BIT 39(3), pp. 539–560, 1999

FAST AND PARALLEL INTERVAL ARITHMETIC

SIEGFRIED M. RUMP

Inst. of Computer Science III, Technical University Hamburg-Harburg, Eißendorfer Str. 38, 21071 Hamburg, Germany. rump@tu-harburg.de

Abstract. Infimum-supremum interval arithmetic is widely used because of ease of implementation and narrow results. In this note we show that the overestimation of midpoint-radius interval arithmetic compared to power set operations is uniformly bounded by a factor 1.5 in radius. This is true for the four basic operations as well as for vector and matrix operations, over real and over complex numbers. Moreover, we describe an implementation of midpoint-radius interval arithmetic entirely using BLAS. Therefore, in particular, matrix operations are *very fast* on almost any computer, with minimal effort for the implementation. Especially, with the new definition it is seemingly the first time that full advantage can be taken of the speed of vector and parallel architectures. The algorithms have been implemented in the Matlab interval toolbox INTLAB.

AMS subject classifications. 65G10.

Key words. Interval arithmetic, parallel computer, BLAS, midpoint-radius, infimum-supremum

1. Introduction and notation.

Let $\mathbb{R}$ denote the set of real numbers, let $\mathbb{C}$ denote the set of complex numbers, and denote the power set over one of these sets by $\mathbb{PR}$, $\mathbb{PC}$, respectively. The two most frequently used representations for intervals over $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$, are the infimum-supremum representation

$$(1) \qquad [a_1, a_2] := \{x \in \mathbb{K} \,:\, a_1 \le x \le a_2\} \quad \text{for some} \quad a_1, a_2 \in \mathbb{K},\, a_1 \le a_2\,,$$

where $\le$ is the partial ordering $x \le y :\Leftrightarrow \operatorname{Re} x < \operatorname{Re} y \;\&\; \operatorname{Im} x < \operatorname{Im} y$ for $x, y \in \mathbb{C}$, and the midpoint-radius representation

$$(2) \qquad < a, \alpha > := \{x \in \mathbb{K} \,:\, |x - a| \le \alpha\} \quad \text{for some } a \in \mathbb{K},\, 0 \le \alpha \in \mathbb{R}\,.$$

The two representations are identical for real intervals, whereas for complex intervals the first representation are rectangles, the second one represents discs in the complex plane. We assume the reader to be familiar with the standard definitions of interval arithmetic for the first and for the second representation [1, 7, 19, 21]. Recent literature on midpoint-radius or circular arithmetic includes [23] and the many papers cited there.

Although the midpoint-radius representation seems to be more appropriate for the complex case, today mostly the infimum-supremum arithmetic is used. There are two main reasons for that. First, the standard definition of midpoint-radius arithmetic causes overestimation for multiplication and division, and second, the computed midpoint of the floating point result of an operation is, in general, not exactly representable in floating point, thus again causing overestimation and additional computational effort.

In this note we will show that the overestimation of operations using midpoint-radius representation compared to the result of the corresponding power set operation is limited by at most a factor 1.5 in radius. As has been mentioned by Prashant Batra to the author, this is already included in [15] for complex intervals, where an area-optimal complex circular arithmetic is introduced. Unfortunately, the computation of midpoint and radius for this type of arithmetic requires the (verified) computation of the roots of a cubic polynomial.

The main point of this paper is to show that the implementation of midpoint-radius arithmetic is very simple, and that on today's computer architectures it allows a much faster code than traditional infimum-supremum arithmetic. This includes in particular vector and parallel computers.

Throughout the paper we refer by $\mathbb{IK}$ *to intervals in midpoint-radius representation* (2), $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$. Operations in $\mathbb{IK}$ are denoted by $\oplus, \ominus, \odot, \oslash$. The same symbols are used for operations on interval vectors $\mathbb{IK}^n$ and interval matrices $\mathbb{IK}^{m\times n}$, and operations among those. The corresponding power set operations are denoted by $+, -, \cdot, /$.

Interval operations always satisfy the fundamental property of isotonicity:

$$(3) \qquad \forall a \in A \quad \forall b \in B : \quad a \circ b \subseteq A \circledcirc B \quad \text{for} \quad \circ \in \{+, -, \cdot, /\}$$

and all suitable A, B. Any representation and implementation of interval arithmetic must obey isotonicity. The radius of a compact set $X \subseteq \mathbb{C}$ is defined by

$$\mathrm{rad}(X) := 0.5 \cdot \max_{x_1, x_2 \in X} |x_1 - x_2| \, .$$

The overestimation of an interval operation with respect to the corresponding power set operation is

$$(4) \qquad \rho := \frac{\mathrm{rad}(A \circledcirc B)}{\mathrm{rad}(A \circ B)} \quad \text{for} \quad \circ \in \{+, -, \cdot, /\} \, .$$

For interval vectors or matrices, ρ is the maximum of the entrywise overestimations. The numbers ρ will be our measure of overestimation. We mention that in algorithms for “global verification” (for example, global optimization or inclusion of *all* zeros of a system of nonlinear equations) the *exclusion* of zeros is the main problem. In this case a fairly small overestimation may lead to additional bisections, and therefore any ratio might be inappropriate to compare midpoint-radius and infimum-supremum arithmetic.

The paper is organized as follows. In the following section we prove that the overestimation (4) of midpoint-radius arithmetic to power set operations is uniformly bounded by 1.5 for the basic arithmetic operations as well as for vector and matrix operations over $\mathbb{R}$ and $\mathbb{C}$. For intervals of not too large radius the factor is quantified to be near 1. In Section 3 we present corresponding algorithms over floating point numbers, where special care has been taken for the error estimation of the midpoint. In Section 4 performance issues are addressed and it is shown that midpoint-radius arithmetic allows very easy and very fast implementation using BLAS [18, 6, 5, 2]. To the author’s knowledge, this is the first implementation of interval arithmetic exclusively using BLAS. Timing comparisons on a Convex SPP 2000 parallel computer are given.

In Section 5 computational evidence shows that the factor 1.5 for the individual operation does not perpetuate when composing operations. In fact, the factor in radius against infimum-supremum arithmetic is around 1.0, and, surprisingly, sometimes even less than 1.

2. Overestimation by midpoint-radius representation.

In this section we first consider real intervals

$$\begin{aligned} A &= <a, \alpha> = \{x \in \mathbb{R} \,:\, a - \alpha \le x \le a + \alpha\} \quad \text{for } a \in \mathbb{R},\ 0 \le \alpha \in \mathbb{R} \quad \text{and} \\ B &= <b, \beta> = \{x \in \mathbb{R} \,:\, b - \beta \le x \le b + \beta\} \quad \text{for } b \in \mathbb{R},\ 0 \le \beta \in \mathbb{R} \, . \end{aligned}$$

The basic operations are defined as usual ([19], [1], [21]):

Definition 2.1. *For* $A = <a, \alpha> \in \mathbb{IR}$, $B = <b, \beta> \in \mathbb{IR}$, *we define*

$$A \oplus B := <a + b,\ \alpha + \beta>$$

$$\begin{aligned} & A \ominus B := < a - b,\ \alpha + \beta > \\ (5) \qquad & A \odot B := < a \cdot b,\ |a|\beta + \alpha|b| + \alpha\beta > \\ & 1 \oslash B := < b/D,\ \beta/D > \quad \textit{where } D := b^2 - \beta^2 \quad \textit{and } 0 \notin B \\ & A \oslash B := A \odot (1 \oslash B) \quad \textit{for } 0 \notin B\ . \end{aligned}$$

All operations satisfy the isotonicity property (3). The following theorem is well known and easily verified.

THEOREM 2.2. *For all $A, B \in \mathbb{IR}$, the result of addition, subtraction and inversion as defined in* (5) *and the result of the corresponding power set operation is identical:*

$$\begin{aligned} A \oplus B &= A + B \\ A \ominus B &= A - B \quad \textit{and} \\ 1 \oslash B &= 1/B\ . \end{aligned}$$

The remaining problem is multiplication. To derive proper bounds we use the following definition.

DEFINITION 2.3. *A real interval $A =< a, \alpha >$ not containing 0 is said to be of relative precision e, $0 \le e \in \mathbb{R}$, if*

$$\alpha \le e \cdot |a|\ .$$

A real interval containing 0 is said to be of relative precision 1.

The wording is justified because for an interval A of relative precision e, the relative error of any $\tilde{a} \in A$ with respect to the midpoint or to one of the endpoints of A is at most e, respectively. By definition, it is always $0 \le e \le 1$.

The following theorem estimates the overestimation of the interval multiplication as defined in (5) with respect to the result of the power set operation.

THEOREM 2.4. *Let $A =< a, \alpha > \in \mathbb{IR}$, $B =< b, \beta > \in \mathbb{IR}$, and denote the overestimation of $A \odot B$ as defined by* (5) *with respect to $A \cdot B$ by*

$$\rho := \frac{rad(A \odot B)}{rad(A \cdot B)}\ ,$$

where $0/0$ is interpreted as 1. Then the following is true.

1) It is always

$$\rho \le 1.5\ .$$

2) If one of the intervals A and B is of relative precision e, then

$$\rho \le 1 + \frac{e}{1+e}\ .$$

3) If interval A is of relative precision e, and B is of relative precision f, then

$$\rho \le 1 + \frac{ef}{e+f}.$$

All estimations are sharp.

Proof. Let intervals $A, B \in \mathbb{IR}$ be given. By a straightforward continuity argument we may assume a, b, α and β to be nonzero. Using $A \odot (-B) = (-A) \odot B = -(A \odot B)$ we have to distinguish three cases. In any case, by definition (5),

$$\text{rad}(A \odot B) = |a|\beta + \alpha|b| + \alpha\beta\ . \qquad (6)$$

i) Assume $A \geq 0$ and $B \geq 0$, and $0 < \alpha \leq e \cdot a$, $0 < \beta \leq f \cdot b$. Then by the usual case distinctions,

$$A \cdot B = \{\tilde{a}\tilde{b} \,:\, \tilde{a} \in A, \tilde{b} \in B\} = \{x \in \mathbb{R} \,:\, (a-\alpha)(b-\beta) \leq x \leq (a+\alpha)(b+\beta)\}\,.$$

Therefore,

$$\mathrm{rad}(A \cdot B) = a\beta + \alpha b\,,$$

and by (6),

$$\frac{\mathrm{rad}(A \odot B)}{\mathrm{rad}(A \cdot B)} = 1 + \frac{\alpha\beta}{a\beta + \alpha b} = 1 + \frac{1}{\frac{a}{\alpha} + \frac{b}{\beta}} \leq 1 + \frac{ef}{e+f}\,.$$

ii) Assume $A \geq 0$, $0 \in B$, and $0 < \alpha \leq e \cdot a$. Then

$$A \cdot B = \{x \in \mathbb{R} \mid (a+\alpha)(b-\beta) \leq x \leq (a+\alpha)(b+\beta)\}\,.$$

Therefore,

$$\mathrm{rad}(A \cdot B) = a\beta + \alpha\beta\,,$$

and using $|b| \leq \beta$ and (6),

$$\frac{\mathrm{rad}(A \odot B)}{\mathrm{rad}(A \cdot B)} = 1 + \frac{\alpha|b|}{a\beta + \alpha\beta} \leq 1 + \frac{1}{\frac{a}{\alpha} + 1} \leq 1 + \frac{e}{1+e}\,.$$

iii) Assume $0 \in A$, and $0 \in B$. Then the lower bound of $A \cdot B$ is either $(a+\alpha)(b-\beta)$ or $(a-\alpha)(b+\beta)$, and the upper bound of $A \cdot B$ is either $(a-\alpha)(b-\beta)$ or $(a+\alpha)(b+\beta)$. Using again $A \odot (-B) = (-A) \odot B = -(A \odot B)$ reduces the analysis to two cases. Suppose

$$A \cdot B = \{x \in \mathbb{R} \,:\, (a+\alpha)(b-\beta) \leq x \leq (a-\alpha)(b-\beta)\}\,,$$

then

$$(a+\alpha)(b-\beta) \leq (a-\alpha)(b+\beta) \quad \text{and} \quad (a+\alpha)(b+\beta) \leq (a-\alpha)(b-\beta)\,,$$

which implies $\alpha b \leq a\beta$ and $\alpha b \leq -a\beta$. Using $\alpha \geq 0$ it follows $b \leq 0$ and $|a|\beta \leq \alpha|b|$. Hence, using $\beta \geq |b|$ and (6),

$$\frac{\mathrm{rad}(A \odot B)}{\mathrm{rad}(A \cdot B)} = \frac{|a|\beta + \alpha|b| + \alpha\beta}{-\alpha b + \alpha\beta} \leq 1 + \frac{\alpha|b|}{\alpha|b| + \alpha\beta} \leq 1.5\,.$$

On the other hand, suppose

$$A \cdot B = \{x \in \mathbb{R} \,:\, (a+\alpha)(b-\beta) \leq x \leq (a+\alpha)(b+\beta)\}\,.$$

Then

$$(a+\alpha)(b-\beta) \leq (a-\alpha)(b+\beta) \quad \text{and} \quad (a-\alpha)(b-\beta) \leq (a+\alpha)(b+\beta),$$

which implies $\alpha b \leq a\beta$ and $-\alpha b \leq a\beta$, and using $\beta \geq 0$,

$$a \geq 0 \quad \text{and} \quad \alpha|b| \leq a\beta\,.$$

Hence, using $\alpha \geq |a|$ and (6),

$$\frac{\operatorname{rad}(A \odot B)}{\operatorname{rad}(A \cdot B)} = \frac{|a|\beta + \alpha|b| + \alpha\beta}{a\beta + \alpha\beta} \leq 1 + \frac{a\beta}{a\beta + \alpha\beta} \leq 1.5\,.$$

Finally, define $A := < 1, e >$, and $B := < 1, f >$. Then setting $e = f = 1$, setting $e < 1$ and $f = 1$, and setting $e < 1$ and $f < 1$, respectively, shows that all three inequalities are sharp. $\square$

Next, consider the complex case. For $a, b \in \mathbb{C}$, $0 \leq \alpha$, $\beta \in \mathbb{R}$ and complex intervals

$$\begin{array}{lrcccl} (7) & A & = & < a, \alpha > & = & \{z \in \mathbb{C} \,:\, |z - a| \leq \alpha\} \\ & B & = & < b, \beta > & = & \{z \in \mathbb{C} \,:\, |z - b| \leq \beta\}\,, \end{array}$$

we use again the symbols $\oplus, \ominus, \odot, \oslash$ for midpoint-radius interval operations. The elementary operations are defined as usual ([1],[19], [21]).

Definition 2.5. *For $A, B \in \mathbb{IC}$ as in* (7), *we define*

$$\begin{array}{lrcl} & A \oplus B & := & < a + b,\, \alpha + \beta > \\ & A \ominus B & := & < a - b,\, \alpha + \beta > \\ (8) & A \odot B & := & < a \cdot b,\, |a|\beta + \alpha|b| + \alpha\beta > \\ & 1 \oslash B & := & < \bar{b}/D, \beta/D > \quad \textit{where } D := b\bar{b} - \beta^2 \quad \textit{and } 0 \notin B \\ & A \oslash B & := & A \odot (1 \oslash B) \quad \textit{for } 0 \notin B\,. \end{array}$$

As in the real case, for all $A, B \in \mathbb{IC}$ the result of addition, subtraction or inversion as defined in (8) and the result of the corresponding power set operation is identical. We extend the definition of relative precision to complex intervals.

Definition 2.6. *A complex interval $A =< a, \alpha >$, $a \in \mathbb{C}$, $0 \leq \alpha \in \mathbb{R}$, not containing 0 is said to be of relative precision e, $0 \leq e \in \mathbb{R}$, if*

$$\alpha \leq e \cdot |a|\,.$$

A complex interval containing 0 is said to be of relative precision 1.

For the analysis of overestimation of complex multiplication and division, observe

$$e^{i\varphi} \cdot A = \left\{e^{i\varphi} \cdot z \,:\, |z - a| \leq \alpha\right\} = \left\{z \,:\, |z - e^{i\varphi}a| \leq \alpha\right\} =< e^{i\varphi}a, \alpha >$$

for any $\varphi \in \mathbb{R}$, thus

$$\underline{\operatorname{rad}(e^{i\varphi} \cdot A) = \operatorname{rad}(A)}\,.$$

For $A =< a, \alpha >$, $B =< b, \beta >$ and $a = r_1 e^{i\varphi_1}$, $b = r_2 e^{i\varphi_2}$, $\operatorname{rad}(A \cdot B) = \operatorname{rad}(A' \cdot B')$ with $A' := e^{-i\varphi_1}A$, $B' := e^{-i\varphi_2}B$, where A' and B' have a real midpoint. Carefully going through the proof of Theorem 2.4 shows that all the estimations in Theorem 2.4 hold, *mutatis mutandis*, for complex intervals as well.

Furthermore, for matrix and vector operations we observe that the components are computed by scalar products, which are sums of products. Since the addition does not cause overestimation at all, neither in the real nor in the complex case, the estimations in Theorem 2.4 can be generalized to real and complex vector and matrix operations as well.

Summarizing we obtain the following proposition.

Proposition 2.7. *The midpoint-radius interval operations defined by* (5) *and* (8) *cause the following overestimations compared to the corresponding power set operation:*

i) For A and B being real or complex intervals, midpoint-radius interval addition, subtraction and inversion does not cause any overestimation.

ii) For A being a real or complex interval of relative precision e, and for B being a real or complex interval of relative precision f, the overestimation of multiplication and division is bounded by

$$1 + \frac{ef}{e+f} .$$

In any case, the overestimation is uniformly bounded by 1.5.

iii) For A and B being real or complex interval vectors or matrices, midpoint-radius interval addition and subtraction does not cause any overestimation.

iv) For A being a real or complex interval vector or matrix with all components of relative precision e, and for B being a real or complex interval vector or matrix with all components of relative precision f, the overestimation of each component of the result of multiplication is bounded by

$$1 + \frac{ef}{e+f} .$$

In any case, the overestimation is uniformly bounded by 1.5.

Note that multiplication in *iv)* is ordinary matrix-vector or matrix-matrix multiplication, respectively, it is not entrywise multiplication (Hadamard product). However, the assertion holds trivially for the Hadamard product as well. For the proof of *iv)* note that

$$\begin{aligned} \text{rad}(A \odot B + C \odot D) &= \text{rad}(A \odot B) + \text{rad}(C \odot D) \\ &\leq \rho_1 \cdot \text{rad}(A \cdot B) + \rho_2 \cdot \text{rad}(C \cdot D) \\ &\leq \max(\rho_1, \rho_2) \cdot \text{rad}(A \cdot B + C \cdot D) . \end{aligned}$$

An induction argument completes the proof. Finally, we mention that all given estimations are sharp and are *worst case* bounds.

3. Implementation issues. Before we discuss the practical implementation, properties of the midpoint-radius arithmetic are summarized; later they are discussed in detail.

The major advantage is that the midpoint-radius matrix operations use exclusively pure floating point matrix operations. Therefore, the fastest (floating point) library routines can be used. This includes vector and parallel algorithms, frequently designed for a specific hardware. To our knowledge this is the first time that full advantage can be taken of the speed of modern computers. This is possible with minimal effort on the part of the user: only one miniature assembly language routine for switching the rounding mode is necessary. All the rest is written in a higher level language like FORTRAN, C or Matlab.

In the computer implementation of interval arithmetic, special care has to be taken for the rounding. Consider a set $\mathbb{F} \subseteq \mathbb{R}$ of real floating point numbers, and define

$$\mathbb{IF} := \{ <\tilde{a}, \tilde{\alpha}> : \ \tilde{a}, \tilde{\alpha} \in \mathbb{F}, \tilde{\alpha} \geq 0 \} .$$

As before, we set

$$<\tilde{a}, \tilde{\alpha}> := \{ x \in \mathbb{R} \, : \, \tilde{a} - \tilde{\alpha} \leq x \leq \tilde{a} + \tilde{\alpha} \} . \tag{9}$$

Then $\mathbb{IF} \subseteq \mathbb{IR}$. Note that the pair of floating point numbers $\tilde{a}, \tilde{\alpha} \in \mathbb{F}$ describes an infinite set of *real* numbers for $\tilde{\alpha} \neq 0$. Also note that in general there need not be floating point numbers $\tilde{a}_1, \tilde{a}_2 \in \mathbb{F}$ with $[\tilde{a}_1, \tilde{a}_2] =< \tilde{a}, \tilde{\alpha} >$. Moreover, the smallest nonzero relative precision of an interval in infimum-supremum representation is limited by the relative rounding error unit ε, whereas much narrower intervals are possible in midpoint-radius representation.

Throughout this section we will denote floating point numbers and results of floating point operations by small letters attached with a tilde.

We assume the floating point arithmetic to satisfy the IEEE 754 arithmetic standard [9]. Among others, this implies (and the following properties are in fact those we need) availability of rounding modes: rounding to nearest, rounding downwards (towards $-\infty$) and rounding upwards (towards $+\infty$). For the remainder of the paper we assume that no overflow or exception (division by zero etc.) occurs (which usually terminates computation anyway). However, *we allow underflow*. It will be shown that all computed results are correct, including the presence of underflow.

Denote the relative rounding error unit by ε, set $\varepsilon' = \frac{1}{2}\varepsilon$, and denote the smallest representable (unnormalized) positive floating point number by η. In IEEE 754 double precision, $\varepsilon = 2^{-52}$ and $\eta = 2^{-1074}$. If an expression is to be evaluated in floating point arithmetic, we put the expression in parentheses with a preceding rounding symbol. Note that *all* operations within the expression are performed with the *same* rounding. The following rounding modes are used:

$\square$	rounding to nearest,
$\bigtriangledown$	rounding downwards,
$\bigtriangleup$	rounding upwards.

This can be accomplished by a small assembly language routine setround(rnd) with rnd $\in \{-1, 0, 1\}$. For example,

$$\tilde{c} = \bigtriangledown(\tilde{a} - \tilde{a} \cdot \tilde{b})$$

implies that floating point multiplication and subtraction is used both with rounding towards $-\infty$. Note that this does not necessarily imply $\bigtriangledown(\tilde{a} - \tilde{a} \cdot \tilde{b}) \le \tilde{a} - \tilde{a} \cdot \tilde{b}$. The following error estimates are valid for any rounding mode:

$$\begin{array}{llll} (10) & \forall a, b \in \mathbb{F} & a \circ b = \mathrm{rnd}(a \circ b)(1 + \varepsilon_1) + \eta_1 & \text{for } \circ \in \{\cdot, /\}, \\ & \forall a, b \in \mathbb{F} & a \circ b = \mathrm{rnd}(a \circ b)(1 + \varepsilon_2) = \mathrm{rnd}(a \circ b)(1 + \varepsilon_3)^{-1} & \text{for } \circ \in \{+, -\}, \end{array}$$

where rnd $\in \{\square, \bigtriangledown, \bigtriangleup\}$ and

$$\begin{array}{l} |\varepsilon_i| \le \varepsilon^*,\ |\eta_1| \le \eta, \quad \varepsilon_1\eta_1 = 0 \quad \text{and} \\ \varepsilon^* = \varepsilon' \quad \text{for rounding to nearest,} \\ \varepsilon^* = \varepsilon \quad \text{otherwise}. \end{array}$$

Note that for addition and subtraction there is no extra constant η_1 covering underflow. This is known to be true if the result is a normalized floating point number, and if the result of addition or subtraction is in the gradual underflow range then it is *a fortiori* a floating point number. To our knowledge this has not been observed in the literature. Furthermore, $\tilde{a} \in \mathbb{F}$ implies $-\tilde{a} \in \mathbb{F}$ and therefore $|\tilde{a}| \in \mathbb{F}$ in IEEE 754 arithmetic. For an excellent treatment of floating point arithmetic and error estimates see [8].

With these preliminaries we can define algorithms for the midpoint-radius interval arithmetic in floating point. Special care is necessary compared to (5) because $\tilde{a} \circ \tilde{b}$ is, in general, not a floating point number.

For the remainder of the section let

$$A = <\tilde{a}, \tilde{\alpha}> \in \mathbb{IF} \quad \text{and} \quad B = <\tilde{b}, \tilde{\beta}> \in \mathbb{IF}$$

be given. Then interval addition and subtraction $C := A \circledcirc B \in \mathbb{IF}$, $\circ \in \{+, -\}$, with $C = <\tilde{c}, \tilde{\gamma}>$ is defined by the following algorithm.

$$\begin{aligned} \tilde{c} &= \square(\tilde{a} \circ \tilde{b}) \\ \tilde{\gamma} &= \triangle(\varepsilon' \cdot |\tilde{c}| + \tilde{\alpha} + \tilde{\beta}) \end{aligned} \tag{11}$$

ALGORITHM 3.1. *Addition and subtraction in* $\mathbb{IF}$.

We have to verify the fundamental property (3), the isotonicity. Let $\bar{a} \in A$, $\bar{b} \in B$ for $\bar{a}, \bar{b} \in \mathbb{R}$ by given. Then by definition (9),

$$|\bar{a} - \tilde{a}| \le \tilde{\alpha} \quad \text{and} \quad |\bar{b} - \tilde{b}| \le \tilde{\beta}\,.$$

The error estimate (10), observing the rounding modes, yields for $\circ \in \{+, -\}$,

$$|\bar{a} \circ \bar{b} - \tilde{c}| \le |\tilde{a} \circ \tilde{b} - \tilde{c}| + \tilde{\alpha} + \tilde{\beta} \le \varepsilon' \cdot |\tilde{c}| + \tilde{\alpha} + \tilde{\beta} \le \tilde{\gamma}\,. \tag{12}$$

The interval multiplication $C = A \odot B$ is defined by the following algorithm.

$$\begin{aligned} \tilde{c} &= \square(\tilde{a} \cdot \tilde{b}) \\ \tilde{\gamma} &= \triangle(\eta + \varepsilon' \cdot |\tilde{c}| + (|\tilde{a}| + \tilde{\alpha})\tilde{\beta} + \tilde{\alpha}|\tilde{b}|) \end{aligned} \tag{13}$$

ALGORITHM 3.2. *Multiplication in* $\mathbb{IF}$.

The proof of isotonicity is almost the same as for addition and subtraction. Note that in the computation of $\tilde{\gamma}$ small terms are added first (from left to right) in order to diminish accumulation of rounding errors.

Instead of Definition 2.1 we use the following definition for the interval reciprocal $C = 1 \oslash B$ in $\mathbb{IF}$. We assume $\tilde{\beta} < |\tilde{b}|$.

$$\begin{aligned} \tilde{c_1} &= \triangledown((-1)/(-|\tilde{b}| - \tilde{\beta})) \\ \tilde{c_2} &= \triangle((-1)/(-|\tilde{b}| + \tilde{\beta})) \\ \tilde{c} &= \triangle(\tilde{c_1} + 0.5 \cdot (\tilde{c_2} - \tilde{c_1})) \\ \tilde{\gamma} &= \triangle(\tilde{c} - \tilde{c_1}) \\ \tilde{c} &= \operatorname{sign}(\tilde{b}) \cdot \tilde{c} \end{aligned} \tag{14}$$

ALGORITHM 3.3. *Inversion in* $\mathbb{IF}$.

This definition uses infimum-supremum arithmetic to compute the endpoints of $1 \oslash B$. The elegant transformation of infimum-supremum representation back to midpoint-radius representation was given by Oishi [22]. The proof of isotonicity (3) is straightforward. Algorithm 14 requires only 2 switches of the rounding mode.

Finally, interval division $C = A \oslash B$ is defined as before by

$$C = A \odot (1 \oslash B)\,, \tag{15}$$

and preserves, of course, isotonicity.

Our estimations in the previous section show that the overestimation of midpoint-radius interval arithmetic increases with the radii of the operands. Therefore, this representation will be disadvantageous in applications where many operations among very wide intervals occur. This may be the case, for example, in global optimization.

On the other hand, it looks like as if midpoint-radius interval arithmetic is always worse than infimum-supremum arithmetic. This is indeed true for the theoretical operations in $\mathbb{IR}$ where no final rounding of the results is necessary. However, this changes when looking at operations in $\mathbb{IF}$.

First, the midpoint-radius representation offers the possibility to define very accurate intervals such as $< 1, 10^{-25} >$, whereas the relative accuracy of infimum-supremum representation is restricted to the machine precision. This may occasionally be useful.

Second, the result of an operation using midpoint-radius representation in $\mathbb{IF}$ may be *narrower* than the corresponding result of infimum-supremum arithmetic. For example, consider a 2-digit decimal arithmetic and the two intervals

$$A = < 1.3, 0.1 > \quad \text{and} \quad B = < 9.2, 0.1 > \ . \tag{16}$$

Both have exactly representable endpoints, and a corresponding (exact) infimum-supremum representation is

$$A = [1.2, 1.4] \quad \text{and} \quad B = [9.1, 9.3] \, .$$

In rounded 2-digit decimal arithmetic we obtain for the infimum-supremum representation

$$A \cdot B = [1.2, 1.4] \cdot [9.1, 9.3] \subseteq [10, 14] \, .$$

By assumption (10) it follows for our model arithmetic $\varepsilon' = 0.05$, and Algorithm 3.2 for midpoint-radius representation yields the result

$$\tilde{c} = 12 \quad \text{and} \quad \tilde{\gamma} = 1.7 \, , \quad \text{which means} \quad A \odot B = < 12, 1.7 > = [10.3, 13.7] \, .$$

This compares to the result $[10, 14]$ in rounded infimum-supremum arithmetic. Thus, the radius of the result of midpoint-radius representation is only 85% of the radius of the result of infimum-supremum representation.

As the next step towards vector and matrix operations consider a scalar product $A^T B$ for $A, B \in \mathbb{IF}^n$, $A = < \tilde{a}, \tilde{\alpha} >$, $B = < \tilde{b}, \tilde{\beta} >$ with $\tilde{a}, \tilde{b}, \tilde{\alpha}, \tilde{\beta} \in \mathbb{F}^n, \tilde{\alpha}, \tilde{\beta} \geq 0$ (comparison and absolute value of vectors and matrices is always to be understood entrywise). We have to estimate

$$|\Box(\tilde{a}^T \tilde{b}) - \tilde{a}^T \tilde{b}| \, .$$

This is simple if a precise scalar product as proposed by Kulisch [16] is available. On a general computer hardware with arithmetic according to IEEE 754 we may use the well known error estimate due to Wilkinson. In the notation of [8] we set

$$\gamma_n := \frac{n\varepsilon'}{1 - n\varepsilon'} \, , \tag{17}$$

and, if no underflow occurs, it is for quantities $|\Theta_j| \leq \gamma_j, |\Theta'_n| \leq \gamma_n$,

$$|\text{fl}(\tilde{a}^T \tilde{b}) - \tilde{a}^T \tilde{b}| \leq \tilde{a}_1 \tilde{b}_1 \cdot \Theta_n + \tilde{a}_2 \tilde{b}_2 \cdot \Theta'_n + \tilde{a}_3 \tilde{b}_3 \cdot \Theta_{n-1} + \ldots + \tilde{a}_n \tilde{b}_n \cdot \Theta_2 \, . \tag{18}$$

We note that this estimate is valid for almost every computer arithmetic, even for those without a guard digit. The right hand side of (18) is expensive to compute and is frequently replaced by

$$|\Box(\tilde{a}^T \tilde{b}) - \tilde{a}^T \tilde{b}| \leq \gamma_n \cdot |\tilde{a}^T||\tilde{b}| \, .$$

The final forward error estimation taking underflow into account is

$$|\Box(\tilde{a}^T \tilde{b}) - \tilde{a}^T \tilde{b}| \leq \gamma_n \cdot (|\tilde{a}^T||\tilde{b}| + \eta) =: \rho_n \, . \tag{19}$$

This estimation can be improved as follows. The definition of the rounding modes imply

$$\bigtriangledown(\tilde{p} \circ \tilde{q}) \le \tilde{p} \circ \tilde{q} \le \triangle(\tilde{p} \circ \tilde{q}) \quad \text{for} \quad \tilde{p}, \tilde{q} \in \mathbb{F},\ \circ \in \{+, -, \cdot, /\}\ , \tag{20}$$

and therefore $\bigtriangledown(\tilde{a}^T\tilde{b}) \le \tilde{a}^T\tilde{b} \le \triangle(\tilde{a}^T\tilde{b})$. Hence, the following algorithm can be used for the scalar product in $\mathbb{IF}^n$.

$$\begin{aligned}
\tilde{c}_1 &= \bigtriangledown(\tilde{a}^T\tilde{b}) \\
\tilde{c}_2 &= \triangle(\tilde{a}^T\tilde{b}) \\
\tilde{c} &= \triangle(\tilde{c}_1 + 0.5(\tilde{c}_2 - \tilde{c}_1)) \\
\tilde{\gamma} &= \triangle((\tilde{c} - \tilde{c}_1) + (|\tilde{a}^T| + \tilde{\alpha}^T)\tilde{\beta} + \tilde{\alpha}^T|\tilde{b}|)
\end{aligned}$$

Algorithm 3.4. *Scalar product in* $\mathbb{IF}^n$.

Note the sequence of summation in the last line of Algorithm 3.4. For execution from left to right we add the small terms first in order to diminish accumulation of rounding errors.

It is clear from the definition and especially from the used rounding modes that Algorithm 3.4 satisfies the isotonicity property

$$\forall a \in <\tilde{a}, \tilde{\alpha}> \ \forall b \in <\tilde{b}, \tilde{\beta}>:\ a^T b \in <\tilde{c}, \tilde{\gamma}>\ . \tag{21}$$

This is also true in the presence of underflow.

Notice that all estimations *are independent of the order of summation.* Therefore, midpoint-radius arithmetic may take full advantage of any computational scheme to compute scalar and matrix products. This includes especially vector and parallel architectures, blocked algorithms etc., where frequently the order of summation is changed for improved performance.

The main advantage of using Algorithm 3.4 instead of (19) appears for narrow intervals. The following table compares the results for point intervals. For each value of n, 1000 random sample scalar products were computed with vector components uniformly distributed in $[-1, 1]$.

n	$\min(\tilde{\gamma}/\rho_n)$	average$(\tilde{\gamma}/\rho_n)$	$\max(\tilde{\gamma}/\rho_n)$
10	0.07600	0.20000	0.4700
100	0.01400	0.03900	0.1200
1000	0.00380	0.01200	0.0360
10 000	0.00092	0.00370	0.0120
100 000	0.00033	0.00120	0.0037
1 000 000	0.00011	0.00037	0.0011

Table 3.1. *Improvement of* $\tilde{\gamma}$ *compared to* (19).

Obviously, making use of the actual data in Algorithm 3.4 is better than the general estimation (19), and for scalar products with cancellation it is much better. The algorithms for matrix operations use Algorithms 3.1 and 3.4, and the algorithms for complex operations including vector and matrix operations can be written in a similar way.

4. Computational speed. The core of many numerical algorithms is the solution of systems of linear equations. The popular Krawczyk-type iteration [14, 24] for the verified solution of dense linear systems requires computation of an approximate inverse R as preconditioner, and the verified computation of the residual $I - RA$. For sparse systems of linear equations [25] an approximation of the smallest singular value (for example, by inverse iteration) and an estimation of a matrix residual is computed.

In any case, those algorithms split i) into a pure floating point part and ii) into a verification part. For the first part, the fastest floating point algorithm may be used, and there are many choices available. When just counting operations, regardless whether (floating) point or interval operations, both parts have roughly the same operation count. Therefore, the second part with the verified computation of a matrix product is the computationally intensive part.

In other words, the computational speed of the verified solution of a linear system is determined by the computational speed of verified matrix multiplication.

We have to distinguish two cases. First, assume the matrix data of the linear system is exactly representable in floating point on the computer, i.e. the system matrix is a point matrix with representable entries. Then, both for dense and for sparse systems, the computationally intensive part is a point matrix multiplication, both operands being exact. In this case we do not have to distinguish between infimum-supremum and midpoint-radius arithmetic at all. In either case Algorithm 3.4 may be used with the radii being zero by definition. The result of infimum-supremum arithmetic is just $[\tilde{c}_1, \tilde{c}_2]$.

The main point is that no case distinctions, switching of rounding mode inner loops, etc. are necessary, only *pure floating point matrix multiplications.* And for those the fastest algorithms available may be used, for example, BLAS. The latter bear the striking advantages that i) they are available for almost every computer hardware, and that ii) they are individually adapted and tuned for specific hardware and compiler configurations. This gives an advantage in computational speed which is difficult to achieve by other implementations.

The advantage is not only speed but *portability together with speed*. The only machine dependent routine is the assembly language routine for switching the rounding mode.

Things change in the second case, for interval linear systems, i.e. the matrix is afflicted with tolerances. Consider $R \in \mathbb{F}^{n\times n}$ and $\mathcal{A} \in \mathbb{IF}^{n\times n}$. We have to calculate an inclusion of $R \cdot \mathcal{A}$. The well known problem for infimum-supremum arithmetic is that the result of $c := R_{ik}\mathcal{A}_{kj}$ depends on the sign of R_{ik}:

$$\begin{array}{llll} \text{if} & R_{ik} \ge 0 & \text{then} & \inf(c) = \triangledown(R_{ik} \cdot \inf(\mathcal{A}_{kj}));\ \sup(c) = \vartriangle(R_{ik} \cdot \sup(\mathcal{A}_{kj})) \\ & & \text{else} & \inf(c) = \triangledown(R_{ik} \cdot \sup(\mathcal{A}_{kj}));\ \sup(c) = \vartriangle(R_{ik} \cdot \inf(\mathcal{A}_{kj}))\ . \end{array}$$

ALGORITHM 4.1. *Inner loop of matrix multiplication: the top-down approach.*

This approach is used in almost all current implementations of interval libraries. It is a top-down approach starting with the usual three-loop definition of matrix multiplication. However, this standard approach to interval matrix multiplication requires some n^3 comparisons and switches of the rounding mode. More important, the extensive compiler optimization techniques for accelerating scalar product code are of virtually no use. It has been observed by Knüppel [13] that the code can be rewritten in order to reduce the number of comparisons and switches of rounding mode to n^2:

```
for $i = 1..n$ do
    for $k = 1..n$ do
        if $R_{ik} \geq 0$ then
            setround(-1)
            for $j = 1..n$ do $\inf(C_{ij}) = \inf(C_{ij}) + R_{ik} \cdot \inf(\mathcal{A}_{kj})$
            setround(1)
            for $j = 1..n$ do $\sup(C_{ij}) = \sup(C_{ij}) + R_{ik} \cdot \sup(\mathcal{A}_{kj})$
        else
            setround(-1)
            for $j = 1..n$ do $\inf(C_{ij}) = \inf(C_{ij}) + R_{ik} \cdot \sup(\mathcal{A}_{kj})$
            setround(1)
            for $j = 1..n$ do $\sup(C_{ij}) = \sup(C_{ij}) + R_{ik} \cdot \inf(\mathcal{A}_{kj})$
```

ALGORITHM 4.2. *Improved infimum-supremum matrix multiplication.*

Using an independent test suite [3] it has been shown [13] that the PROFIL/BIAS library, which is based on Algorithm 4.2, is faster by up to two orders of magnitude compared to existing interval libraries [10, 11, 17]. In Knüppel's implementation the inner j-loops are scalar products, free of comparisons or rounding switches. But still, only BLAS-1 can be used. An implemention following Algorithm 3.4 with result $<\tilde{C}, \tilde{\Gamma}> \supseteq R \cdot \mathcal{A}$ looks as follows.

```
setround(-1)
$\tilde{C}_1 = R \cdot \mathrm{mid}(\mathcal{A})$
setround(1)
$\tilde{C}_2 = R \cdot \mathrm{mid}(\mathcal{A})$
$\tilde{C} = \tilde{C}_1 + 0.5(\tilde{C}_2 - \tilde{C}_1)$
$\tilde{\Gamma} = (\tilde{C} - \tilde{C}_1) + |R| \cdot \mathrm{rad}(\mathcal{A})$
```

ALGORITHM 4.3. *Midpoint-radius matrix multiplication.*

At first sight, one recognizes three floating point matrix multiplications, resulting in $3n^3$ operations. However, these may make use of full optimization techniques, these are BLAS-3 and therefore readily available and very fast on almost any platform.

Finally we remark that optimized BLAS may change the order of computation of a scalar product. Notice that the error estimations following Algorithm 3.4 are independent of the order of execution. Therefore, all results are reliable with taking full advantage of increased performance by use of BLAS. Especially, the full performance of vector and parallel architectures is achievable. The algorithms above, especially Algorithm 4.3, may convince the reader that this is possible with minimal effort on the part of the user.

Below we will exploit this performance impact in more detail. We will compare the three different approaches:

i) the standard approach (Algorithm 4.1),
ii) the improved approach by BIAS (Algorithm 4.2),
iii) the new approach (Algorithm 4.3).

The algorithms are implemented on a parallel computer Convex SPP 2000 using up to 4 processors. The comparisons were performed by Jens Zemke. The programs are written in C with full optimization by the compiler.

The first example measures multiplication of a point matrix and an interval matrix for different matrix sizes. This type of matrix multiplication occurs typically in the verified solution of an interval linear or nonlinear system when multiplying by a preconditioning matrix.

The computations have been performed in scalar and parallel mode. All of the following tables show computing time in seconds.

dimension		100	200	500	1000
i)	standard	0.31	2.48	71.4	570.8
ii)	BIAS	0.03	0.33	13.4	107.6
iii)	new	0.02/0.01	0.07/0.03	1.76/0.63	12.3/3.8

TABLE 4.1. *Computing times for point matrix times interval matrix.*

For the new method, two computing times are given: the first for scalar mode, the second for parallel mode with 4 processors. For larger dimensions, the code is almost fully parallelized. This is in fact the parallelization of the Convex BLAS library (veclib).

For the methods *i*) and *ii*), we give only one computing time because scalar and parallel times are identical. For the standard method it would be difficult to parallelize the code because of the switching of rounding modes in the inner loop. For the second, the BIAS approach (Algorithm 4.2), a parallelization would be possible and would improve the computing time. However, this has to be done on the part of the user. And this is the fundamental advantage of the new approach: The BLAS library is already parallelized by the manufacturer. This guarantees a most easy and fast implementation of the new algorithms.

The second example is the multiplication of two interval matrices. In this case the BIAS implementation is almost identical to the standard approach. There are case distinctions together with switching of the rounding mode in the most inner loop. That means we expect computing times of the standard approach and BIAS approach to be similar.

We have to distinguish two cases. First, not both intervals contain zero, second, both intervals contain zero. For the standard and BIAS approach that means the following. In the first case (which splits into 8 subcases due to different sign combinations), the operands determining the lower and upper bound can be calculated in advance. That means 2 multiplications are necessary in the inner loop. In the second case, the operands determining the lower and upper bound cannot be calculated in advance but have to be detected within the loop resulting in 4 multiplications and various switching of the rounding mode in the inner loop (cf. [1, 19, 21]).

In the new approach, no case distinctions are necessary at all. This is the main reason for its speed. Therefore, it does not matter for the computing time whether the input intervals contain zero or not.

The first table gives computing times for interval matrix times interval matrix, where all interval components are generated *not* to contain zero. This is most advantageous for the standard and for the BIAS approach. In fact we generate the intervals to be positive such that the very first case distinction for interval multiplications is satisfied in every component. In other words, the computing times in Table 4.2 are the best possible for the first and for the second approach.

dimension		100	200	500	1000
i)	standard	0.30	2.40	72.3	613.6
ii)	BIAS	0.29	2.30	71.0	581.8
iii)	new	0.02/0.02	0.15/0.10	2.77/0.79	16.2/4.8

TABLE 4.2. *Times for interval matrix times interval matrix, no zero intervals.*

In case of interval matrix times interval matrix the new approach requires 5 point matrix multiplications (Algorithm 4.3 adapted according to Algorithm 3.4). This seems a lot at first sight. However, our previous argument applies: In the practical implementation on today's computer architectures the possibility to optimize code is far more important than the naked floating point operations count. As the table shows, the new approach is again faster by a factor 20 to 100. And again, the speedup by having and using more processors on a parallel machine applies one to one to the new method, whereas the parallelization of the standard approach is at least difficult, if possible at all. The times in scalar and parallel mode for the first and for the second approach are again identical.

dimension		100	200	500	1000
i)	standard	0.58	4.66	106.4	855.9
ii)	BIAS	0.44	3.56	90.1	735.5
iii)	new	0.02/0.02	0.15/0.10	2.78/0.79	16.1/4.8

TABLE 4.3. *Times for interval matrix times interval matrix, only zero intervals.*

Finally, we give the computing times for interval matrix times interval matrix when all intervals contain zero.

The computing times for the new approach are, of course, the same as before. The additional case distinctions in the standard and BIAS approach cost some extra 20 to 50 % computing time. Note the speed up by more than two orders of magnitude by the new method using four processors.

In the following section a number of model problems are numerically investigated to check overestimations in some practical computations.

5. Computational results. The estimations of the ratio between the result of midpoint-radius interval operations and power set operations are worst case. The question is whether the worst case factor 1.5 is attained in practical computations, and whether it perpetuates for composed operations. All computations in this section are performed using IEEE 754 double precision corresponding to approximately 16 decimal digits.

The first test problem is the solution of systems of linear equations. We use the popular Krawczyk-type iteration [14] with improvements such as inclusion of the error with respect to an approximate solution [24]. For different values of n we generate

$$A = (2 \cdot \mathrm{rand}(n) - 1) \cdot (1 \pm e), \quad b = A \cdot (2 \cdot \mathrm{rand}(n,1) - 1)\,, \tag{22}$$

that means the midpoint of A_{ij} is a random number uniformly distributed in $[-1, 1]$, the intervals A_{ij} are of relative accuracy e, and the right hand side is generated such that the solution of the midpoint linear system is a random vector.

Let X denote the inclusion vector computed by midpoint-radius arithmetic, and let Y denote the inclusion vector computed by infimum-supremum arithmetic. In the following table we display the average entrywise ratio

$$\mathrm{sum}(\mathrm{rad}(X)./\mathrm{rad}(Y))/n \tag{23}$$

for 10 samples each, and for combinations of n and e (the above is Matlab notation). A number greater than one implies that (on the average) the radius of the components of the inclusion vector computed by midpoint-radius arithmetic is larger by this factor than the corresponding radius for infimum-supremum arithmetic.

n	$e = 10^{-6}$	$e = 10^{-9}$	$e = 10^{-12}$	$e = 10^{-15}$
10	1.0000	1.0000	1.0000	0.9426
20	1.0000	1.0000	1.0000	0.9395
50	1.0000	1.0000	1.0000	0.9405
100	0.9970	1.0000	0.9999	0.9435
200	0.9945	1.0000	0.9999	0.9402
500	0.9830	0.9998	0.9998	0.9199

TABLE 5.1. *Ratio mid-rad/inf-sup for linear systems.*

By the proof of Proposition 2.7 one may expect the ratio to increase for larger diameters. This is not the case. On the contrary we observe even an improvement compared to infimum-supremum arithmetic, similar to example (16). Another reason for that behaviour is the fact that, rather than computing an inclusion of the solution itself, an inclusion of the difference of the exact solution and an approximate solution is computed.

The next two examples concern the validated inclusion of *all* solutions of a system of nonlinear equations within a certain domain. We use the algorithm proposed by Knüppel [12]. The first well-known example is due to Price [4],[12, Problem 5.1]:

$$\begin{aligned} f_1(x) &= 2x_1^3x_2 - x_2^3 \\ f_2(x) &= 6x_1 - x_2^2 + x_2 \quad \text{within} \quad -10 \le x_i \le 10,\ i \in \{1,2\}\,. \end{aligned}$$

There are three zeros within that domain. For this example the midpoint-radius arithmetic used 19.7% more bisections yielding inclusion intervals with relative error less than 2% worse than infimum-supremum arithmetic.

The second example is Powell's almost singular function [4],[12, Problem 5.5]:

$$\begin{aligned} f_1(x) &= x_1 + 10x_2 \\ f_2(x) &= \sqrt{5}(x_3 - x_4) \\ f_3(x) &= (x_2 - 2x_3)^2 \\ f_4(x) &= \sqrt{10}(x_1 - x_4)^2 \quad \text{within} \quad -3 \le x_i \le 4 \quad \text{for} \quad i \in \{1, ..., 4\}\,. \end{aligned}$$

Here, the zero vector is the unique solution within that domain. For this example, the number of bisections and final results are identical for both kinds of arithmetic.

In the final examples, pure evaluation of functions is tested. The first is Brown's almost linear function [20], [12, Problem 5.7]:

$$\begin{aligned} f_i(x) &= x_i + \sum x_j - (n+1) \quad \text{for} \quad 1 \le i \le n-1 \\ f_n(x) &= \prod x_i - 1\,. \end{aligned}$$

We evaluate f near the solution $(1, ..., 1)^T$ of $f(x) = 0$ for interval arguments of different radii. We use

$$x = < 1 + 0.01 \cdot y, e > ,$$

where y is a vector with entries uniformly distributed within $[-1, 1]$. For different dimensions n and radii e we list again the average ratio for some 10 samples. The results are as follows.

n	$e = 10$	$e = 1$	$e = 0.1$	$e = 10^{-2}$	$e = 10^{-12}$	$e = 10^{-15}$
10	1.01	1.10	1.04	1.00	1.00	1.03
20	1.01	1.05	1.04	1.00	0.99	1.03
50	1.00	1.02	1.02	1.00	1.00	1.01

TABLE 5.2. *Ratio mid-rad/inf-sup for Brown's almost linear function.*

For e between 10^{-11} and 10^{-3} the ratio was always 1.00, for all $n \in \{10, 20, 50\}$.

The next example is designed to test whether the worst case factor 1.5 for the overestimation of a single multiplication in midpoint-radius arithmetic may perpetuate for several successive multiplications. For $x, x_1, ..., x_n$ randomly distributed in $[-10, 10]$ define

$$f(X) = \sum_{i=1}^{n} \prod_{\substack{j=1 \\ j \neq i}}^{n} (X - x_j)\,, \tag{24}$$

where $X := [x - e, x + e]$. The following table shows results for different combinations of n and e.

n	$e = 10$	$e = 1$	$e = 0.1$	$e = 10^{-2}$	$e = 10^{-12}$	$e = 10^{-15}$
10	1.09	1.63	1.14	1.01	1.00	0.98
20	1.05	1.30	1.24	1.03	1.00	0.92
50	1.02	1.21	1.59	1.12	1.00	0.90
100	1.01	1.11	1.82	1.22	1.00	0.84

TABLE 5.3. *Ratio mid-rad/inf-sup for* (24).

The worst overestimations of midpoint-radius arithmetic are observed for e around 1. This confirms Theorem 2.4. For e larger than 1, the midpoint of X moves towards zero (on the average), thus reducing the overestimation. For the same reason the overestimation grows with n for smaller e, but reduces with growing n for large values of e. For e between 10^{-4} and 10^{-11}, the ratio was always 1.00 for all $n \in \{10, 20, 50, 100\}$. For very small values of e we observe the same behaviour as in example (16), namely that midpoint-radius arithmetic becomes better than infimum-supremum representation.

Finally, we repeat this test for Griewank's function [27]

$$f(x) = \sum_{\nu=1}^{n} \frac{x_\nu^2}{4000} - \prod_{\nu=1}^{n} \cos\left(\frac{x_\nu}{\sqrt{\nu}}\right) + 1$$

with test vectors

$$x = < 10y, e >$$

and random vector y with entries uniformly distributed within $[-1, 1]$. The results for different radii e are as follows.

n	$e = 10$	$e = 1$	$e = 0.1$	$e = 10^{-2}$	$e = 10^{-12}$	$e = 10^{-15}$
10	1.09	1.21	1.12	1.02	0.99	0.88
20	1.15	1.12	1.01	1.00	0.99	0.97
50	1.21	1.09	1.01	1.00	0.99	0.91

TABLE 5.4. *Ratio mid-rad/inf-sup for Griewank's function.*

As before, the ratio was always 1.00 for e between 10^{-11} and 10^{-3}, for all $n \in \{10, 20, 50\}$. We observe that for small radius the midpoint-radius arithmetic yields better results than infimum-supremum arithmetic, a behaviour similar to example (16).

6. Conclusion.

We proved error estimates for the maximal overestimation of midpoint-radius interval arithmetic compared to power set operations. We saw that this overestimation is uniformly bounded by a worst case factor 1.5. There is numerical evidence that this factor is much smaller in practical computations and does not perpetuate.

The main advantage of our algorithms for midpoint-radius interval arithmetic is that full speed of modern computer architectures can be utilized. To our knowledge this is the first time that the peak performance of vector and parallel computers is approached for interval calculations.

We showed that the implementation is easy, and it requires only one small assembly language routine for switching the rounding mode. No cumbersome case distinctions, which may slow down the computation by orders of magnitude, are necessary.

All algorithms including real and complex vectors and matrices have been implemented in the Matlab interval toolbox INTLAB [26], which is available for unlimited non-profit use from

http://www.ti3.tu-harburg.de/rump/intlab/index.html.

Beside the algorithms described in this note it comprises of a gradient toolbox for real/complex points, vectors and matrices (full and sparse) as well as intervals over those, a similar slope toolbox, rigorous input/output routines, rigorous and fast standard functions for point and interval input and more.

Acknowledgement. The author wishes to thank the two annonymous referees for their thorough reading and their hints improving the exposition. The author is especially indebted to Prof. Shin'Ishi Oishi from Waseda University, Tokyo, for stimulating discussions.

REFERENCES

[1] G. Alefeld and J. Herzberger. *Introduction to Interval Computations.* Academic Press, New York, 1983.
[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, and S. Hammarling. *LAPACK User's Guide.* SIAM Publications, Philadelphia, 1992.
[3] G.F. Corliss. Comparing software packages for interval arithmetic. Preprint presented at SCAN'93, Vienna, 1993.
[4] J.E. Dennis Jr. and R.B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Englewood Cliffs, N.J., 1983.
[5] J.J. Dongarra, J.R. Bunch, C.B. Moler, and G.W. Stewart. *LINPACK user's Guide.* SIAM, Philadelphia, PA, 1979.
[6] J.J. Dongarra, J. Du Croz, I.S. Duff, and S. Hammarling. Algorithm 679: A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Soft.*, 16:18–28, 1990.
[7] I. Gargantini and P. Henrici. Circular Arithmetic and the Determination of Polynomial Zeros. *Numer. Math.*, 18:305–320, 1972.
[8] N.J. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM Publications, Philadelphia, 1996.
[9] *ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic*, 1985.
[10] R.B. Kearfott, M. Dawande, and C. Hu. INTLIB: A portable Fortran-77 interval standard function library. *ACM Trans. Math. Software*, 20:447–459, 1994.
[11] R. Klatte, U. Kulisch, M. Neaga, D. Ratz, and Ch. Ullrich. *PASCAL-XSC: Language reference with examples.* Springer, 1992.

[12] O. Knüppel. *Einschließungsmethoden zur Bestimmung der Nullstellen nichtlinearer Gleichungssysteme und ihre Implementierung.* PhD thesis, Technische Universität Hamburg-Harburg, 1994.

[13] O. Knüppel. PROFIL / BIAS — A Fast Interval Library. *Computing*, 53:277–287, 1994.

[14] R. Krawczyk. Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken. *Computing*, 4:187–201, 1969.

[15] R. Krier. *Komplexe Kreisarithmetik.* PhD thesis, Universität Karlsruhe, 1973.

[16] U. Kulisch. *Grundlagen des numerischen Rechnens (Reihe Informatik 19).* Bibliographisches Institut, Mannheim, Wien, Zürich, 1976.

[17] C. Lawo. C-XSC, a programming environment for verified scientific computing and numerical data processing. In E. Adams and U. Kulisch, editors, *Scientific computing with automatic result verification*, pages 71–86. Academic Press, Orlando, Fla., 1992.

[18] C.L. Lawson, R.J. Hanson, D. Kincaid, and F.T. Krogh. Basic Linear Algebra Subprograms for FORTRAN usage. *ACM Trans. Math. Soft.*, 5:308–323, 1979.

[19] R.E. Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs, N.J., 1966.

[20] J.J. Morè and M.Y. Cosnard. Numerical solution of non-linear equations. *ACM Trans. Math. Software*, 5:64–85, 1979.

[21] A. Neumaier. *Interval Methods for Systems of Equations.* Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1990.

[22] S. Oishi. Finding all solutions of nonlinear systems of equations using linear programming with guaranteed accuracy. *J. Universal Computer Science*, 4(2):171–177, 1998.

[23] L.D. Petković and M.S. Petković. Inequalities in Circular Arithmetic: a Survey. *Mathematics and Its Applications*, 430:325–340, 1998.

[24] S.M. Rump. *Kleine Fehlerschranken bei Matrixproblemen.* PhD thesis, Universität Karlsruhe, 1980.

[25] S.M. Rump. Validated Solution of Large Linear Systems. In R. Albrecht, G. Alefeld, and H.J. Stetter, editors, *Validation numerics: theory and applications*, volume 9 of *Computing Supplementum*, pages 191–212. Springer, 1993.

[26] S.M. Rump. INTLAB - INTerval LABoratory. In Tibor Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999.

[27] A. Törn and A. Žilinskas. *Global Optimization.* Springer-Verlag, Berlin, Heidelberg, New York, 1989.