J. Symbolic Computation (1998) **26**, 71–87

Modular Arithmetic for Linear Algebra Computations in the Real Field

IOANNIS Z. EMIRIS [†] AND VICTOR Y. PAN [‡] AND YANQIANG YU

INRIA, B.P. 93, 06902 Sophia-Antipolis, France. emiris@sophia.inria.fr.
Department of Mathematics and Computer Science, Lehman College,
City University of New York, Bronx, NY 10468, USA. VPAN@LCVAX.LEHMAN.CUNY.EDU.
Ph.D. Program in Computer Science, The Graduate School and University Center,
City University of New York, New York, NY 10036, USA.

The aim of this work is to decrease the bit precision required in computations without affecting the precision of the answer, whether this is computed exactly or within some tolerance. By precision we understand the number of bits in the binary representation of the values involved in the computation, hence a smaller precision requirement leads to a smaller complexity. We achieve this by combining the customary numerical techniques of rounding the least significant bits with the algebraic technique of reduction modulo an integer, which we extend to the reduction modulo a positive number. In particular, we show that if the sum of several numbers has small magnitude, relative to the magnitude of the summands, then the precision used in the computation of this sum can be decreased without affecting the precision of the answer. Furthermore, if the magnitude of the inner product of two vectors is small and if one of them is filled with "short" binary numbers, then again we may decrease the precision of the computation. The method is applied to the iterative improvement algorithm for a linear system of equations whose coefficients are represented by "short" binary numbers, as well as to the solution of PDEs by means of multigrid methods. Some results of numerical experiments are presented to demonstrate the power of the method.

1. Introduction

This article combines an algebraic and a numeric approach in order to decrease the required precision of some important computations in linear algebra, thus improving the time and space complexity of the computations without affecting the output precision. This work may be regarded as an effort to exploit the interaction between computer algebra and numerical computation, which is nowadays an area of strong interest.

We rely on the observation that some major computations in linear algebra involve inner products whose magnitude is substantially less than the magnitudes of some coordinates of the two input vectors. Such examples include the iterative improvement of an approximate solution to a linear system of equations and the solution of discretized partial differential equations (PDEs) by means of multigrid methods (compare Pan and Reif (1992), Pan and Reif (1993)). Then, in many cases, we may ignore the most significant

[†] Partially supported by European ESPRIT project FRISCO (LTR 21.024)
[‡] Supported by NSF Grant CRR 9020690 and PSC CUNY Awards ## 662478 and 664334.

0747–7171/98/070071 + 17 $30.00/0

digits in the representation of these coordinates and thus decrease the precision of the computations without affecting the output errors. Such an idea must be counter-intuitive for a numerical analyst, who views the loss of the most significant digits of the operands as a major disaster of numerical computing, because of the implied contamination of the output. In particular, many numerical analysts believe that the usual scheme for iterative improvement of approximate solution to the linear system of equations cannot produce correct output without investment of substantial additional computer resources, unless the residual vector is computed with double precision. According to the classical numerical analysis of matrix computations, the convergence of the iterative improvement algorithm is ensured if the coefficient matrix is well-conditioned and if the residual vector is computed with double precision in each iteration step (Golub and Van Loan, 1989). A more recent study (Higham, 1996, p. 234) supplies an estimate for the deterioration of the convergence when the single precision is used throughout the entire computation. The deterioration factor, for an $n \times n$ matrix A with condition number κ, is $2n\kappa$, which is substantial unless A is very well-conditioned. Higham considers this limitation on proceeding with single precision as a major obstacle for the practical use of the algorithm. Our techniques, however, enable us to compute the correct solution by performing the computations of this algorithm with a precision that is substantially *smaller* than the *single* machine precision.

On the other hand, the power of our approach should be less surprising to the designers of algebraic algorithms, who are familiar with using reduction modulo an integer as a common means of decreasing the precision of computations (Aho, Hopcroft and Ullman, 1974, Gregory, 1980, Davenport, Siret and Tournier, 1988). Unlike the previous works, however, we were able to utilize modular (residue) arithmetic within some customary schemes of numerical computing.

The main idea is to combine the numerical technique of rounding off, which truncates the least significant digits, with reduction modulo a noninteger positive m. To achieve this, we have introduced new techniques which we call *backward modular reduction* (b.m.r.) and *backward binary segmentation* (b.b.s.). The b.b.s. combines the algebraic techniques of b.m.r. with the customary numerical techniques of truncation of the least significant bits. Both b.m.r. and b.b.s. require to estimate the range in which we may truncate the operands depending on the estimated magnitudes of the output values and of their allowed approximation errors. The analysis goes from the output values back to the operands, thus motivating the adjectives "backward".

Besides applications to linear algebra, our modular reduction techniques can be useful in some other numerical computations, for instance, in calculating certain special functions to a limited precision, as suggested by an anonymous referee. In general, our method applies to computations where bounds exist on the size of the answers. Alternatively, it may be useful when we seek an answer to some limited precision, for example when this answer is to be combined with data of limited precision, such as those obtained from experiments. We return to the issue of applicability of our method in section 8.

To take advantage of these techniques we need a computer (such as the MasPar parallel computer) that performs lower precision computations faster than higher precision ones. Due to the recent progress in the data compression area, one may expect that more computers of this kind will be available in the future. We also recall the recent specific progress in data compression for basic matrix operations (Pan, 1991, Pan, 1992, Pan, 1993, Bini and Pan, 1994), which implies a possible acceleration of computations by a factor of order P/b, where P is the the number of bits in the machine's single precision and b is the actual number of bits required.

We organize our presentation as follows. In the next two sections, we apply the algebraic techniques of backward modular reduction to summation and the computation of the inner product of two vectors. In section 4, we complement these techniques with the numerical techniques of truncation and apply the resulting b.b.s. process to the computation of the inner product. We combine b.b.s. with a modification of the iterative improvement algorithm in section 5, where we also apply briefly b.b.s. to the classical iterative algorithm. Section 6 presents two more sample applications, namely to the Gauss-Seidel iteration and to the solution of PDEs by means of multigrid algorithms. Section 7 contains the results of our numerical experiments. We conclude with future work in section 8.

The third author has performed numerical tests whose results have been reported in section 7. The other sections were written by the first two authors, who revised (Pan, 1992a) and its modular unpublished version of 1995.

2. Backward modular reduction for summation

This section introduces our modular method for limiting the precision needed in summing real inputs. This discussion extends the classical context of modular (or residue) arithmetic from the rational setting (Aho, Hopcroft and Ullman, 1974, Bini and Pan, 1994) to the real numbers.

Definition 2.1. *For a positive m and any real r, define the unique real number $r \bmod m$ such that $0 \leq r \bmod m < m$, $r = r \bmod m + jm$, for some integer j.*

It is easy to show that the set $\mathbf{R}/m$ of real numbers reduced modulo m has the structure of an additive group, and actually of a ring. Indeed, $0 \leq r^{(i)} < m$ if $r^{(i)} \in \mathbf{R}/m$, $i = 1, 2$, and in $\mathbf{R}/m$ the sum of $r^{(1)}$ and $r^{(2)}$ equals $(r^{(1)} + r^{(2)}) \bmod m$.

Lemma 2.2. *For any pair of a positive m and a real $r \in [-m/2, m/2)$, $r = r \bmod m$, if $r \bmod m < m/2$, otherwise $r = (r \bmod m) - m$.*

Remark 2.3. *This lemma also allows us to choose between two equivalent representations of $\mathbf{R}/m$, namely by the set $[0, m)$ and by the set $[-m/2, m/2)$. The latter is preferable, for instance, in testing the sign of algebraic expressions, as in (Brönnimann, Emiris, Pan and Pion, 1997).*

For the sake of simplicity we assume the binary representation of real numbers, as in most modern computers, and apply modular reduction for $m = 2^{\mu}$ and integer μ. In particular, we concentrate on binary rationals of finite length. Thus all results are stated in the context of binary rationals, and all logarithms, denoted $\log(\cdot)$, are binary. The only exceptions are examples 2.6, 3.4, 4.1 and section 7, where a decimal representation is used in order to illustrate that the decimal case, and more generally, the b-ary case for any integer $b \geq 2$, can be treated similarly.

Now suppose that we are given a bound on the output magnitude, when the sum $r = \sum_{i=1}^{k} r^{(i)}$ of k integers $r^{(1)}, \ldots, r^{(k)}$ is to be computed. Hence, the output precision can be bounded. Let

$$|r| < 2^h \quad \text{and} \quad 2|r| < m = 2^\mu \quad \text{for integers } h, \mu. \tag{2.1}$$

With a binary number representation, $\mu = h + 1$, but in general this does not hold. Even if $|r^{(i)}| > m$ for some i, $1 \le i \le k$, we still obtain r by computing modulo m. In other words, a precision of $h + 1$ bits is sufficient. It is straightforward to extend this method to the case of binary rational summands.

Lemma 2.4. *Assume that we wish to compute the sum r of k binary rationals $r^{(1)}, \dots, r^{(k)}$, where*

$$|r| < 2^h \quad \textit{and} \quad r^{(i)} = 2^{g(i)} z^{(i)},\ i = 1, \dots, k,$$

for integers $h, g(i)$ and $z^{(i)}$ and $g(i)$ maximum. Then we can obtain r exactly *by a computation modulo $m = 2^{h+1}$, with a precision of $h + 1 - g$ bits, where $g = min_i\{g(i)\}$. If the numbers are represented in some base b, then m is an integer power of b such that $2|r| < 2\,b^h \le m = b^\mu$.*

The two examples that follow illustrate this technique, for which we will use the name *backward modular reduction* (b.m.r.), since it extends the reduction modulo m *backward*, from the sum r to the summands $r^{(i)}$.

Example 2.5. Let $k = 3$,

$$r^{(1)} = (0.1111111111111)_2\,, \quad r^{(2)} = -(0.010101010111)_2\,, \quad r^{(3)} = -(0.1010101010111)_2.$$

Then computing with the full precision gives $r = -(0.11)_2 2^{-10}$. On the other hand, if we know in advance that $2|r| < 2^{-9}$, we may set $m = 2^{-9}$ and compute in $\mathbf{R}/m$. Since $h = -10$ and $g = -13$, the precision, given by lemma 2.4, is $h + 1 - g = -10 + 1 + 13 = 4$ bits.

$$r^{(1)}\text{mod } m = (0.1111)_2 2^{-9}\,, \quad r^{(2)}\text{mod } m = (0.001)_2 2^{-9}\,, \quad r^{(3)}\text{mod } m = (0.1001)_2 2^{-9}$$

and the computation is

$$\begin{aligned} r \text{ mod } m &= ((r^{(1)}\text{mod } m) + (r^{(2)}\text{mod } m) + (r^{(3)}\text{mod } m)) \text{ mod } m = \\ &= (1.101)_2 2^{-9}\text{mod } m = (0.101)_2 2^{-9}\ . \end{aligned}$$

Lemma 2.2 implies that $r = -m + r \text{ mod } m = -(0.11)_2 2^{-10}$.

Example 2.6. This example uses the decimal representation. Let

$$k = 3,\ r^{(1)} = 3.1416048,\ r^{(2)} = -2.718288,\ r^{(3)} = -0.4233216.$$

Then computing with the full precision of 8 decimals gives us $r = -(0.48)10^{-5}$. However, knowing in advance that $|r| < (0.5)10^{-5}$, we may set $m = 10^{-5}$ and $h = -5$. Moreover $g = -7$. Hence we can perform the computation modulo m with a precision of $-5+1+7 = 3$ decimals.

$$r^{(1)}\text{mod } m = (0.48)10^{-5},\ r^{(2)}\text{mod } m = (0.2)10^{-5},\ r^{(3)}\text{mod } m = (0.84)10^{-5}$$

and the computation is

$$\begin{aligned} r \text{ mod } m &= ((r^{(1)}\text{mod } m) + (r^{(2)}\text{mod } m) + (r^{(3)}\text{mod } m))\text{mod } m = \\ &= ((1.52)10^{-5})\text{mod } m = (0.52)10^{-5}. \end{aligned}$$

Apply lemma 2.2 and obtain $r = -m + (r \bmod m) = -(0.48)10^{-5}$.

3. Backward modular reduction for the inner product

This section extends the backward modular reduction from sums to the inner product of two real vectors, which is a fundamental operation in linear algebra.

We denote the inner product of two vectors by

$$r = \sum_{i=1}^{k} r^{(i)}, \quad r^{(i)} = u^{(i)} v^{(i)},$$

where the two vectors are

$$\vec{u} = (u^{(i)}), \quad \vec{v} = (v^{(i)}), \quad i = 1, \ldots, k.$$

To compute r, we may first multiply $u^{(i)}$ and $v^{(i)}$ pairwise, for all i, and then sum the products. Knowing in advance a bound on $|r|$ as in the hypothesis of lemma 2.4, we may apply b.m.r. at the summation stage. Let us extend b.m.r. also to the multiplication stage. Let

$$u^{(i)} = 2^{c(i)} z^{(i)} \ , \qquad v^{(i)} = 2^{d(i)} w^{(i)} \ , \tag{3.1}$$

where $c(i)$, $d(i)$, $z^{(i)}$, and $w^{(i)}$ are given integers, and $c(i)$, $d(i)$ are the maximum possible, for $i = 1, \ldots, k$.

Definition 3.1. *For any real x and positive m,*

$$x \text{ trunc } m = \begin{cases} x \bmod m & \text{if } x \geq 0 \\ -((-x) \bmod m) & \text{otherwise.} \end{cases}$$

Thus, x trunc 2^g denotes the value of x with its leftmost bits truncated up to and including the bit corresponding to 2^g.

Lemma 3.2. *With the above notation, if $\mu > c(i) + d(i)$ for some i, then*

$$r^{(i)} \bmod 2^{\mu} = u^{(i)} v^{(i)} \bmod 2^{\mu} = ((u^{(i)} \text{ trunc } 2^{\mu - d(i)})(v^{(i)} \text{trunc } 2^{\mu - c(i)})) \bmod 2^{\mu}.$$

If $\mu \leq c(i) + d(i)$ then $r^{(i)} \bmod 2^{\mu} = 0$.

Proof. The case $\mu \leq c(i) + d(i)$ is obvious. Let $\text{sgn}(\cdot) \in \{-1, 0, 1\}$ represent the sign function. Then

$$\begin{aligned} &(u^{(i)} \text{ trunc } 2^{\mu - d(i)})(v^{(i)} \text{ trunc } 2^{\mu - c(i)}) = \\ &= \text{sgn}(u^{(i)}) \text{sgn}(v^{(i)}) (|u^{(i)}| \bmod 2^{\mu - d(i)})(|v^{(i)}| \bmod 2^{\mu - c(i)}). \end{aligned}$$

So the general case reduces to the case $u^{(i)}, v^{(i)} \geq 0$. Then $u^{(i)}$ trunc $2^{\mu - d(i)} = z^{(i)} 2^{c(i)} + l_u 2^{\mu - d(i)}$ and $v^{(i)}$ trunc $2^{\mu - c(i)} = w^{(i)} 2^{d(i)} + l_v 2^{\mu - c(i)}$, for some integers l_u, l_v. Therefore,

$$(u^{(i)} \text{ trunc } 2^{\mu - d(i)})(v^{(i)} \text{ trunc } 2^{\mu - c(i)}) = z^{(i)} w^{(i)} 2^{c(i) + d(i)} \bmod 2^{\mu},$$

and the lemma is established. □

Due to lemma 3.2, we may reduce $u^{(i)}$ or $-u^{(i)}$ modulo $2^{\mu - d(i)}$ and $v^{(i)}$ or $-v^{(i)}$ modulo

$2^{\mu-c(i)}$ when we compute $r^{(i)}$ mod 2^{μ}. In other words, we again extend the modular reduction *backwards*, this time from the product $r^{(i)}$ to the multiplicands $u^{(i)}$ and $v^{(i)}$. If $|u^{(i)}| \geq 2^{\mu-d(i)}$ and/or $|v^{(i)}| \geq 2^{\mu-c(i)}$, then such a backward modular reduction decreases the binary lengths of $u^{(i)}$ and/or $v^{(i)}$. Applying lemma 3.2 with $h = \mu - 1$ we arrive at the main result of this section.

Lemma 3.3. *Given k-dimensional vectors $\vec{u}$ and $\vec{v}$ with binary rational entries as in expression (3.1), assume that their inner product r satisfies $|r| < 2^h$ for some integer h. Then using a precision of $h - g + 1$ bits, where $g = \min_i\{c(i), d(i)\}$, coupled with the reduction modulo 2^{h+1} suffices to compute exactly each summand $r^{(i)}$, $i = 1, \ldots, k$, as well as r.*

Example 3.4. This example demonstrates the above techniques in the decimal case. Let

$$k = 3, \vec{u} = (u^{(i)}) = (0.4176, 1.8877, 1.248), \vec{v} = (v^{(i)}) = (7.523, -1.44, -0.3392).$$

Then $r^{(1)}$, $r^{(2)}$, $r^{(3)}$, and r take on the same values as in example 2.6, so that $m = 10^{-5}$. We next represent $u^{(i)}$ and $v^{(i)}$ as in expression (3.1):

$$\vec{u} = (10^{-4}4176, 10^{-4}18877, 10^{-3}1248), \vec{v} = (10^{-3}7523, -10^{-2}144, -10^{-4}3392),$$

so that $c(1) = -4, c(2) = -4, c(3) = -3, d(1) = -3, d(2) = -2, d(3) = -4$. In lemma 3.2, $\mu = -5$ and, replacing the powers of 2 by powers of 10, we obtain that

$$u^{(1)}\ \mathrm{trunc}\ 10^{-2} = (0.76)10^{-2}, v^{(1)}\ \mathrm{trunc}\ 10^{-1} = (0.23)10^{-1}, r^{(1)} \bmod 10^{-5} = (0.48)10^{-5},$$
$$u^{(2)}\ \mathrm{trunc}\ 10^{-3} = (0.7)10^{-3},\ v^{(2)}\ \mathrm{trunc}\ 10^{-1} = -(0.4)10^{-1},\ r^{(2)} \bmod 10^{-5} = (0.2)10^{-5},$$
$$u^{(3)}\ \mathrm{trunc}10^{-1} = (0.48)10^{-1}, v^{(3)}\ \mathrm{trunc}\ 10^{-2} = -(9.2)10^{-3}, r^{(3)} \bmod 10^{-5} = (0.84)10^{-5}.$$

This gives us the same values of $r^{(i)}$ mod m, $i = 1, 2, 3$, as in example 2.6.

4. Backward binary segmentation for the inner product

The algebraic technique of b.m.r. has enabled us to get rid of the most significant bits of the input and intermediate values in the computation of the inner products. Next, we will combine this approach with the customary numerical technique of truncating the least significant bits (Atkinson, 1978, Conte and de Boor, 1980), also known as chopping, in order to further decrease the precision of computing. We start with an example that motivates our approach and illustrates the subtlety of the issue.

Example 4.1. Assume that, for the same input as in example 3.4, we should compute the inner product r on a computer that chops all values to 5 floating point decimal digits. If we apply the straightforward numerical algorithm, we arrive at the following, where $f\ell(\cdot)$ denotes rounding off and $r^{(i)*}$ denotes the obtained approximate result.

$$\begin{array}{rll} r^{(1)*} & = f\ell((0.4176)(7.523)) & = (0.31416)10^1, \\ r^{(2)*} & = -f\ell((1.8877)(1.44)) & = -(0.27182)10^1, \\ r^{(3)*} & = -f\ell((1.248)(0.3392)) & = -0.42332, \\ r^* & = f\ell(r^{(1)*} + r^{(2)*} + r^{(3)*}) & = (0.8)10^{-4}. \end{array}$$

Here, the rounding errors have completely contaminated the correct output value $r =$

$-(0.48)10^{-5}$, which, however, can be correctly computed if we operate with the same number of bits but apply the b.m.r., as described in sections 2 and 3.

Definition 4.2. (binary segments) *Let g and h denote two integers, $g < h$; let q_i denote 0 or 1 for all i; let $S[g, h)$ denote the* binary segment *of real numbers representable as $\pm \sum_{i=g}^{h-1} q_i 2^i$ and let $S_{g,h}(q)$ denote the projection of a real number $q = \pm \sum_{i=-\infty}^{h-1} q_i 2^i$, $|q| < 2^h$, into the binary segment $S[g, h)$, that is, $S_{g,h}(q) = \pm \sum_{i=g}^{h-1} q_i 2^i$ with $sgn(q) = sgn(S_{g,h}(q))$.*

Note that, for any real q, q trunc $2^h = q$ trunc $2^g + S_{g,h}(q)$. If $q \geq 0$, then $q \bmod 2^\mu \in S[-\infty, \mu)$. Writing $|r| < 2^h$ and $r = 2^g z$ where h, g, z are integers, is equivalent to assuming $r \in S[g, h)$. Below we seek r with limited precision such that $|r| < 2^h$, in other words we seek projection $S_{g,h}(r)$.

Theorem 4.3. *Suppose that the sum r of k binary rationals is sought with absolute output error bounded by 2^t and that $|r| < 2^h$, for integers t and h. Then it suffices to compute modulo $m = 2^{h+1}$ on the projections of the summands into binary segment $S[t - \lceil \log k \rceil, h + 1)$, hence computing with a precision of $h + 1 - t + \lceil \log k \rceil$ bits.*

Proof. The modulo operation is justified by lemma 2.4. The error bound implies that the bit at position t in the computed answer must be correct, hence we should use all bits up to position $t + \lceil \log k \rceil$ in the summands, provided that we perform additions in a tree fashion. □

Adopting the notation of equations (3.1), assume

$$|w^{(i)}| \leq 2^{a(i)}, \quad i = 1, \ldots, k, \tag{4.1}$$

for some fixed integers $a(i)$. This hypothesis is used by the algorithm below; the corresponding hypothesis on the $|z^{(i)}|$ would lead to an analogous algorithm. Moreover, suppose that $r = \sum_{i=1}^{k} u^{(i)} v^{(i)}$ satisfies the bound

$$|r| + 2^t < 2^h, \tag{4.2}$$

for fixed integers h and t, where t expresses the numerical tolerance and h expresses the output magnitude bound as in previous sections. Then consider the following algorithm for the inner product.

Algorithm 4.4. This algorithm applies b.m.r. to the $u^{(i)}$ but not the $v^{(i)}$ values.

Input: integers $a(i)$, $d(i)$, $\mu = h + 1$, and t, a natural number k, and real $u^{(i)}$, $v^{(i)}$, $i = 1, \ldots, k$; these quantities satisfy $v^{(i)} = 2^{d(i)} w^{(i)}$, for some integers $w^{(i)}$ so that the $d(i)$ are maximized, and equations (4.1) and (4.2).

Output: an approximation r^* to the inner product r of $\vec{u}$ and $\vec{v}$, such that $|r^* - r| \leq 2^t$.

Computations: To simplify notation, let $S_g(q)$ denote the projection of some real q into binary segment $S[g, \mu)$, where μ is fixed and given in the input. Successively compute the following quantities.

```
1.  $g = \lfloor \log(2^t/k) \rfloor = t - \lceil \log k \rceil$,
2.  $\tilde{u}^{(i)} = S_{g-a(i)-d(i)}(u^{(i)} \bmod 2^{\mu-d(i)})$,   $i = 1, \ldots, k$,
```

3. $\tilde{r}^{(i)} = S_g((\tilde{u}^{(i)} v^{(i)}) \bmod 2^\mu), \quad i = 1, \ldots, k,$

4. $\tilde{r} = \left(\sum_{i=1}^{k} \tilde{r}^{(i)} \right) \bmod 2^\mu,$

5. $r^* = \tilde{r}$, if $|\tilde{r}| < 2^{\mu-1}$, otherwise $r^* = \tilde{r} - 2^\mu$.

Input integers $a(i)$ and $d(i)$, $i = 1, \ldots, k$, can be computed from the vector entries $v^{(i)}$. To draw the parallel with the previous sections, note that the last steps compute modulo $m = 2^\mu$. The computation of $\tilde{u}^{(i)}$ is performed with a precision of $\mu - g + a(i) = h + 1 - t + \lceil \log k \rceil + a(i)$ bits at stage 2, for $i = 1, \ldots, k$. Stages 3 and 4 compute in $\mathbf{R}/2^\mu$ with a precision of $\mu - g = h + 1 - t + \lceil \log k \rceil$ bits. Step 5 applies lemma 2.2.

Lemma 4.5. *Assuming $|r| + 2^t < 2^h$, for $h = \mu - 1$, the above algorithm correctly computes r^* such that $|r^* - r| \leq 2^t$.*

Proof. We would have had $\tilde{r} = r \bmod 2^\mu$, due to lemma 3.2, if we excluded chopping by replacing g by $-\infty$ in stages 2 and 3, and then we would have had $r^* = r$, based on lemma 2.4 and the bound $|r^*| < 2^h$. This bound follows from bound (4.2) and the output condition. It remains to deduce the latter by estimating the errors due to chopping.

For completeness we will supply these simple routine estimates: In stage 2, chopping errors are less than $2^{g-a(i)-d(i)}$. This bound turns into 2^g after multiplication of $\tilde{u}^{(i)}$ by the integer $v^{(i)}$ since $|v^{(i)}| \leq 2^{a(i)+d(i)}$ due to equations (3.1) and (4.1). In stage 3 the latter error bound grows to 2^{g+1}, due to chopping. In stage 4, the k errors, each having magnitude bounded by 2^{g+1}, are added with each other, which gives the overall error bound $k2^{g+1}$ for the approximation of $r \bmod 2^\mu$ by $\tilde{r}$, and we observe that $2^t \geq k2^{g+1}$, for g defined at stage 1. □

We will call this technique the *backward binary segmentation* (b.b.s.) process since it extends the output bound (4.2) backward, to the operands, restricting their binary values to certain segments. The discussion above proves our main result, by recalling that $\mu = h + 1$.

Theorem 4.6. *Given k-dimensional vectors $\vec{u}$ and $\vec{v}$ with binary rational entries, assume that the entries of $\vec{v}$ can be written $v^{(i)} = 2^{d(i)} w^{(i)}$, with $|w^{(i)}| \leq 2^{a(i)}$, where $d(i), w^{(i)}$ and $a(i)$, for $i = 1, \ldots, k$, are integers. Suppose that the inner product r of the two vectors is bounded by $|r| + 2^t < 2^h$ and that we seek an approximation r^* such that $|r^* - r| \leq 2^t$. Then, we may take appropriately rounded off moduli of the vector entries, as in algorithm 4.4, and reduce the computation modulo 2^{h+1}, with the following precision: $h + 1 - t + \lceil \log k \rceil + a(i)$ bits in step 2, for each i, and $h + 1 - t + \lceil \log k \rceil$ bits in the rest of the algorithm.*

5. Application of b.b.s. to the iterative improvement algorithm

This section applies b.b.s. in order to bound the precision of computations in the well known algorithm (Atkinson, 1978, Wilkinson, 1965) for iterative improvement of a solution to a nonsingular linear system of equations. First, we consider a generalized version of the algorithm, and then we outline the application of b.b.s. to the classical algorithm.

Hereafter, we will assume the matrix and vector norm $\|.\| = \|.\|_\infty$. Consider the non-singular system of n equations

$$A\vec{x} = \vec{f}.$$

The input to the iterative algorithm consists of a vector $\vec{f}$, some initial vector $\vec{x}(0)$, say $\vec{x}(0) = \vec{0}$, and a pair of $n \times n$ matrices A and C, where the latter approximates A^{-1} so that

$$\|I - CA\| \leq 2^{-b} < 1, \tag{5.1}$$

for some fixed positive scalar b. Then the iterative improvement algorithm successively computes the vectors

$$\vec{r}(p) = \vec{f} - A\vec{x}(p-1),\ \vec{e}(p) = C\vec{r}(p),\ \vec{x}(p) = \vec{x}(p-1) + \vec{e}(p), \quad \text{for } p = 1, 2, \ldots. \tag{5.2}$$

It can be easily shown that, for $p = 1, 2, \ldots$, we have

$$\begin{aligned} \vec{x} - \vec{x}(p) &= (I - CA)(\vec{x} - \vec{x}(p-1)) = (I - CA)^p(\vec{x} - \vec{x}(0)), \\ \vec{r}(p) &= A(\vec{x} - \vec{x}(p-1)), \\ \vec{e}(p) &= CA(\vec{x} - \vec{x}(p-1)). \end{aligned}$$

Therefore, if bound (5.1) holds, then $\vec{r}(p)$ and $\vec{e}(p)$ converge to $\vec{0}$ with the speed of a geometric progression, as $p \to \infty$. It is customary to use the double precision in computing $\vec{r}(p)$ for all p in iteration (5.2) in order to ensure such a rapid convergence; see Golub and Van Loan (1989), pp. 126–127, or Atkinson (1978), pp. 467-471. However, we observe that the precision of computing can be controlled and decreased by means of using the b.b.s. process. Indeed, apart from an addition and a subtraction of two pairs of vectors, the computation of (5.2) amounts to two multiplications of $n \times n$ matrices A and C by vectors, that is, to the evaluation of $2n$ inner products of $2n$ pairs of vectors. Two issues are encountered in decreasing precision:

First, to show that the *convergence* of iteration (5.2) with the speed of geometric progression will be preserved even if the vectors $\vec{r}(p)$ and $\vec{e}(p)$ are replaced by their numerical approximations $\vec{r^*}(p) = \vec{r}(p) + \Delta\vec{r}(p)$, $\vec{e^*}(p) = \vec{e}(p) + \Delta\vec{e}(p)$, for $p = 1, 2, \ldots$. Second, to *bound* the norm $\|A\vec{x^*}(p)\|$ of the vector approximating vector $A\vec{x}(p)$ in this process, for $p = 1, 2, \ldots$.

Routine error analysis shows that $\log \|\Delta\vec{r}(p)\| \leq g^* - bp$ and $\log \|\Delta\vec{e}(p)\| \leq g - bp$, $p = 1, 2, \ldots$, where b is defined in bound (5.1) and g and g^* are some fixed scalars.

LEMMA 5.1. *There exist two fixed scalars h and h^*, independent of p, such that*

$$\log \|\vec{r^*}(p)\| \leq h^* + \lceil \log p \rceil - bp, \quad \log \|\vec{e^*}(p)\| \leq h + \lceil \log p \rceil - bp, \quad p = 1, 2, \ldots.$$

PROOF. Replace $\vec{e}(p)$ by $\vec{e^*}(p)$ and $\vec{r}(p)$ by $\vec{r^*}(p)$ in equations (5.2) and obtain that

$$\begin{aligned} \vec{x} - \vec{x}(p) &= \vec{x} - \vec{x}(p-1) - C\vec{r}(p) - C\Delta\vec{r}(p) - \Delta\vec{e}(p) \\ &= (I - CA)(\vec{x} - \vec{x}(p-1)) - C\Delta\vec{r}(p) - \Delta\vec{e}(p) \\ &= (I - CA)^p(\vec{x} - \vec{x}(0)) - \sum_{i=1}^{p}(I - CA)^{p-i}(C\Delta\vec{r}(i) + \Delta\vec{e}(i)). \end{aligned}$$

The bounds on $\|\Delta\vec{r}(p)\|$ and $\|\Delta\vec{e}(p)\|$ yield

$$\|\vec{x}-\vec{x}(p)\| \leq \|I-CA\|^p\|\vec{x}-\vec{x}(0)\| + \sum_{i=1}^{p}\|I-CA\|^{p-i}(\|C\|2^{g^*}+2^g)2^{-bi}.$$

Write $N = \|C\|2^{g^*}+2^g$, $E_0 = \|\vec{x}-\vec{x}(0)\|$, substitute bound (5.1), and obtain that $\|\vec{x}-\vec{x}(p)\| \leq 2^{-bp}(E_0+pN)$. Now, apply $\vec{r}(p) = A(\vec{x}-\vec{x}(p))$ and $\vec{e}(p) = C\vec{r}(p)+\Delta\vec{r}(p)$ to establish the lemma. Clearly, since h and h^* depend on N and E_0, they also depend on g and g^*. □

The lemma implies rapid convergence of $\|\vec{x}-\vec{x^*}(p)\|$ to 0, with the speed of a geometric progression. This immediately implies that both requirements are satisfied. Note that $\|A\vec{x}\|$ is bounded since A and $\vec{x}$ are fixed, and that $\|A(\vec{x^*}(p)-\vec{x})\| \leq \|A\|\ \|\vec{x^*}(p)-\vec{x}\| = \|A\|\ \|\vec{e^*}(p)\|$ tends to zero with the speed of a geometric progression as p tends to infinity.

Next, we will assume that b, g, g^*, h and h^* have been precomputed, and that every entry a_{ij} of the input matrix A lies in a fixed binary segment $S[g_{ij}(A), h_{ij}(A))$ of moderately small length.

Remark 5.2. *The latter assumption about a_{ij} is needed in order to bound the precision of computing the product of A by $\vec{x}(p-1)$ in iteration (5.2). This assumption holds, for instance, for many linear systems obtained by means of the discretization of linear PDEs with constant coefficients. Generally, if A is a well-conditioned matrix, we may decrease $h_{ij}(A)-g_{ij}(A)$ by chopping the entries of A and/or by applying the standard technique of algebraic segmentation (Aho, Hopcroft and Ullman, 1974, Duhamel, 1986, Nussbaumer, 1980, Winograd, 1980).*

By applying the results of sections 2–4, whose tedious but straightforward elaboration we omit, we arrive at the following bounds.

Corollary 5.3. *Let $\rho_i(W)$ denote the number of nonzero entries of row i of a matrix W and let $\ell_i(A) = \max_j |h_{ij}(A)-g_{ij}(A)|$, where each matrix entry a_{ij} lies in a fixed binary segment of moderately small length, namely $S[g_{ij}(A), h_{ij}(A))$. Then, with the above notation and $p = 1, 2, \ldots$, we can apply b.b.s. in iteration (5.2), implying that it suffices to use the following bit-precision in the corresponding operations:*

$$d_i^*(p) = h^* - g^* + \log p + \lceil \log \rho_i(A) \rceil$$

bits in the representation of each operand in the computation of the i-th entry of $\vec{r}(p)$, $d_i^*(p) + \ell_i(A)$ bits in the representation of each entry of $\vec{x}(p-1)$ when this entry is multiplied by an entry of row i of A,

$$d_i(p) = h - g + \lceil \log p \rceil + \lceil \log \rho_i(C) \rceil$$

bits in the representation of each operand in computing the i-th component of $\vec{e^*}(p)$, and

$$d_i(p) + h^* - g^* + \lceil \log p \rceil$$

bits in the representation of any entry of row i of C when this entry is multiplied by a component of $\vec{r}(p)$.

Thus, the b.b.s. process enables us to compute the solution vector $\vec{x}$, within error norm of the order of 2^{-p}, after p calls to the loop (5.2), even though only an order of $\log i$ bits of precision are needed in the computations in the p-th call for this loop, for $p = 1, 2, \ldots$. In particular, even if $p = 60$ (which is much greater than what is usually needed in practice), then, still $\log p < 6$, whereas over 50 bits of single machine precision are usually allowed on modern computers performing matrix computations.

For comparison, if we perform the iteration based on iteration (5.2) but do not use the b.b.s. process, then we generally must increase the precision of the computations in the first two stages of (5.2). Let c_{ij} denote an entry of matrix C, let $x_j(p-1)$ denote the j-th entry of $\vec{x}(p-1)$ and $r^*_j(p)$ the respective entry of $\vec{r^*}(p)$. Then the required precision must increase at least to $H^*(p) - g(p)$ and $H(p) - g(p)$, where

$$H^*(p) = \log \max \left\{ \|\vec{f}\|,\ \max_{i,j} |a_{ij} x_j(p-1)| \right\}, \quad H(p) = \log \max_{i,j} |c_{ij} r^*_j(p)|,$$

for $i, j = 1, \ldots, n$; $p = 1, 2, \ldots$. This represents an increase of at least $H^* + bp - \log p$ and $\log \max\limits_{i,j} |c_{ij}| + H$ bits, for constants H^* and H.

In the classical iterative improvement algorithm $C = (PLU)^{-1}$ where P is a permutation matrix, L is unit lower triangular, $U = D + U^*$ with U^* proper upper triangular, D is a nonsingular diagonal matrix, and every entry of L has absolute value bounded by 1. In this case, the computation of $\vec{e}(p)$ is replaced by the successive solution of two triangular linear systems of equations:

$$L\vec{y}(p) = P^{-1}\vec{r}(p), \quad U\vec{e}(p) = \vec{y}(p). \tag{5.3}$$

Then we can bound $\|\vec{y}(p)\| = \|U\vec{e}(p)\| \leq 2^{h(p)}\|U\|$. Since we seek $\vec{e}(p)$ within the bound $2^{g(p)}$ on the error vector norm, we only need to compute the components of $\vec{y}(p)$ within the error bound $\|U^{-1}\|2^{g(p)}$, assuming that a sufficiently good upper estimate for $\|U^{-1}\|$ is available. Let $\tilde{h}$ and $\tilde{g}$ be two fixed integers such that $\|U\| \leq 2^{\tilde{h}}$, $\|U^{-1}\| \leq 2^{\tilde{g}}$. Then $\|\vec{y}(p)\| \leq 2^{\tilde{h}+h(p)}$.

Corollary 5.4. *If we apply of the b.b.s. process to the evaluation of every inner product in the classical iterative improvement algorithm, the precision of the computations substantially decreases against the usual solution except possibly for small values of p. In particular, it suffices to use the following bit-precision in the corresponding operations:* $h - g + \tilde{h} - \tilde{g} + \lceil \log p \rceil + \lceil \log \rho_i(L) \rceil$ *bits in the representation of each operand of any addition or subtraction involved in the evaluation of the i-th component of the vector* $\vec{y}(p)$*, and* $2(h - g + \tilde{h} - \tilde{g} + \lceil \log p \rceil) + \lceil \log \rho_i(L) \rceil$ *bits in the representation of any entry of row i of L when this entry is multiplied by a component of* $\vec{y}(p)$*,* $h - g + \lceil \log p \rceil + \lceil \log \rho_i(U) \rceil$ *bits in the representation of each operand of any addition or subtraction involved in the evaluation of the i-th component of the vector* $\vec{e^*}(p)$*,* $2(h - g + \lceil \log p \rceil) + \lceil \log \rho_i(U) \rceil$ *bits in the representation of any entry of row i of the matrix* $D^{-1}U^*$ *when this entry is multiplied by the components of* $\vec{e}(p)$*.*

6. Two further applications of the b.b.s. process

This section presents two further examples of the b.b.s. process, applied to important linear algebra problems. The first example considers linear systems whose matrix is real and symmetric; the analysis and results can be easily extended to several other well-known iterative techniques, such as Jacobi's, SOR and SSOR (Varga, 1962, Young 1971). The second example is on piecewise linear PDEs solved by multigrid methods.

Let A denote a real symmetric matrix filled with "short" binary numbers so that $A = L + I + U$, with $L = U^T$ being a proper lower triangular matrix. Gauss-Seidel's iteration for $A\vec{x} = \vec{f}$ takes the following form (Golub and Van Loan, 1989, Isaacson and Keller, 1966).

$$\vec{x}(p+1) = \vec{f} - L\vec{x}(p+1) - U\vec{x}(p), \quad p = 0, 1, \dots.$$

If $\vec{x}(p) = \sum_{i=0}^{p} \Delta\vec{x}(i)$, for $p = 0, 1, \dots$, then $\Delta\vec{x}(p+1) = -L\Delta\vec{x}(p+1) - U\Delta\vec{x}(p)$. The iteration converges to the solution if and only if A is positive definite (Isaacson and Keller, 1966, pp. 70–71, Golub and Van Loan, 1989, p. 509). In this case, $\|\Delta\vec{x}(p)\| < 3^{g-bp}$, $p = 0, 1, \dots$, where g is a fixed constant and 2^{-b} is the spectral radius of the matrix $B = (L+I)^{-1}U$, $2^{-b} \leq \| B \|$, $b > 0$. Estimating b generally takes a substantial amount of work, but for some important classes of the input matrix, a good positive lower bound on b is readily available. Then due to the rapid decrease of the error norm $\|\Delta\vec{x}(p)\|$, an application of the b.b.s. process enables us to decrease the precision of the computations.

The second application refers to solving differential equations. In particular, we are able to apply the b.b.s. technique in the solution of piecewise linear partial differential equations (PDEs) by means of multigrid methods (compare Pan and Reif (1992), Pan and Reif (1993)).

Let us show this, by outlining the multigrid approach and by observing its similarity to the iterations of section 5 and above. For a given PDE and for a fixed sequence of d-dimensional grids $G_0 \subset G_1 \subset G_1 \ldots \subset G_n$, define $n+1$ linear systems of difference equations,

$$D_i\vec{u}_i = \vec{b}_i, \tag{6.1}$$

by discretizing the PDE over the grids G_i, $i = 0, 1, \cdots, n$. Vector $\vec{u}_i$ that satisfies equation (6.1) approximates the solution to the given PDE on the grid G_i, hence its dimension equals the number of vertices on the grid G_i, for $i = 0, 1, \dots, n$. Let $v_i(\vec{x})$ for $\vec{x} \in G_i$ denote the respective component of any vector $\vec{v}_i$ defined on G_i. Define the operators P_i of prolongation of $u_{i-1}(\vec{x})$ from G_{i-1} to G_i (such operators usually amount to interpolation by averaging).

We now recall the customary loop (V-cycle) of the multigrid algorithm for solving system (6.1), for $i = n$. Starting with, say, $u_0(\vec{x}) = 0$ for $\vec{x} \in G_0$, we successively evaluate, for $i = 1, \dots, n$ and all $\vec{x} \in G_i$, the following values. First, we compute $\vec{r}_i = \vec{b}_i - D_iP_i\vec{u}_{i-1}$, then we find $\vec{e}_i$ by solving linear system $D_i\vec{e}_i = \vec{r}_i$, and, finally, we compute $u_i(\vec{x})$ from identity $e_i(\vec{x}) = u_i(\vec{x}) - P_iu_{i-1}(\vec{x})$.

In the case of a piecewise-linear PDE with constant coefficients, the entries of the matrix D_i are "short" binary values, each represented with $O(1)$ bits. The only difference with the usual application of the iterative improvement scheme is in solving the linear system by means of iterative methods (say, of Gauss-Seidel's or of SSOR type in the symmetric case). Furthermore, the number of iterations required in order to solve this linear system is typically bounded from above by a fixed constant, which corresponds to setting $p = O(1)$ in section 5. Thus, by applying techniques exemplified earlier we decrease the precision of these computation to $O(1)$ bits.

7. Numerical tests

In this section we present the results of some numerical experiments designed in order to compare the performance of two implementations of the generalized algorithm of section 5 for the iterative improvement of the solution of a linear system of equations. That is, we tested a customary implementation and one using the b.m.r./b.b.s. techniques.

We have run our experiments on a general purpose computer that relies on fixed precision representation of floating point numbers and uses floating point hardware logic for acceleration of numerical computations. Since the b.m.r./b.b.s. techniques rely on using variable precision representation of numbers, we could not directly compare the CPU time, executable size or run-time memory consumption of the b.m.r./b.b.s. algorithm with that of the customary algorithm. Instead, we emulated both algorithms with a high level language using special data structures and then approximately measured the bit-complexity as follows.

Definition 7.1. *For floating point numbers r_1 and r_2 represented with the precision of p_1 and p_2 bits, respectively, we define the bit-complexity of their addition c^+ and multiplication c^*, as follows.*

$$c^+(r_1, r_2) = \max\{p_1, p_2\}, \quad c^*(r_1, r_2) = p_1 \times p_2.$$

In our experiments, the input to the algorithm consists of an $n \times n$ matrix A, a matrix C that approximates A^{-1}, an n-dimensional vector $\vec{f}$, and an error bound $\epsilon > 0$. The program calculates and outputs $\vec{x}(p)$ such that $||\vec{x}(p) - \vec{x}|| < \epsilon$, where $\vec{x}$ is the solution of the linear system $A\vec{x} = \vec{f}$.

We have implemented the algorithm in the ANSI C language and the program was compiled, linked and run on a SUN Sparc station running SUN OS version 4.

We store floating point numbers in a C structure consisting of a sign, a mantissa, an exponent, and a precision value. All the input, output and intermediate results have been stored in this format, and all the arithmetic operations needed for the experiments (such as addition, multiplication and modular reduction) have been implemented with C functions. The approximate inverse matrices have been calculated by using PLU decomposition with partial pivoting.

In the remainder of this section, we show only the input and output of our experiments, with complexity estimates based on the above definition. These results confirm the theory by showing a consistent decrease of the bit-complexity in the transition from the customary implementation to the b.m.r./b.b.s. implementation. In particular, we report a decrease of 6–17%. This is shown in the respective tables by the ratio of the additive and multiplicative complexities, respectively, between the two algorithms. We use the decimal representation for the sake of clarity.

Example 7.2. This example is from Dahlquist and Björck (1974), pp. 184–185, and has input:

$$\begin{aligned}
\epsilon \quad &= +(0.10)_{10} \times 10^{-4}, \\
A \quad &= \begin{pmatrix} +(0.20000)_{10} \times 10^0 & +(0.16667)_{10} \times 10^0 & +(0.14286)_{10} \times 10^0 \\ +(0.16667)_{10} \times 10^0 & +(0.14286)_{10} \times 10^0 & +(0.12500)_{10} \times 10^0 \\ +(0.14286)_{10} \times 10^0 & +(0.12500)_{10} \times 10^0 & +(0.11111)_{10} \times 10^0 \end{pmatrix}, \\
C \quad &= \begin{pmatrix} +(0.21618)_{10} \times 10^4 & -(0.57597)_{10} \times 10^4 & +(0.37001)_{10} \times 10^4 \\ -(0.57597)_{10} \times 10^4 & +(0.15793)_{10} \times 10^5 & -(0.10362)_{10} \times 10^5 \\ +(0.37001)_{10} \times 10^4 & -(0.10362)_{10} \times 10^5 & +(0.69087)_{10} \times 10^4 \end{pmatrix}, \\
\vec{f} \quad &= (\, +(0.50953)_{10} \times 10^0 \quad +(0.43453)_{10} \times 10^0 \quad +(0.37897)_{10} \times 10^0 \,)^T.
\end{aligned}$$

Table 1. Complexity of example 7.2

iteration	customary		b.m.r/b.b.s.	
i	c_i^+	c_i^*	c_i^+	c_i^*
1	285	1575	261	1350
2	510	2925	453	2475
3	735	4275	629	3600
4	960	5625	802	4725
total	2490	14400	2145	12150
ratio	1	1	0.86	0.84

The initial and final approximations to $\vec{x}$ are

$$\begin{aligned} \vec{x}(0) &= (\,+(0.90000)_{10} \times 10^0 \quad +(0.90000)_{10} \times 10^0 \quad +(0.90000)_{10} \times 10^0\,)^T , \\ \vec{x}(4) &= (\,+(0.99999)_{10} \times 10^0 \quad +(0.99999)_{10} \times 10^0 \quad +(0.99999)_{10} \times 10^0\,)^T . \end{aligned}$$

Both the customary as well as the b.m.r./b.b.s. algorithms produce the same output after 4 iterations. However, the second method has lower additive and multiplicative complexity at every iteration, as shown in table 1.

Example 7.3. This example is from Johnston (1982), p. 52, and has input:

$$\begin{aligned} \epsilon &= +(0.10)_{10} \times 10^{-3}, \\ A &= \begin{pmatrix} +(0.1230)_{10} \times 10^1 & +(0.4560)_{10} \times 10^1 & +(0.9870)_{10} \times 10^1 \\ -(0.9610)_{10} \times 10^1 & +(0.6020)_{10} \times 10^1 & +(0.1110)_{10} \times 10^2 \\ +(0.7310)_{10} \times 10^1 & +(0.2890)_{10} \times 10^1 & +(0.5040)_{10} \times 10^1 \end{pmatrix} , \\ C &= \begin{pmatrix} +(0.1452)_{10} \times 10^{-1} & -(0.4629)_{10} \times 10^{-1} & +(0.7351)_{10} \times 10^{-1} \\ -(0.1082)_{10} \times 10^1 & +(0.5508)_{10} \times 10^0 & +(0.9062)_{10} \times 10^0 \\ +(0.5995)_{10} \times 10^0 & -(0.2487)_{10} \times 10^0 & -(0.4278)_{10} \times 10^0 \end{pmatrix} , \\ \vec{f} &= (\,+(0.4120)_{10} \times 10^1 \quad +(0.5340)_{10} \times 10^1 \quad -(0.3560)_{10} \times 10^1\,)^T . \end{aligned}$$

The initial and final approximations to $\vec{x}$ are

$$\begin{aligned} \vec{x}(0) &= (\,-(0.4491)_{10} \times 10^0 \quad -(0.4743)_{10} \times 10^1 \quad +(0.2665)_{10} \times 10^1\,)^T , \\ \vec{x}(4) &= (\,-(0.4490)_{10} \times 10^0 \quad -(0.4743)_{10} \times 10^1 \quad +(0.2665)_{10} \times 10^1\,)^T , \end{aligned}$$

where the same output is obtained by both customary and b.m.r./b.b.s. algorithms. Again, the second algorithm is faster at every iteration, as seen in table 2.

Example 7.4. This example is from Golub and Van Loan (1989), p. 147, and has the input:

$$\begin{aligned} \epsilon &= +(0.10)_{10} \times 10^{-2}, \\ A &= \begin{pmatrix} +(0.986)_{10} \times 10^3 & +(0.579)_{10} \times 10^3 \\ +(0.409)_{10} \times 10^3 & +(0.237)_{10} \times 10^3 \end{pmatrix} , \\ C &= \begin{pmatrix} -(0.757)_{10} \times 10^{-1} & +(0.185)_{10} \times 10^0 \\ +(0.131)_{10} \times 10^0 & -(0.315)_{10} \times 10^0 \end{pmatrix} , \end{aligned}$$

Table 2. Complexity of example 7.3

iteration	customary		b.m.r/b.b.s.	
i	c_i^+	c_i^*	c_i^+	c_i^*
1	228	1008	228	864
2	408	1872	379	1584
3	588	2736	530	2304
4	768	3600	681	3024
total	1992	9216	1818	7776
ratio	1	1	0.91	0.84

Table 3. Complexity of example 7.4

iteration	customary		b.m.r/b.b.s.	
i	c_i^+	c_i^*	c_i^+	c_i^*
1	90	252	90	216
2	162	468	152	396
3	234	684	202	556
total	486	1404	444	1168
ratio	1	1	0.91	0.83

$$\vec{f} \quad = (\,+(0.235)_{10} \times 10^3 \qquad +(0.107)_{10} \times 10^3\,)^T\,.$$

The initial and final approximations to solution $\vec{x}$ are:

$$\begin{aligned}
\vec{x}(0) &= (\,+(0.200)_{10} \times 10^1 \qquad -(0.290)_{10} \times 10^1\,)^T\,,\\
\vec{x}(3) &= (\,+(0.200)_{10} \times 10^1 \qquad -(0.300)_{10} \times 10^1\,)^T\,,
\end{aligned}$$

where the same output is obtained by both algorithms. The savings due to the b.m.r./b.b.s. algorithm are reported in table 3.

Example 7.5. Here is a random input.

$$\begin{aligned}
\epsilon \quad &= +(0.10)_{10} \times 10^{-2},\\
A \quad &= \begin{pmatrix} +(0.6000)_{10} \times 10^1 & +(0.1100)_{10} \times 10^2 & +(0.2000)_{10} \times 10^2 \\ +(0.3500)_{10} \times 10^2 & +(0.6800)_{10} \times 10^2 & +(0.1330)_{10} \times 10^3 \\ +(0.2600)_{10} \times 10^3 & +(0.5170)_{10} \times 10^3 & +(0.1030)_{10} \times 10^4 \end{pmatrix},\\
C \quad &= \begin{pmatrix} -(0.6525)_{10} \times 10^1 & +(0.5051)_{10} \times 10^1 & -(0.5255)_{10} \times 10^0 \\ +(0.7500)_{10} \times 10^1 & -(0.5000)_{10} \times 10^1 & +(0.5000)_{10} \times 10^0 \\ -(0.2117)_{10} \times 10^1 & +(0.1235)_{10} \times 10^1 & -(0.1173)_{10} \times 10^0 \end{pmatrix},\\
\vec{f} \quad &= (\,-(0.1440)_{10} \times 10^1 \qquad +(0.7523)_{10} \times 10^1 \qquad -(0.3392)_{10} \times 10^0\,)^T\,.
\end{aligned}$$

The initial and final approximations to solution $\vec{x}$ are as follows, obtained by the cus-

Table 4. Complexity of example 7.5

iteration	customary		b.m.r/b.b.s.	
i	c_i^+	c_i^*	c_i^+	c_i^*
1	230	1056	230	912
2	411	1968	392	1680
3	592	2880	555	2448
4	773	3792	709	3171
total	2006	9696	1886	8211
ratio	1	1	0.94	0.85

tomary and b.m.r./b.b.s. algorithms respectively:

$$\begin{aligned}
\vec{x}(0) &= (\,+(0.4758)_{10}\times 10^2 \qquad -(0.4859)_{10}\times 10^2 \qquad +(0.12379)_{10}\times 10^2\,)^T\,,\\
\vec{x}(4) &= (\,+(0.4757)_{10}\times 10^2 \qquad -(0.4858)_{10}\times 10^2 \qquad +(0.12377)_{10}\times 10^2\,)^T\,,\\
\vec{x}(4) &= (\,+(0.4757)_{10}\times 10^2 \qquad -(0.4858)_{10}\times 10^2 \qquad +(0.12381)_{10}\times 10^2\,)^T\,.
\end{aligned}$$

Note that the two outputs are different, though within the same error bound. The complexities do differ, as seen in table 4.

8. Further work

An immediate extension of this work would be to computing products of k binary rationals, when a bound is known on the product. However, it is not obvious how to ignore the most significant digits in this case.

Clearly, the main issue is to extend applicability of the method, in other words obtain tight bounds on the size of the answer. A general technique that would circumvent this question may be based on probabilistic methods used in exact modular arithmetic on the rationals, such as those by Emiris (1998).

Acknowledgments

Some anonymous referees, the area editor Richard Zippel, and Franck Avitabile have made several helpful comments.

References

K.E. Atkinson (1978). *Introduction to Numerical Analysis*, Wiley, New York.

A. V. Aho, J. E. Hopcroft, J. D. Ullman (1974). *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, Massachusetts.

D. Bini, V. Y. Pan (1994). *Polynomial and Matrix Computations*, volume 1: *Fundamental Algorithms*, Birkhäuser, Boston, Massachusetts.

H. Brönnimann, I.Z. Emiris, V. Pan, S. Pion (1997). Computing Exact Geometric Predicates Using Modular Arithmetic With Single Precision. In *Proc. ACM Symp. on Computational Geometry*, Nice, pp. 174–182.

S.D. Conte, C. de Boor (1980). *Elementary Numerical Analysis: An Algorithmic Approach*, McGraw Hill.

J.H. Davenport, Y. Siret, E. Tournier (1988). *Computer Algebra*, Academic Press, London.

P. Duhamel (1986). Implementation of "Split Radix" FFT Algorithms for Complex, Real and Real Symmetric Data, *IEEE Trans. Acoust., Speech, Signal Processing*, 34, pp. 285–295.

G. Dahlquist, Å. Björck (1974). *Numerical Methods*, Prentice-Hall, Englewood Cliffs, New Jersey.

I.Z. Emiris (1998). A Complete Implementation for Computing General Dimensional Convex Hulls, *Intern. J. Computational Geometry & Applications.* To appear. A preliminary version in Tech. Report 2551, INRIA Sophia-Antipolis, France, 1995.

R.T. Gregory (1980). *Error-Free Computation: Why It Is Needed and Methods for Doing It*, Krieger, New York.

G.H. Golub, C.F. Van Loan (1989). *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland.

N.J. Higham (1996). *Accuracy and Stability of Numerical Algorithms*, SIAM Publications, Philadelphia, 1996.

E. Isaacson, H. Keller (1966). *Analysis of Numerical Methods*, Wiley, New York.

R.L. Johnston (1982). *Numerical Methods : A Software Approach*, Wiley, New York.

H.J. Nussbaumer (1980). Fast Polynomial Transform Algorithms for Digital Convolution, *IEEE Trans. on Acoust., Speech, Signal Processing*, 28, 2, pp. 205–215.

V.Y. Pan (1991). Complexity of Algorithms for Linear Systems of Equations, *in Computer Algorithms for Solving Linear Algebraic Equations, The State of the Art*, edited by E. Spedicato, NATO ASI Series, Series F: Computer and Systems Sciences, 77, pp.27–56, Springer, Berlin.

V.Y. Pan (1992). Complexity of Computations with Matrices and Polynomials, *SIAM Review.,* 34, 2, pp. 225–262.

V.Y. Pan (1992a). Can We Utilize the Cancelation of the Most Significant Digits?, Tech. Report TR-92-061, *The International Computer Science Institute,* Berkeley, California.

V.Y. Pan (1993). On Binary Segmentation for Matrix and Vector Operations, *Computer and Math. (with Applications),* 25, 3, pp. 69–71.

V.Y. Pan, J. Reif (1992). Compact Multigrid, *SIAM J. on Sci. and Statist. Comp.,* 13, 1, pp. 119–127.

V.Y. Pan, J. Reif (1993). Generalized Compact Multigrid, *Computer and Math. (with Applications),* 25, 9, pp. 3–5.

R. Varga (1962). *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey.

J.M. Wilkinson (1965). *Algebraic Eigenvalue Problem,* Clarendon Press, Oxford.

S. Winograd (1980). *Arithmetic Complexity of Computations*, SIAM, Philadelphia.

D. Young (1971). *Iterative Solutions of Large Linear Systems*, Academic Press, New York.