the basis primes* $3, 5, 7, 11$, *and* 13 *which are each representable using just* 4 *bits. The symmetric dynamic range for this RNS is* $[-7507, 7507]$. *(This system is sufficient to accommodate all the computation for the linear systems of Section II, including all interim results obtained before the removal of known factors.)*

One of the divisions which takes place early in the LU factorization is $264/2$. *Now* 264 *is represented in our RNS by the vector* $(0, 4, 5, 0, 4)$ *while* 2 *is* $(2, 2, 2, 2, 2)$. *The modular division then yields* $(0, 2, 6, 0, 2)$, *as can be readily verified. These residues are indeed the representation of the quotient* 132 *in this particular RNS.*

Later in this solution process the division $(-3060)/30$ must be performed. The apparent problem here is that the residues of the numerator and denominator are $(0, 0, 6, 9, 8)$ and $(0, 0, 2, 8, 4)$. The divisions for the first two moduli are undefined. Since $\langle 8^2 \rangle_{11} = \langle 64 \rangle_{11} = 9$, it follows that the final

three residues of this quotient are 3,8,2 which are the true residues of the result -102 relative to the moduli 7, 11 and 13. The question now is "Does this provide a generally applicable technique?" The answer is "Yes" as is seen from the following fairly elementary result. For simplicity we shall state this result in the context of a symmetric dynamic range $[-P, P]$.

Theorem 6 *Suppose m, n, q are integers, all representable in the RNS, and such that $q = m/n$ and that $n_k = 0$ for some k. Then $|q| \leq P/p_k$. It follows that q is representable in the RNS with basis $\{p_1, p_2, \ldots, p_L\} \backslash p_k$. Furthermore, q_k can then be obtained by* base-completion. *(That is base-extension from $\{p_1, p_2, \ldots, p_L\} \backslash p_k$ to $\{p_1, p_2, \ldots, p_L\}$.)*

To see the first part, we simply observe that if $n_k = 0$, then $m_k = 0$ and so $m = m' p_k, n = n' p_k$. Since $|m|, |n| \leq P$, it follows that $|m'|, |n'| \leq P/p_k$ and therefore $|q| \leq P/p_k$, too. That q is representable in that reduced RNS is then immediate. Once that representation is obtained base-completion necessarily yields the remaining residue q_k.

We note that if more than one n_k is zero this Theorem can be applied recursively to obtain the appropriate quotient. Of course, the base-completion could be performed simultaneously for all "undetermined" residues. The problem of base-extension has been considered in several contexts. One simple description of a general purpose algorithm based on conversion to a mixed-radix system is given in [5], see also [10].

We now summarize this RNS division process.

Algorithm RNSdiv Exact RNS integer division

Input Integers m, n represented in the RNS with basis $\{p_1, p_2, \ldots, p_L\}$; n is known to divide m.
Compute
For every k such that $n_k \neq 0$, $\quad q_k = \langle m_k / n_k \rangle_{p_k}$
For any k for which $n_k = 0$, obtain q_k by base-completion
Output Quotient q represented by $(q_1, q_2, \ldots, q_L)$.

Example 7 *We return to the example above, where the division $(-3060)/30$ is to be performed in the RNS with basis 3,5,7,11,and 13.*

The RNS representations of the numerator and denominator are $(0, 0, 6, 9, 8)$ and $(0, 0, 2, 8, 4)$. We deduce, from the first step of the algorithm, that $q_7 = 3, q_{11} = 8$, and $q_{13} = 2$, as before. To complete the algorithm, we apply a base-extension algorithm to obtain q_3, q_5. It is readily verifiable that the residues given above are the representation of -102 in the symmetric RNS with basis $\{7, 11, 13\}$. Any correct base-extension algorithm for a symmetric RNS will therefore yield the remaining residues $q_3 = 0, q_5 = 3$.

It is worth noting here that not all base-extension algorithms which are valid for the dynamic range $[0, M]$ remain valid for symmetric dynamic ranges. However, if the algorithm of Gregory & Matula [5] is modified so that "symmetric" residues are used throughout, it will return the correct residues for the base-completion phase here. For the above example, we would first convert the quotient to its *symmetric* mixed radix form:

$$q = a_0 + a_1 (7) + a_2 (7)(11) \qquad (4)$$

where the coefficients are constrained to satisfy $-3 \leq a_0 \leq 3, -5 \leq a_1 \leq 5, -6 \leq a_2 \leq 6$. This is achieved with entirely modular arithmetic and yields, for this case, $a_0 = 3, a_1 = -4, a_2 = -1$. Now evaluating the right-hand side of (4) relative to the remaining moduli 3 and 5 gives $q_3 = 0, q_5 = 3$, as expected.

The Algorithm RNSdiv above allows exact integer division to be performed in the RNS without the need for conversion to standard binary form. Its use within fraction-free linear algebra makes the use of RNS arithmetic for such problems more practical.

There is of course still some problem of range growth but the fraction-free algorithms help to control this. For the example system we have used here, the final factored matrix and solutions generated numbers no greater than 162 (in absolute value) while the second system generated elements as large as 618. (Before the removal of common factors however, substantially larger quantities may occur such as the -3060 used in the division example above.) To illustrate the savings made by the fraction-free algorithm in the range growth, we simply observe that if *division*-free versions of the same algorithms were used, the solution of our first example generates integers as large as 146,880. The fraction-free algorithm has reduced the dynamic range for this 4×4 example by a factor of about 1000. At least three more moduli (each needing 5 bits instead of 4) would be needed for this. The comparison of the results of the forward elimination phase summarized in (2) and the corresponding analysis for FFsolve illustrate that this sort of saving in the dynamic range is typical of what may be expected.

4. Conclusions

In this paper we have presented a new approach to the solution of linear systems of equations using RNS arithmetic. This resulted from two distinct developments. The first was a (simplification and) extension of the fraction-free Gauss elimination algorithm of Bareiss [1] to a more powerful algorithm which has the properties of a conventional LU factorization algorithm. These fraction-free algorithms rely on the identification of certain known common factors which arise during the elimination. Because they are known factors, it follows that their removal can be accomplished by *exact* integer division.

The removal of these common factors has two distinct benefits. Firstly, the dynamic range needed for the computation is substantially reduced. This makes the integer solution of such systems more feasible. The reduction of the dynamic range growth leads to the second benefit and the second main development of this paper. Fast integer computation can often be enhanced by the use of RNS arithmetic. The drawbacks have always been the dynamic range growth and the lack of a good RNS division algorithm. The second development addresses this last issue. In the event of integer division where the result is known to be an exact integer, we have presented an algorithm for RNS division which only uses modular arithmetic. This algorithm combined with the fraction-free algorithms makes a realistic approach to the solution of linear systems using RNS arithmetic and its natural parallelism.

Simple illustrative examples demonstrated the merits of these algorithms: simplicity of the linear algebra and the RNS exact division algorithm even in the situation where some residues of the denominator are 0, and a dramatic reduction in the dynamic range which would of course become much more extreme for larger (more realistic) systems.

Acknowledgement The author is grateful to the Naval Academy Research Council and the Office of Naval Research for financial support under grant N0001496WR20018.

References

[1] E.H.Bareiss, *Sylvester's identity and multistep integer-preserving Gaussian elimination*, Math Comp 22 (1968) 565-578.

[2] W.A.Chren, Jr., *A new residue number system division algorithm*, Comp. Math. Appl. 19 (1990) 13-29.

[3] G.I.Davida and B.Litow, *Fast parallel arithmetic via modular representation*, SIAM J Comp 20 (1991) 756-765.

[4] D.Gamberger, *New approach to integer division in residue number systems*, Proc ARITH10, IEEE Comp Soc, Washington DC, 1991, pp 84-91.

[5] R.T.Gregory and D.W.Matula, *Base conversion in residue number systems*, Proc ARITH3, IEEE Comp Soc, Washington DC, 1975, pp 117-125.

[6] M.A.Hitz and E.Kaltofen, *Integer division in residue number systems*, IEEE TC 44 (1995) 983-989.

[7] B.J.Kirsch and P.R.Turner, *Adaptive beamforming using RNS arithmetic*, Proc ARITH11, IEEE Comp Soc, Washington DC, 1993, pp 36-43.

[8] Mi Lu and J-S.Chiang, *A novel division algorithm for the residue number system*, IEEE TC 41 (1992) 1026-1032.

[9] G.C.Nakos, P.R.Turner and R.M.Williams, *Fraction-free algorithms for linear and polynomial equations*, NAWCADPAX TR, NAWC Aircraft Division, Patuxent River, MD, 1997

[10] N.S.Szabo and R.I.Tanaka, *Residue arithmetic and its application to computer technology*, McGraw-Hill, 1967.

[11] P.R.Turner, *A simplified fraction-free integer Gauss elimination algorithm*, NAWCADPAX 96-196-TR, NAWC Aircraft Division, Patuxent River, MD, 1996