Journal of Computational and Applied Mathematics 236 (2012) 1795–1814

Contents lists available at SciVerse ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

Fast algorithms for floating-point interval matrix multiplication

Katsuhisa Ozaki [a,d,*], Takeshi Ogita [b,d], Siegfried M. Rump [e,c], Shin'ichi Oishi [c,d]

[a] Department of Mathematical Sciences, College of Systems Engineering and Science, Shibaura Institute of Technology, 307 Fukasaku, Minuma-ku, Saitama-shi, Saitama 337-8570, Japan

[b] Department of Mathematical Sciences, Tokyo Woman's Christian University, 2-6-1 Zempukuji, Suginami-ku, Tokyo 167-8585, Japan

[c] Faculty of Science and Engineering, Waseda University, 3-4-1 Okubo, Shinjyuku-ku, Tokyo 169-8555, Japan

[d] JST (Japan Science and Technology Agency), CREST, Japan

[e] Institute for Reliable Computing, Hamburg University of Technology, Schwarzenbergstr. 95, 21071 Hamburg, Germany

ARTICLE INFO

Article history:
Received 11 November 2010
Received in revised form 21 October 2011

Keywords:
Matrix multiplication
Interval arithmetic
Verified numerical computations
INTLAB

ABSTRACT

We discuss several methods for real interval matrix multiplication. First, earlier studies of fast algorithms for interval matrix multiplication are introduced: naive interval arithmetic, interval arithmetic by midpoint–radius form by Oishi–Rump and its fast variant by Ogita–Oishi. Next, three new and fast algorithms are developed. The proposed algorithms require one, two or three matrix products, respectively. The point is that our algorithms quickly predict which terms become dominant radii in interval computations. We propose a hybrid method to predict which algorithm is suitable for optimizing performance and width of the result. Numerical examples are presented to show the efficiency of the proposed algorithms.

1. Introduction

This paper is concerned with interval matrix multiplication. Nowadays, interval arithmetic [1] is widely used, for example, to determine accuracy of numerical computations or in computer-assisted proof [2,3]. There are implementations of interval arithmetic, for example, INTLAB [4], In4sci [5], C-XSC [6], b4m [7] and so on.

We develop new algorithms for computing the product of interval matrices with floating-point entries. Before explaining the problem in detail, we discuss representations of an interval. Let $\mathbb{R}$ be the set of real numbers; let $\mathbb{IR}$ be the set of real intervals. Then $[a] \in \mathbb{IR}$ can be represented by the inf–sup form:

$$[a] = [\underline{a}, \overline{a}] = \{x \in \mathbb{R} \mid \underline{a} \leq x \leq \overline{a}\},$$

or by the mid–rad form:

$$[a] = \langle c, r \rangle = \{x \in \mathbb{R} \mid c - r \leq x \leq c + r\}, \quad r \geq 0.$$

Interval matrices or vectors can be represented similarly. For example, $[C] = [\underline{C}, \overline{C}] \in \mathbb{IR}^{m \times n}$ is defined by

$$\{C \in \mathbb{R}^{m \times n} \mid \underline{C} \leq C \leq \overline{C}\}$$

provided

$$\underline{c}_{ij} \leq c_{ij} \leq \overline{c}_{ij}, \quad 1 \leq i \leq m,\ 1 \leq j \leq n$$

* Corresponding author at: Department of Mathematical Sciences, College of Systems Engineering and Science, Shibaura Institute of Technology, 307 Fukasaku, Minuma-ku, Saitama-shi, Saitama 337-8570, Japan.
E-mail address: ozaki@sic.shibaura-it.ac.jp (K. Ozaki).

0377-0427/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2011.10.011

and using entrywise comparison. For $[a], [b] \in \mathbb{IR}$, $[a] \circ [b]$ with $\circ \in \{+, -, *\}$ is defined by

$$[a] \circ [b] := \cap\{[c] \in \mathbb{IR} : \ a \circ b \in [c] \text{ for all } a \in [a], \ b \in [b]\}. \tag{1}$$

One verifies

$$\begin{aligned}
&[a] + [b] = [\underline{a} + \underline{b}, \overline{a} + \overline{b}],\\
&[a] - [b] = [\underline{a} - \overline{b}, \overline{a} - \underline{b}],\\
&[a] * [b] = [\min(\underline{a} * \underline{b}, \underline{a} * \overline{b}, \overline{a} * \underline{b}, \overline{a} * \overline{b}), \ \ \max(\underline{a} * \underline{b}, \underline{a} * \overline{b}, \overline{a} * \underline{b}, \overline{a} * \overline{b})].
\end{aligned}$$

Interval matrix multiplication is defined similar to (1). Oishi and Rump [4,8] developed an algorithm for interval matrix multiplication by clever use of the midpoint–radius interval. The dominant computations in their algorithm are four floating-point matrix products. Therefore, the computational performance of their algorithm is high since it receives much benefit from optimized BLAS. Ogita and Oishi [9] reduced the number of floating-point matrix products to two at the cost of wider output intervals. Alternative algorithms were developed in [10].

The aim of this paper is to investigate fast algorithms which are well-balanced between tightness of the result interval and computational performance. We introduce three new algorithms for real interval matrix multiplication. There is a tradeoff between tightness of the result interval and the number of matrix products. One of the algorithms mainly requires only one floating-point matrix product, the others mainly involve two or three matrix products, respectively. Our algorithms quickly predict which terms in the interval computations become dominant radii. After that, computational efforts are made in order to avoid the overestimation of the radius. From numerical examples, the tightness of the result interval by the proposed algorithms is comparable to that by Oishi–Rump's algorithm in many cases.

This paper is organized as follows. In the following section, we introduce earlier work of interval arithmetic: rounded interval arithmetic, naive interval arithmetic, interval matrix multiplication by Oishi and Rump and its fast variant by Ogita and Oishi. In Section 3, we investigate algorithms to compute interval matrix multiplication with one, two and three matrix products in turn. In Section 4, we specialize the proposed algorithms into a product of a point matrix and an interval matrix. Numerical examples are shown in Sections 3 and 4 to illustrate the efficiency of the proposed algorithms. In this paper, MATLAB-like notation [11] is used for readability. It implies that all expressions are evaluated by floating-point arithmetic.

2. Interval matrix multiplication

In this section, we state earlier work of real interval matrix multiplication. First, we briefly review rounded interval arithmetic and naive interval arithmetic. Next, an algorithm developed by Oishi and Rump is introduced. Finally, we explain an algorithm developed by Ogita and Oishi.

2.1. Rounded interval arithmetic and naive interval arithmetic

First of all, we introduce the notation used in this paper. Let $\mathbb{F}$ be the set of binary floating-point numbers defined by the IEEE 754 standard [12]. Let $\mathbb{IF}$ be the set of intervals whose end points are floating-point numbers. An inequality for matrices $X < Y$, $X, Y \in \mathbb{R}^{m\times n}$ means that $x_{ij} < y_{ij}$ is satisfied for all (i, j). Similar notation is used for inequality for vectors. For $x \in \mathbb{R}^m$, $|x|$ denotes a nonnegative vector with $|x| = (|x_1|, \ldots, |x_m|)^T$. For real and floating-point matrices, similar notation taking absolute values is used. Let $\mathrm{fl}(\cdots)$, $\mathrm{fl}_\nabla(\cdots)$ and $\mathrm{fl}_\Delta(\cdots)$ denote that an expression inside the parenthesis is performed by pure floating-point arithmetic [12,13] with rounding to nearest, rounding downward and rounding upward, respectively. In algorithms, we use the following function which switches rounding modes defined by the IEEE 754 standard:

- `setround(-1)`: rounding downward,
- `setround(0)`: rounding to nearest,
- `setround(1)`: rounding upward.

The above-mentioned function is used in INTLAB [4]. A function $\mathtt{infsup}(\underline{a}, \overline{a})$ for $\underline{a}, \overline{a} \in \mathbb{F}$ with $\underline{a} \le \overline{a}$ indicates an interval $[a] \in \mathbb{IF}$ such that

$$[a] = \mathtt{infsup}(\underline{a}, \overline{a}) = \{x \in \mathbb{R} \mid \underline{a} \le x \le \overline{a}\}. \tag{2}$$

For $c, r \in \mathbb{F}$ with $r \ge 0$, a function $\mathtt{midrad}(c, r)$ outputs an enclosure of $\langle c, r\rangle$ by

$$\mathtt{midrad}(c, r) = [\mathrm{fl}_\nabla(c - r), \mathrm{fl}_\Delta(c + r)]. \tag{3}$$

Note that $\langle c, r\rangle \subseteq \mathtt{midrad}(c, r)$ since the end-points must be represented by floating-point numbers, respectively. The following algorithm computes a pair $\langle M, R\rangle$ of a midpoint and a radius from inf–sup form $[\underline{A}, \overline{A}]$ by floating-point arithmetic such that $[\underline{A}, \overline{A}] \subseteq \langle M, R\rangle$.

Algorithm 1 (*Oishi [4]*)**.** For $[A] \in \mathbb{IR}^{m\times n}$, the following algorithm computes a center M and radius R of $[A]$ such that $[A] \subseteq \langle M, R\rangle$.

```
function [M, R] = Is2Mr([A])
    setround(1);
    M = A_ + 0.5 * (A¯ − A_);
    R = M − A_;
end
```

We introduce naive interval arithmetic for real interval matrix multiplication $[A][B]$, where $[A] \in \mathbb{IF}^{m\times n}$, $[B] \in \mathbb{IF}^{n\times p}$. Interval arithmetic can be applied to each scalar operation in matrix multiplication straightforwardly (the strategy is called naive interval arithmetic). For example, for $[a], [b] \in \mathbb{IF}$, $[a] + [b]$ is enclosed by

$$[a] + [b] \subseteq [\mathrm{fl}_\nabla(\underline{a} + \underline{b}), \mathrm{fl}_\triangle(\overline{a} + \overline{b})]. \tag{4}$$

A product $[a] * [b]$ is enclosed by

$$[a] * [b] \subseteq [\mathrm{fl}_\nabla(\mathtt{min}(\underline{a} * \underline{b}, \underline{a} * \overline{b}, \overline{a} * \underline{b}, \overline{a} * \overline{b})),\ \mathrm{fl}_\triangle(\mathtt{max}(\underline{a} * \underline{b}, \underline{a} * \overline{b}, \overline{a} * \underline{b}, \overline{a} * \overline{b}))]. \tag{5}$$

Therefore, it is possible to obtain the enclosure of $[A][B]$ by using (4) and (5). But $\mathcal{O}(n^3)$ switches of rounding mode are necessary. Or, for example, the following procedure based on rank-1 updates introduced in Rump's paper [4] is applicable to obtain $[A][B] \subseteq [C]$ with two switches of rounding mode:

```
C¯ = zeros(m, p);
C_ = zeros(m, p);
setround(−1);
for i = 1 : n
    C_ = C_ + min(A_(:, i) * B_(i, :), A_(:, i) * B¯(i, :),
                  A(:, i) * B_(i, :), A¯(:, i) * B¯(i, :));
end
setround(1);
for i = 1 : n
    C¯ = C¯ + max(A_(:, i) * B_(i, :), A_(:, i) * B¯(i, :),
                  A(:, i) * B_(i, :), A¯(:, i) * B¯(i, :));
end
[C] = infsup(C_, C¯);
```

The number of switching rounding modes in the above-mentioned approach is much less than that in naive interval arithmetic. In particular, this algorithm is fast in terms of interpretation overhead in MATLAB. Naive interval computations have an advantage of tightness of the interval, compared to midpoint–radius computations introduced later. However, it takes much computing time compared to pure floating-point matrix multiplication. There are the following two reasons. First, the algorithm includes many branches for taking the maximum and the minimum. Next reason is that optimized BLAS (Basic Linear Algebra Subprograms) is usually used for computations of pure floating-point matrix multiplication. For example, GotoBLAS [14], Intel Math Kernel Library and ATLAS [15] are well-known as optimized BLAS. The routines of matrix multiplication are highly optimized for various architectures, so that the computational performance is nearly peak. However, such BLAS-3 routines cannot be applied for these interval computations. This is a principal problem of any implementation of naive interval arithmetic for interval matrix multiplication.

2.2. Oishi–Rump's algorithm

Oishi and Rump [16,17,4] promote a fast algorithm for interval matrix multiplication. The midpoint–radius form is useful for interval matrix multiplication since we can exploit fast routines for matrix multiplication, for example, xGEMM in BLAS. For $[A] \in \mathbb{IF}^{m\times n}$ and $[B] \in \mathbb{IF}^{n\times p}$, we first execute

$$[M_A, R_A] = \mathtt{Is2Mr}([A]), \qquad [M_B, R_B] = \mathtt{Is2Mr}([B]) \tag{6}$$

for converting inf–sup form to mid–rad form. Let rC be

$$rC = \mathrm{fl}_\triangle(|M_A|R_B + R_A(|M_B| + R_B)).$$

Interval matrix multiplication $[A][B]$ can be enclosed by floating-point arithmetic as follows:

$$[A][B] \subseteq [M_A - R_A, M_A + R_A][M_B - R_B, M_B + R_B] \tag{7}$$

$$\subseteq [\mathrm{fl}_\nabla(M_AM_B - rC), \mathrm{fl}_\Delta(M_AM_B + rC)] =: [W]. \tag{8}$$

This means that four matrix products

$$\mathrm{fl}_\nabla(M_AM_B), \qquad \mathrm{fl}_\Delta(M_AM_B), \qquad \mathrm{fl}_\Delta(|M_A|R_B), \qquad \mathrm{fl}_\Delta(R_A(|M_B| + R_B)) \tag{9}$$

suffice for the evaluation of an inclusion of $[A][B]$. The total computational cost of evaluation (8) is $8mnp + \mathcal{O}(mp) + \mathcal{O}(np)$ flops.[1] The following is an algorithm outputting $[W]$ in (8).

Algorithm 2 (*Oishi–Rump [17,8]*)**.** For $[A] \in \mathbb{IF}^{m\times n}$ and $[B] \in \mathbb{IF}^{n\times p}$, the following algorithm computes an enclosure of $[A][B]$, i.e., $[A][B] \subseteq [C]$. It involves four matrix products.

```
function [C] = midrad_mul([A], [B])
    [M_A, R_A] = Is2Mr([A]);
    [M_B, R_B] = Is2Mr([B]);
    setround(1);
    R = abs(M_A) * R_B + R_A * (abs(M_B) + R_B);
    C̄ = M_A * M_B + R;
    setround(−1);
    C̲ = M_A * M_B − R;
    [C] = infsup(C̲, C̄);    %[C] = [W] in (8)
end
```

Interval matrix multiplication by midpoint–radius representation is well-known; see e.g. [20,21]. However, until INTLAB [4] it was not used in interval libraries because of fear of overestimation. In [17], Rump showed that the overestimation is usually small, and that it is globally limited to 50% in the worst case. Therefore, it is used in INTLAB because the performance compared naive interval arithmetic increases by orders of magnitude.

2.3. Ogita–Oishi's algorithm

We introduce an algorithm for interval matrix multiplication in [9]. They used the fact that all elements in $|M_A|$, R_B, R_A and $(|M_B| + R_B)$ are nonnegative. Therefore, upper bounds for $|M_A|R_B$ and $R_A(|M_B| + R_B)$ can be obtained without full matrix multiplication. For $X \in \mathbb{F}^{m\times n}$ and $Y \in \mathbb{F}^{n\times p}$, they developed a fast algorithm computing an upper bound of $|X||Y|$. Let $s, t \in \mathbb{F}^n$ denote

$$s_j = \max_{1\le i\le m} |x_{ij}|, \qquad t_i = \max_{1\le j\le p} |y_{ij}|. \tag{10}$$

Then,

$$\mathrm{fl}_\Delta(|X||Y|) \le \mathrm{fl}_\Delta\left(\min(e\cdot(s^T\cdot|Y|), (|X|\cdot t)\cdot f^T)\right), \tag{11}$$

where $e = (1, 1, \ldots, 1)^T \in \mathbb{R}^m$ and $f = (1, 1, \ldots, 1)^T \in \mathbb{R}^p$. The computational cost for (10) and (11) is $\mathcal{O}(mn) + \mathcal{O}(np) + \mathcal{O}(mp)$ flops compared to $2mnp$ flops for $\mathrm{fl}_\Delta(|X||Y|)$. The following implements algorithm (11).

Algorithm 3 (*Ogita and Oishi [22]*)**.** For floating-point matrices $X \in \mathbb{F}^{m\times n}$ and $Y \in \mathbb{F}^{n\times p}$ with $X \ge 0$ and $Y \ge 0$, the following algorithm outputs an upper bound U of XY with $\mathcal{O}(mn) + \mathcal{O}(mp) + \mathcal{O}(np)$ flops.

```
function U = fastNN(X, Y)
    n = size(X, 2);
    setround(1);
    p1 = X * max(Y, [ ], 2);    % X * v,  v(i, 1) = max_{1≤j≤p} Y_ij
    p2 = max(X) * Y;    % w * Y,  w(1, j) = max_{1≤i≤m} X_ij
    U = min(repmat(p1, 1, n), repmat(p2, n, 1));
end
```

[1] The notation 'flops' means the number of floating-point operations. It is frequently used in [18,19]. Note that it does not mean 'floating-point number operations per second' in this paper.

All elements in the result by $\texttt{fastNN}(X, Y)$ bound those of $\mathrm{fl}_\triangle(|X| * |Y|)$ from above. If the elements in $|X|$ and $|Y|$ are of similar order of magnitude, then the overestimation of the upper bounds by Algorithm 3 is not so serious. By using Algorithm 3, the number of matrix products in (9) can be reduced since upper bounds of two matrix products are computed by

$$\mathrm{fl}_\triangle(|M_A|R_B) \le \texttt{fastNN}(|M_A|, R_B) := K_1,$$
$$\mathrm{fl}_\triangle(R_A(|M_B| + R_B)) \le \texttt{fastNN}(R_A, \mathrm{fl}_\triangle(|M_B| + R_B)) := K_2.$$

Then, $[A][B]$ is enclosed by

$$[A][B] \subseteq [\mathrm{fl}_\triangledown(M_A M_B) - K_1 - K_2,\ \mathrm{fl}_\triangle(M_A M_B) + K_1 + K_2].$$

In total, Ogita–Oishi's algorithm involves two matrix products for interval matrix multiplication. We write Ogita–Oishi's algorithm as follows.

Algorithm 4 (*Ogita and Oishi [9]*)**.** For matrices $[A] \in \mathbb{IF}^{m\times n}$ and $[B] \in \mathbb{IF}^{n\times p}$, the following algorithm returns an enclosure $[C] \in \mathbb{IF}^{m\times p}$ of $[A][B]$:

```
function [C] = fastInReII([A], [B])
    [M_A, R_A] = Is2Mr([A]);
    [M_B, R_B] = Is2Mr([B]);
    setround(−1);
    T_ = M_A * M_B;
    setround(1);
    T¯ = M_A * M_B;
    M = T_ + 0.5 * (T¯ − T_);
    R = M − T_ + fastNN(abs(M_A), R_B) + fastNN(R_A, abs(M_B) + R_B);
    [C] = midrad(M, R);    %[W] ⊆ [C]
end
```

3. Proposed algorithms

In this section, we develop three algorithms for computing an interval matrix product. The difference among our algorithms is the number of matrix products and tightness of the result interval. In Section 3.1, we introduce a technique which accelerates interval computations for special matrices. In Section 3.2, we develop an algorithm which involves only one matrix product for interval matrix multiplication. Next, we investigate algorithms which require two or three matrix products for interval matrix multiplication in turn. Finally, pseudo-codes of the proposed algorithms and numerical examples are given.

3.1. Acceleration of interval arithmetic

Before proposing our algorithms, we suggest a technique accelerating interval computations for special matrices when the magnitude of midpoints is much smaller than the radii in many cases. Let us show an example:

$$M^{(1)} := \begin{pmatrix} 100 & 1 & 1 \\ 1 & 1 & 100 \\ 1 & 100 & 1 \end{pmatrix}, \qquad R^{(1)} := \begin{pmatrix} 10 & 10 & 10 \\ 10 & 10 & 10 \\ 10 & 10 & 10 \end{pmatrix}.$$
$$M^{(2)} := \begin{pmatrix} 2 & 1 & 2 \\ 1 & 1 & 2 \\ 1 & 2 & 1 \end{pmatrix}, \qquad R^{(2)} := \begin{pmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{pmatrix}.$$

Then,

$$\langle M^{(1)}, R^{(1)}\rangle \cdot \langle M^{(2)}, R^{(2)}\rangle = \begin{pmatrix} [148.8, 255.2] & [49.8, 156.2] & [139.8, 266.2] \\ [49.8, 156.2] & [148.8, 255.2] & [40.8, 167.2] \\ [49.8, 156.2] & [49.8, 156.2] & [139.8, 266.2] \end{pmatrix}. \tag{12}$$

We simplify this interval matrix multiplication. If $|M_{ij}^{(1)}|$ is sufficiently smaller than $R_{ij}^{(1)}$, we increase the radius and make the center zero, namely

```
if α|M^(1)|_ij ≤ R^(1)_ij
        R^(1)'_ij = R^(1)_ij + |M^(1)|_ij;
        M^(1)'_ij = 0;
else
        M^(1)'_ij = M^(1)_ij
        R^(1)'_ij = R^(1)_ij
end
```

For $\alpha \geq 10$, this results in

$$M^{(1)'} := \begin{pmatrix} 100 & 0 & 0 \\ 0 & 0 & 100 \\ 0 & 100 & 0 \end{pmatrix}, \qquad R^{(1)'} := \begin{pmatrix} 10 & 11 & 11 \\ 11 & 11 & 10 \\ 11 & 10 & 11 \end{pmatrix}.$$

Therefore, we obtain $\langle M^{(1)}, R^{(1)} \rangle \subseteq \langle M^{(1)'}, R^{(1)'} \rangle$ and

$$\langle M^{(1)'}, R^{(1)'} \rangle \cdot \langle M^{(2)}, R^{(2)} \rangle = \begin{pmatrix} [144.8, 255.2] & [43.8, 156.2] & [133.8, 266.2] \\ [43.8, 156.2] & [144.8, 255.2] & [32.8, 167.2] \\ [43.8, 156.2] & [43.8, 156.2] & [133.8, 266.2] \end{pmatrix}. \tag{13}$$

Note that result (13) is not so overestimated compared to (12). In addition, $M^{(1)}$ becomes a sparse matrix in this case. Generally, if the number of zero elements is sufficiently large, then it is expected that a calculation of sparse matrix multiplication is faster than that of dense matrix multiplication. We define β such that if the amount of nonzero elements in a matrix is less than $100*\beta$ percent, the sparse computations work faster than the dense computations.[2] The performance of interval computations can be accelerated without much overestimation of the width for such special interval matrices. If α is sufficiently large (at least α should be greater than 10), then the overestimation of the width is not serious (see Theorem 1 in details). We now give the following algorithm based on above-mentioned discussion.

Algorithm 5. If $\mathrm{fl}_{\triangle}(\alpha|(M_A)_{ij}|) \leq (R_A)_{ij}$ are satisfied in more than $\beta * m * n$ pairs of (i, j), then we slightly expand the corresponding radii and adopt a sparse form as a representation of M_A.

```
function [M_A, R_A] = SP_check(M_A, R_A, α, β)
        [m, n] = size(M_A);
        D = abs(M_A);
        setround(1);
        T = (α * D ≤ R_A);
        if nnz(T) > β * m * n
                R_A(T) = R_A(T) + D(T);
                M_A(T) = 0;
                M_A = sparse(M_A);
        end
end
```

If Algorithm 5 changes both M_A and M_B into the sparse forms, the intervals become little wider. Therefore, upper bounds of $|M_A|R_B$ and $R_A(|M_B| + R_B)$ in Algorithms 2 and 4 are also overestimated. However, this expansion seems to be negligible for large α. We explain it by the following theorem.

Theorem 1. *If we apply*

$$[M_A', R_A'] = \mathtt{SP\_check}(M_A, R_A, \alpha, \beta), \qquad [M_B', R_B'] = \mathtt{SP\_check}(M_B, R_B, \alpha, \beta),$$

[2] The constant β depends on computational environments.

then

$$R'_A(|M'_B| + R'_B) \le \left\{(1+2\mathbf{u})\left(1+\frac{1}{\alpha}\right)\right\}^2 R_A(|M_B| + R_B),$$
$$|M'_A|R'_B \le (1+2\mathbf{u})\left(1+\frac{1}{\alpha}\right)|M_A|R_B$$

are satisfied. Here $\mathbf{u}$ *denotes the unit roundoff. For the IEEE* 754 *standard* [12,13], $\mathbf{u}$ *is* 2^{-53} *for double precision (binary* 64*) floating-point arithmetic. For single precision (binary* 32*),* $\mathbf{u}$ *is* 2^{-24}.

Proof. From Algorithm 5, the following two relations can be obtained:

$$\mathrm{fl}_\triangle(\alpha|M_A|_{ij}) \le (R_A)_{ij} \Longrightarrow (R'_A)_{ij} := \mathrm{fl}_\triangle((R_A)_{ij} + |M_A|_{ij})$$
$$\mathrm{fl}_\triangle(\alpha|M_A|_{ij}) > (R_A)_{ij} \Longrightarrow (R'_A)_{ij} := (R_A)_{ij}.$$

In either case, we obtain

$$R'_A \le \mathrm{fl}_\triangle\left(R_A + \frac{1}{\alpha}R_A\right), \qquad |M'_A| \le |M_A|.$$

Similarly, we have

$$R'_B \le \mathrm{fl}_\triangle\left(R_B + \frac{1}{\alpha}R_B\right), \qquad |M'_B| \le |M_B|.$$

For $a, b \in \mathbb{F}$, the definition of floating-point arithmetic by the IEEE 754 standard yields

$$\mathrm{fl}_\triangle(a+b) \le (1+2\mathbf{u})|a+b|.$$

First, we derive an upper bound of $R'_A(|M'_B| + R'_B)$:

$$\begin{aligned} R'_A(|M'_B| + R'_B) &\le \mathrm{fl}_\triangle\left(R_A + \frac{1}{\alpha}R_A\right)\left(|M_B| + \mathrm{fl}_\triangle\left(R_B + \frac{1}{\alpha}R_B\right)\right) \\ &\le (1+2\mathbf{u})\left(R_A + \frac{1}{\alpha}R_A\right)\left(|M_B| + (1+2\mathbf{u})\left(R_B + \frac{1}{\alpha}R_B\right)\right) \\ &= (1+2\mathbf{u})\left(1+\frac{1}{\alpha}\right)R_A \cdot \left(|M_B| + (1+2\mathbf{u})\left(1+\frac{1}{\alpha}\right)R_B\right) \\ &\le \left((1+2\mathbf{u})\left(1+\frac{1}{\alpha}\right)\right)^2 R_A(|M_B| + R_B). \end{aligned}$$

Next, we take an upper bound of $|M'_A|R'_B$ as follows:

$$\begin{aligned} |M'_A|R'_B &\le |M_A|\mathrm{fl}_\triangle\left(R_B + \frac{1}{\alpha}R_B\right) \\ &\le (1+2\mathbf{u})|M_A|\left(R_B + \frac{1}{\alpha}R_B\right) \\ &\le (1+2\mathbf{u})\left(1+\frac{1}{\alpha}\right)|M_A|R_B. \end{aligned}$$

This completes the proof. □

We have shown that the upper bounds of $R'_A(|M'_B| + R'_B)$ and $|M'_A|R'_B$ are not significantly overestimated by Algorithm 5. We cannot derive

$$[\mathrm{fl}_\triangledown(M'_AM'_B), \mathrm{fl}_\triangle(M'_AM'_B)] \subseteq [\mathrm{fl}_\triangledown(M_AM_B), \mathrm{fl}_\triangle(M_AM_B)].$$

However, if α is large, this is not a serious problem. The reason is as follows. There exist $D_A \in \mathbb{F}^{m\times n}$ and $D_B \in \mathbb{F}^{n\times p}$ which satisfy

$$M'_A = M_A + D_A, \qquad M'_B = M_B + D_B, \qquad |D_A| \le \frac{1}{\alpha}R_A, \qquad |D_B| \le \frac{1}{\alpha}R_B.$$

Then, $|M'_A M'_B - M_A M_B|$ can be bounded by

$$\begin{aligned} |M'_A M'_B - M_A M_B| &= |(M_A + D_A)(M_B + D_B) - M_A M_B| \\ &= |M_A D_B + D_A M_B + D_A D_B| \\ &\le |M_A||D_B| + |D_A||M_B| + |D_A||D_B| \\ &\le \frac{1}{\alpha}|M_A|R_B + \frac{1}{\alpha}R_A|M_B| + \frac{1}{\alpha^2}R_A R_B. \end{aligned} \tag{14}$$

We evaluate $|M_A|R_B$ and $R_A(|M_B| + R_B)$ in (8). Therefore, when α is large, first and second terms in (14) are smaller than them. Moreover, $\frac{1}{\alpha^2}R_A R_B$ is negligible.

Now, we consider two forms for the enclosure of $[A][B]$.

$$[A][B] \subseteq [M_A M_B - |M_A|R_B - R_A(|M_B| + R_B),\ M_A M_B + |M_A|R_B + R_A(|M_B| + R_B)] \tag{$*$}$$

$$[A][B] \subseteq [M_A M_B - R_A|M_B| - (|M_A| + R_A)R_B,\ M_A M_B + R_A|M_B| + (|M_A| + R_A)R_B]. \tag{$**$}$$

Form ($*$) has already been introduced in Section 3.2. Based on ($**$), the similar algorithms can be developed. As a strategy by Oishi and Rump, if M_A is a sparse matrix while M_B is not a sparse matrix, then it is better to compute interval matrix multiplication by ($*$) in terms of computing time. If M_B is a sparse matrix while M_A is not a sparse matrix, then it is better to compute it by ($**$) in terms of computing time. As for strategy by Ogita and Oishi, there is almost no difference in computing time between the use of ($*$) and ($**$). Accuracy of a result by using ($*$) is almost the same as that by using ($**$) for both algorithms. From the next subsection, we develop algorithms based on ($*$). Similar discussions apply to ($**$).

3.2. Algorithm with one matrix product

We suggest an algorithm which involves only one matrix product for interval matrix multiplication. First, we introduce an a priori error bound for pure floating-point matrix multiplication. For $X \in \mathbb{F}^{m\times n}$ and $Y \in \mathbb{F}^{n\times p}$, if no underflow occurs, it holds from [19] that

$$|XY - \mathrm{fl}(XY)| \le \gamma_n |X||Y|, \quad \gamma_n = \frac{n\mathbf{u}}{1 - n\mathbf{u}},\ n \in \mathbb{N},\ n < \mathbf{u}^{-1}.$$

Let $\overline{\gamma}_n$ be $\mathrm{fl}_\triangle(n\mathbf{u}/(1 - n\mathbf{u}))$. It follows that

$$\begin{aligned} &XY \subseteq \langle \mathrm{fl}(XY),\ \gamma_n|X||Y|\rangle \subseteq \mathtt{midrad}(\mathrm{fl}(XY),\ \mathrm{fl}_\triangle(\overline{\gamma}_n|X||Y|)), \\ &XY \subseteq \mathtt{infsup}(\mathrm{fl}_\nabla(\mathrm{fl}(XY) - \mathrm{fl}_\triangle(\overline{\gamma}_n|X||Y|)),\ \mathrm{fl}_\triangle(\mathrm{fl}(XY) + \overline{\gamma}_n|X||Y|)). \end{aligned} \tag{15}$$

Let S_1, S_2 and S_3 denote

$$\begin{aligned} S_1 &= \mathtt{rad}(\mathtt{infsup}(\mathrm{fl}_\nabla(M_A M_B),\ \mathrm{fl}_\triangle(M_A M_B))), \\ S_2 &= \mathrm{fl}_\triangle(|M_A|R_B), \\ S_3 &= \mathrm{fl}_\triangle(R_A(|M_B| + R_B)). \end{aligned} \tag{16}$$

Here, $\mathtt{rad}(\ldots)$ returns a radius of an interval in the parentheses.[3] Let T_1, T_2 and T_3 denote

$$\begin{aligned} T_1 &:= \mathrm{fl}_\triangle(\overline{\gamma}_n * \mathtt{fastNN}(|M_A|,\ |M_B|)), \\ T_2 &:= \mathtt{fastNN}(|M_A|,\ R_B), \\ T_3 &:= \mathtt{fastNN}(R_A,\ \mathrm{fl}_\triangle(|M_B| + R_B)). \end{aligned} \tag{17}$$

Form (7) is enclosed by

$$\langle M_A M_B,\ |M_A|R_B + R_A(|M_B| + R_B)\rangle. \tag{18}$$

Since (15), $|M_A|R_B \le S_2 \le T_2$ and $R_A * (|M_B| + R_B) \le S_3 \le T_3$, interval (18) can be enclosed by

$$\mathtt{midrad}(\mathrm{fl}(M_A M_B),\ \mathrm{fl}_\triangle(T_1 + T_2 + T_3)) =: [Y]. \tag{19}$$

The evaluation of (19) involves only one matrix product $\mathrm{fl}(M_A M_B)$ in rounding to nearest. An algorithm evaluating (19) is denoted by

$$[C] = \mathtt{IMM1}([A],\ [B],\ \alpha,\ \beta).$$

The detail of the function $\mathtt{IMM1}([A],\ [B],\ \alpha,\ \beta)$ is described in Section 3.5. The total computational cost of this algorithm is $2mnp + \mathcal{O}(mn) + \mathcal{O}(mp) + \mathcal{O}(np)$ flops. Note that the radius of the interval $[W]$ is nearly $S_1 + S_2 + S_3$, so that $[Y]$ in (19) is a superset of $[W]$.

[3] The result may be larger than the true radius since the true radius is not a floating-point number.

3.3. Algorithm with two matrix products

Recall that the radius of a result by $\texttt{IMM1}([A], [B], \alpha, \beta)$ is $\mathrm{fl}_\triangle(T_1+T_2+T_3)$. First, T_1, T_2 and T_3 in (17) are evaluated. To find out which becomes a dominant radius in $[Y]$ in (19) with cheap cost, all elements in T_1, T_2 and T_3 are added in floating-point arithmetic, respectively, i.e.

$$m_1 = \mathrm{fl}\left(\sum_{i=1}^{m}\sum_{j=1}^{p}(T_1)_{ij}\right), \qquad m_2 = \mathrm{fl}\left(\sum_{i=1}^{m}\sum_{j=1}^{p}(T_2)_{ij}\right), \qquad m_3 = \mathrm{fl}\left(\sum_{i=1}^{m}\sum_{j=1}^{p}(T_3)_{ij}\right). \tag{20}$$

In MATLAB notation, $m_1 = \texttt{sum}(T_1(:))$. It is also possible to use other criteria such as the maximum elements in T_1, T_2 and T_3

$$\texttt{max}(T_1(:)), \qquad \texttt{max}(T_2(:)), \qquad \texttt{max}(T_3(:)),$$

or the maximum norm

$$\texttt{norm}(T_1, \texttt{inf}), \qquad \texttt{norm}(T_2, \texttt{inf}), \qquad \texttt{norm}(T_3, \texttt{inf})$$

as an alternative for m_1, m_2 and m_3. Remark that there is no matrix multiplication until now. If m_1 is the largest of the three, our algorithm spends a good amount of effort to avoid overestimation of the enclosure of M_AM_B. Namely, $[A][B]$ is enclosed by

$$[\mathrm{fl}_\triangledown(M_AM_B - T_2 - T_3),\ \mathrm{fl}_\triangle(M_AM_B + T_2 + T_3)]. \tag{21}$$

Here, two matrix products $\mathrm{fl}_\triangle(M_AM_B)$ and $\mathrm{fl}_\triangledown(M_AM_B)$ appear in (21). If m_2 is the largest, T_2 seems to become the dominant radius in $[Y]$. Let G be $\mathrm{fl}_\triangle(|M_A|(\overline{\gamma}_n|M_B| + R_B))$. Then, we have

$$\gamma_n|M_A||M_B| + |M_A|R_B \le G.$$

Therefore, $[A][B]$ is enclosed by $\langle \mathrm{fl}(M_AM_B), \mathrm{fl}_\triangle(G+T_3)\rangle$. There are two matrix products: $\mathrm{fl}(M_AM_B)$ and $\mathrm{fl}_\triangle(|M_A|(\gamma_n|M_B|+R_B))$. If m_3 is the largest, AB is enclosed by using S_3 instead of T_3:

$$\langle \mathrm{fl}(M_A * M_B), T_1 + T_2 + S_3\rangle \tag{22}$$

In (22), there are two matrix products: $\mathrm{fl}(M_AM_B)$ and $\mathrm{fl}_\triangle(R_A(|M_B| + R_B))$. In any case, our algorithm involves two matrix products. This algorithm is denoted by

$$[C] = \texttt{IMM2}([A], [B], \alpha, \beta),$$

and is described in detail in Section 3.5. If one of m_2 and m_3 is large compared to the other two, it is expected that this algorithm outputs a tighter interval than Algorithm 4. If the orders of the magnitude in m_1, m_2 and m_3 are almost the same, this algorithm can hardly improve the tightness of the interval in the sense of average.

Observation 1. *For $[A] \in \mathbb{IR}^{m\times n}$ and $[B] \in \mathbb{IR}^{n\times p}$, two algorithms are executed as follows:*

$$[C_3] = \texttt{fastInReII}([A], [B]), \qquad [C_4] = \texttt{IMM2}([A], [B], \alpha, \beta).$$

If m_1 is the largest of the three, $[C_4] = [C_3]$ is satisfied. If m_2 is the largest, then we have

$$\begin{aligned} &\texttt{rad}([C_3]) \approx S_1 + T_2 + T_3, \\ &\texttt{rad}([C_4]) \approx |M_A|(\gamma_n|M_B| + R_A) + T_3 \approx \gamma_n|M_A||M_B| + S_2 + T_3. \end{aligned} \tag{23}$$

Since T_2 seems to be a dominant term and $S_2 \le T_2$ is satisfied, we can expect $\texttt{rad}([C_3]) > \texttt{rad}([C_4])$. However, it is not guaranteed because we may obtain $S_2 = T_2$ in the worst case. Then, from $S_1 \le \gamma_n|M_A||M_B|$, $\texttt{rad}([C_3]) < \texttt{rad}([C_4])$ may be satisfied. If m_3 is the largest, then we have

$$\texttt{rad}([C_4]) \approx \gamma_n\texttt{fastNN}(|M_A|, |M_B|) + T_2 + S_3.$$

Since T_3 seems to be the dominant radius in this case and $S_3 \le T_3$, we expect $\texttt{rad}([C_4]) < \texttt{rad}([C_3])$. However, if $S_3 = T_3$ may be satisfied, then we may obtain $\texttt{rad}([C_3]) < \texttt{rad}([C_4])$.

3.4. Algorithm with three matrix products

Next, we develop an algorithm which involves three matrix products for interval matrix multiplication. First, we compute T_1, T_2 and T_3 in (17) and m_1, m_2 and m_3 in (20). Our algorithm predicts which is the minimum of the three (m_1, m_2 and m_3). If m_1 is the smallest, the resultant interval is obtained by $\langle \mathrm{fl}(M_AM_B), \mathrm{fl}_\triangle(G + S_3)\rangle$. If m_2 is the smallest, then a result is obtained by

$$[\mathrm{fl}_\triangledown(M_AM_B - T_2 - S_3),\ \mathrm{fl}_\triangle(M_AM_B + T_2 + S_3)].$$

Otherwise (m_3 is the smallest), our algorithm outputs

$$[\mathrm{fl}_\nabla(M_AM_B - S_2 - T_3),\ \mathrm{fl}_\Delta(M_AM_B + S_2 + T_3)].$$

In any case, the proposed algorithm involves three matrix products. This algorithm is denoted by

$$[C] = \mathtt{IMM3}([A], [B], \alpha, \beta).$$

If one of m_1, m_2 and m_3 is small compared to others, this algorithm may output tighter intervals than Algorithm 4 and the algorithms proposed in Sections 3.2 and 3.3. If the orders of magnitude of m_1, m_2 and m_3 are almost the same, this algorithm cannot efficiently improve the tightness of the interval.

Observation 2. *For $[A] \in \mathbb{IR}^{m\times n}$ and $[B] \in \mathbb{IR}^{n\times p}$, two algorithms are executed as follows:*

$$[C_3] = \mathtt{fastInReII}([A], [B]), \qquad [C_4] = \mathtt{IMM3}([A], [B], \alpha, \beta).$$

If m_2 is the smallest, then the radius of $[C_4]$ is approximately

$$\mathtt{rad}([C_4]) \approx S_1 + T_2 + S_3.$$

If m_3 is the smallest, then the radius of $[C_3]$ is nearly

$$\mathtt{rad}([C_4]) \approx S_1 + S_2 + T_3.$$

From little considerations, $\mathtt{rad}([C_4]) \le \mathtt{rad}([C_3])$ is satisfied when m_2 or m_3 is the smallest. If m_1 is the smallest, then it holds that

$$\mathtt{rad}([C_4]) \approx |M_A|(\gamma_n M_B + R_B) + S_3 \approx \gamma_n |M_A||M_B| + S_2 + S_3.$$

Therefore, we can expect $\mathtt{rad}([C_4]) \le \mathtt{rad}([C_3])$. In the worst case, there exist cases: $T_2 = S_2$ and $T_3 = S_3$. If $S_1 \le \gamma_n|M_A||M_B|$ is satisfied, then $\mathtt{rad}([C_3]) \le \mathtt{rad}([C_4])$ may be satisfied.

3.5. Pseudo-codes

We give pseudo-codes of algorithms developed in Sections 3.2–3.4. All pseudo-codes work in INTLAB [4] almost as is. We explain arguments of the functions. Let $[A] \in \mathbb{F}^{m\times n}$ and $[B] \in \mathbb{F}^{n\times p}$. Two constants α and β are arguments for Algorithm 5. We define a function $\mathtt{gamma}(n)$ which computes an upper bound of γ_n. First, we give an algorithm calculating common factors in the proposed algorithms.

Algorithm 6. The following algorithm outputs common elements used in proposed algorithms.

```
function [M_A, R_A, M_B, R_B, T] = Common([A], [B], α, β)
    [M_A, R_A] = Is2Mr([A]);
    [M_B, R_B] = Is2Mr([B]);
    [M_A, R_A] = SP_check(M_A, R_A, α, β);
    [M_B, R_B] = SP_check(M_B, R_B, α, β);
    n = size(A, 2);
    setround(1);
    T{1} = gamma(n) * fastNN(abs(M_A), abs(M_B));
    if issparse(M_B) == 1
        T{2} = fastNN(R_A, abs(M_B));
        T{3} = fastNN(abs(M_A) + R_A, R_B);
    else
        T{2} = fastNN(abs(M_A), R_B);
        T{3} = fastNN(R_A, abs(M_B) + R_B);
    end
end
```

Next, we show the source code of algorithms discussed in Sections 3.2–3.4.

Algorithm 7. For matrices $[A] \in \mathbb{IF}^{m\times n}$ and $[B] \in \mathbb{IF}^{n\times p}$, the following algorithm returns an enclosure $[C] \in \mathbb{IR}^{m\times p}$ of $[A][B]$ with one matrix product:

```
function [C] = IMM1([A], [B], α, β)
    [M_A, R_A, M_B, R_B, T] = Common([A], [B], α, β);
    setround(0);
    M = M_A * M_B;
    setround(1);
    [C] = midrad(M, T{1} + T{2} + T{3});
end
```

Algorithm 8. For matrices $[A] \in \mathbb{IF}^{m\times n}$ and $[B] \in \mathbb{IF}^{n\times p}$, the following algorithm returns an enclosure $[C] \in \mathbb{IR}^{m\times p}$ of $[A][B]$ with two matrix products:

```
function [C] = IMM2([A], [B], α, β)
    [M_A, R_A, M_B, R_B, T] = Common([A], [B], α, β);
    me(1) = sum(sum(T{1}));
    me(2) = sum(sum(T{2}));
    me(3) = sum(sum(T{3}));
    [temp, i] = max(me);
    if i == 1
        setround(−1);
        Qd = M_A * M_B − T{2} − T{3};
        setround(1);
        Qu = M_A * M_B + T{2} + T{3};
        [C] = infsup(Qd, Qu);
    else
        setround(0);
        M = M_A * M_B;
        setround(1);
        if issparse(M_B) == 1
            if i == 2
                T{2} = abs(M_A) * (gamma(n) * abs(M_B) + R_B);
                [C] = midrad(M, T{2} + T{3});
            else
                T{3} = R_A * (abs(M_B) + R_B);
                [C] = midrad(M, T{1} + T{2} + T{3});
            end
        else
            if i == 2
                T{2} = R_A * abs(M_B);
                [C] = midrad(M, T{1} + T{2} + T{3});
            else
                T{3} = (gamma(n) * abs(M_A) + R_A) * R_B;
                [C] = midrad(M, T{2} + T{3});
            end
        end
    end
end
```

Algorithm 9. For matrices $[A] \in \mathbb{IF}^{m\times n}$ and $[B] \in \mathbb{IF}^{n\times p}$, the following algorithm returns an enclosure $[C] \in \mathbb{IR}^{m\times p}$ of $[A][B]$ with three matrix products:

```
function [C] = IMM3([A], [B], α, β)
    [M_A, R_A, M_B, R_B, T] = Common([A], [B], α, β);
    me(1) = sum(sum(T{1}));
    me(2) = sum(sum(T{2}));
    me(3) = sum(sum(T{3}));
    [temp, i] = min(me);
    if issparse(M_B) == 1
        if i == 1
            M = M_A * M_B;
            setround(1);
            T{2} = abs(M_A) * (gamma(n) * abs(M_B) + R_B);
            T{3} = R_A * (abs(M_B) + R_B);
            [C] = midrad(M, T{2} + T{3});
        else
            setround(−1);
            Qd = M_A * M_B;
            setround(1);
            Qu = M_A * M_B;
            [Q] = infsup(Qd, Qu);
            if i == 2
                T{3} = R_A * (abs(M_B) + R_B);
                [C] = midrad(mid([Q]), rad([Q]) + T{2} + T{3});
            else
                T{2} = abs(M_A) * R_B;
                [C] = midrad(mid([Q]), rad([Q]) + T{2} + T{3});
            end
    else
        if i == 1
            M = M_A * M_B;
            setround(1);
            T{2} = R_A * abs(M_B);
            T{3} = (gamma(n) * abs(M_A) + R_A) * R_B;
            [C] = midrad(M, T{2} + T{3});
        else
            setround(−1);
            Qd = M_A * M_B;
            setround(1);
            Qu = M_A * M_B;
            [Q] = infsup(Qd, Qu);
            if i == 2
                setround(1);
                T{3} = (abs(M_A) + R_A) * R_B;
                [C] = midrad(mid([Q]), rad([Q]) + T{2} + T{3});
            else
                T{2} = R_A * abs(M_B);
```

```
                [C] = midrad(mid([Q]), rad([Q]) + T{2} + T{3});
            end
        end
    end
```

3.6. Numerical experiments

We present numerical examples to illustrate the efficiency of the proposed algorithms. First, we generate two point matrices P and Q by

$$P = \texttt{randn}(n), \qquad Q = \texttt{randn}(n), \tag{24}$$

where the $\texttt{randn}(n)$ function generates an n-by-n floating-point matrix whose elements are taken from normal distribution with mean 0 and variance 1. Next, interval matrices $[A]$ and $[B]$ are generated by

$$[A] = [\mathrm{fl}(P - c|P|), \mathrm{fl}(P + c|P|)], \qquad [B] = [\mathrm{fl}(Q - d|Q|), \mathrm{fl}(Q + d|Q|)], \tag{25}$$

where both c and d are positive constants. We compare computational performance and tightness of resultant intervals by the following five algorithms:

- M1: Algorithm 2 ($[C_1] = \texttt{midrad\_mul}([A], [B])$), (4 matrix products)
- M2: Algorithm 7 ($[C_2] = \texttt{IMM1}([A], [B], 100, 0.1)$), (1 matrix product, the proposed algorithm)
- M3: Algorithm 4 ($[C_3] = \texttt{fastInReII}([A], [B])$), (2 matrix products)
- M4: Algorithm 8 ($[C_4] = \texttt{IMM2}([A], [B], 100, 0.1)$), (2 matrix products, the proposed algorithm)
- M5: Algorithm 9 ($[C_5] = \texttt{IMM3}([A], [B], 100, 0.1)$), (3 matrix products, the proposed algorithm).

All examples in this subsection are tested on Intel Xeon X5550 2.67 GHz, MATLAB 2010a and INTLAB version 6. Let l_k, m_k and s_k denote

$$l_k = \texttt{max}(\texttt{max}(\texttt{rad}([C_k])./\texttt{rad}([C_1]))),$$
$$m_k = \texttt{mean}(\texttt{mean}(\texttt{rad}([C_k])./\texttt{rad}([C_1]))).$$

Each item in Tables 1, 3 and 5 shows l_2, l_3, l_4 and l_5 in turn with various c and d. Tables 2, 4 and 6 displays $m_2, \ldots, m_5$ in each item, respectively.[4] We generate matrices with dimension 1000, 5000 and 10 000 by n in (24) for these examples. From Tables 1–6, it can be confirmed that there is a tradeoff between the number of matrix products and tightness of the interval. M1 outputs the tightest interval of all. While both M3 and M4 require two matrix products, the result interval by M4 is narrower than that by M3 in these examples. Although the number of matrix products in M4 and M5 are less than that of M1, M4 and M5 can work as well as M1 in terms of the accuracy in many cases. From Tables 1–6, the result by M2 is not significantly overestimated except $c = d = 10^{-15}$.

Next, we compare the computing times for each algorithm. We generate $[A]$ and $[B]$ in (25) with $c = d = 10^{-15}$. If we take another values for c and d, the computing times are not significantly different. The ratio of elapsed times compared to M1 are shown in Table 7 for various values n. When n is small, we can see that M5 works as well as M1 in terms of computing time although the number of matrix products in M5 is less than that in M1. In these cases, the computations with $\mathcal{O}(n^2)$ flops were not negligible. In addition, MATLAB's interpretation overhead slows down the performance. It is also confirmed that the computing time of M4 is almost the same as M3. From the tables, if n is larger than 2000, computing times of each algorithm are almost proportional to the number of matrix products (Tables 8 and 9).

Remark 1. It is difficult to clarify when the computing times are related to the number of matrix products. In MATLAB, the problem is sometimes the interpretation overhead. On a multi-core architecture, it may happen that a routine for matrix–vector product is not performed by multi-threads although a routine for matrix–matrix product is computed in parallel. In addition, sparse routines are not performed in parallel in MATLAB (at least until 2010a). The ratio of computing times depends on the computational environment.

4. Product of point matrix and interval matrix

We specialize the algorithms proposed in Section 3 into a product of a point and an interval matrix. Let $A \in \mathbb{F}^{m\times n}$ and $[B] \in \mathbb{IF}^{n\times p}$, the purpose is to obtain an enclosure of $A[B]$.[5] As for a practical example, A is an approximate inverse

[4] 1.00 in tables is larger than 1, so that the results by our algorithms are a little overestimated.

Table 1
The maximum ratio of radii ($n = 1000$).

$c \setminus d$		10^{-15}	10^{-10}	10^{-5}	10^{0}	10^{5}
10^{-15}	l_2	215	4.79	4.99	4.85	4.82
	l_3	3.99	4.78	4.80	4.85	4.82
	l_4	3.99	1.00	1.00	1.00	1.00
	l_5	2.49	1.00	1.00	1.00	1.00
10^{-10}	l_2	4.79	4.79	4.88	4.81	4.87
	l_3	4.79	4.79	4.88	4.81	4.87
	l_4	1.01	2.90	1.00	1.00	1.00
	l_5	1.00	1.00	1.00	1.00	1.00
10^{-5}	l_2	4.83	4.84	4.82	4.88	4.80
	l_3	4.83	4.84	4.82	4.88	4.80
	l_4	1.00	1.00	2.91	1.00	1.00
	l_5	1.00	1.00	1.00	1.00	1.00
10^{0}	l_2	4.84	4.83	4.88	4.83	4.86
	l_3	4.84	4.83	4.88	4.83	4.86
	l_4	1.00	1.00	1.00	2.27	1.00
	l_5	1.00	1.00	1.00	1.00	1.00
10^{5}	l_2	4.83	4.83	4.82	4.88	4.76
	l_3	4.83	4.83	4.82	4.88	4.76
	l_4	1.00	1.00	1.00	2.94	1.00
	l_5	1.00	1.00	1.00	1.00	1.00

Table 2
The mean ratio of radii ($n = 1000$).

$c \setminus d$		10^{-15}	10^{-10}	10^{-5}	10^{0}	10^{5}
10^{-15}	m_2	156	4.26	4.24	4.27	4.27
	m_3	3.13	4.25	4.24	4.27	4.27
	m_4	3.13	1.00	1.00	1.00	1.00
	m_5	2.06	1.00	1.00	1.00	1.00
10^{-10}	m_2	4.24	4.26	4.26	4.24	4.24
	m_3	4.24	4.26	4.26	4.24	4.24
	m_4	1.00	2.63	1.00	1.00	1.00
	m_5	1.00	1.00	1.00	1.00	1.00
10^{-5}	m_2	4.26	4.24	4.24	4.24	4.25
	m_3	4.26	4.24	4.24	4.24	4.25
	m_4	1.00	1.00	2.62	2.62	1.00
	m_5	1.00	1.00	1.00	1.00	1.00
10^{0}	m_2	4.25	4.26	4.26	4.25	4.25
	m_3	4.25	4.26	4.26	4.25	4.25
	m_4	1.00	1.00	1.00	2.08	1.00
	m_5	1.00	1.00	1.00	1.00	1.00
10^{5}	m_2	4.25	4.25	4.26	4.25	4.24
	m_3	4.25	4.25	4.26	4.25	4.24
	m_4	1.00	1.00	1.00	2.62	1.00
	m_5	1.00	1.00	1.00	1.00	1.00

matrix of the midpoint of B. This occurs frequently in the solution of linear or nonlinear equations, differential equations and so on.

4.1. Previous research

First, we introduce two algorithms for the purpose. After executing $[M_B, R_B] = \texttt{Is2Mr}([B])$, an enclosure of $A[B]$ is obtained as follows:

$$A[B] \subseteq [AM_B - |A|R_B,\ AM_B + |A|R_B] = \langle AM_B, |A|R_B \rangle.$$

Oishi–Rump's algorithm outputs the interval result as

$$[\mathrm{fl}_{\nabla}(AM_B - Rc),\ \mathrm{fl}_{\triangle}(AM_B + Rc)], \tag{26}$$

where $Rc = \mathrm{fl}_{\triangle}(|A|R_B)$. Evaluation (26) involves three matrix products: $\mathrm{fl}_{\nabla}(AM_B)$, $\mathrm{fl}_{\triangle}(AM_B)$ and $\mathrm{fl}_{\triangle}(|A|R_B)$. The following is an algorithm evaluating (26).

[5] Similar arguments for $[B]A$ can be done.

Table 3
The maximum ratio of radii ($n = 5000$).

$c \setminus d$		10^{-15}	10^{-10}	10^{-5}	10^{0}	10^{5}
10^{-15}	l_2	1025	5.14	5.12	5.12	5.12
	l_3	4.06	5.11	5.12	5.12	5.12
	l_4	4.06	1.00	1.00	1.00	1.00
	l_5	2.53	1.00	1.00	1.00	1.00
10^{-10}	l_2	5.15	5.13	5.13	5.12	5.11
	l_3	5.12	5.13	5.13	5.12	5.11
	l_4	1.02	1.00	1.00	1.00	1.00
	l_5	1.00	1.00	1.00	1.00	1.00
10^{-5}	l_2	5.15	5.11	5.12	5.12	5.11
	l_3	5.15	5.11	5.12	5.12	5.11
	l_4	1.00	1.00	1.00	1.00	1.00
	l_5	1.00	1.00	1.00	1.00	1.00
10^{0}	l_2	5.13	5.12	5.12	5.11	5.11
	l_3	5.13	5.12	5.12	5.11	5.11
	l_4	1.00	1.00	1.00	2.37	1.00
	l_5	1.00	1.00	1.00	1.00	1.00
10^{5}	l_2	5.13	5.10	5.11	5.10	5.11
	l_3	5.13	5.10	5.11	5.10	5.11
	l_4	1.00	1.00	1.00	1.00	1.00
	l_5	1.00	1.00	1.00	1.00	1.00

Table 4
The mean ratio of radii ($n = 5000$).

$c \setminus d$		10^{-15}	10^{-10}	10^{-5}	10^{0}	10^{5}
10^{-15}	m_2	830	4.82	4.79	4.80	4.80
	m_3	3.49	4.80	4.79	4.80	4.80
	m_4	3.49	1.00	1.00	1.00	1.00
	m_5	2.24	1.00	1.00	1.00	1.00
10^{-10}	m_2	4.82	4.81	4.79	4.79	4.79
	m_3	4.79	4.80	4.79	4.79	4.79
	m_4	1.02	2.91	1.00	1.00	1.00
	m_5	1.00	1.00	1.00	1.00	1.00
10^{-5}	m_2	4.80	4.80	4.79	4.80	4.79
	m_3	4.80	4.80	4.79	4.80	4.79
	m_4	1.00	1.00	2.89	1.00	1.00
	m_5	1.00	1.00	1.00	1.00	1.00
10^{0}	m_2	4.80	4.80	4.80	4.79	4.79
	m_3	4.80	4.80	4.80	4.79	4.79
	m_4	1.00	1.00	1.00	2.26	1.00
	m_5	1.00	1.00	1.00	1.00	1.00
10^{5}	m_2	4.79	4.79	4.79	4.79	4.79
	m_3	4.79	4.79	4.79	4.79	4.79
	m_4	1.00	1.00	1.00	1.00	1.00
	m_5	1.00	1.00	1.00	1.00	1.00

Algorithm 10 (*Oishi–Rump [17,8]*)**.** For $A \in \mathbb{F}^{m\times n}$ and $[B] \in \mathbb{IF}^{n\times p}$, the following algorithm computes an enclosure of $A[B]$, i.e., $A[B] \subseteq [C]$. It involves three matrix products.

```
function [C] = midrad_PI(A, [B])
    [M_B, R_B] = Is2Mr([B]);
    setround(1);
    R_1 = abs(A) * R_B;
    C̄ = A * M_B + R_1;
    setround(−1);
    C̲ = A * M_B − R_1;
    [C] = infsup(C̲, C̄);
end
```

Table 5
The maximum ratio of radii ($n = 10\,000$).

$c \setminus d$		10^{-15}	10^{-10}	10^{-5}	10^{0}	10^{5}
10^{-15}	l_2	2040	5.30	5.27	5.24	5.24
	l_3	4.09	5.25	5.27	5.24	5.24
	l_4	4.09	1.01	1.00	1.00	1.00
	l_5	2.55	1.00	1.00	1.00	1.00
10^{-10}	l_2	5.30	5.29	5.27	5.25	5.27
	l_3	5.24	5.26	5.27	5.25	5.27
	l_4	1.05	3.14	1.00	1.00	1.00
	l_5	1.00	1.00	1.00	1.00	1.00
10^{-5}	l_2	5.25	5.24	5.25	5.26	5.27
	l_3	5.25	5.24	5.25	5.26	5.27
	l_4	1.00	1.00	3.12	1.00	1.00
	l_5	1.00	1.00	1.00	1.00	1.00
10^{0}	l_2	5.26	5.25	5.26	5.23	5.25
	l_3	5.26	5.25	5.26	5.23	5.25
	l_4	1.00	1.00	1.00	2.41	1.00
	l_5	1.00	1.00	1.00	1.00	1.00
10^{5}	l_2	5.25	5.26	5.25	5.25	5.25
	l_3	5.25	5.26	5.25	5.25	5.25
	l_4	1.00	1.00	1.00	1.00	1.00
	l_5	1.00	1.00	1.00	1.00	1.00

Table 6
The mean ratio of radii ($n = 10\,000$).

$c \setminus d$		10^{-15}	10^{-10}	10^{-5}	10^{0}	10^{5}
10^{-15}	m_2	1725	5.06	5.01	5.01	5.01
	m_3	3.63	5.01	5.01	5.01	5.01
	m_4	3.63	1.01	1.00	1.00	1.00
	m_5	2.31	1.00	1.00	1.00	1.00
10^{-10}	m_2	5.07	5.04	5.01	5.01	5.01
	m_3	5.01	5.01	5.01	5.01	5.01
	m_4	1.05	3.01	1.00	1.00	1.00
	m_5	1.00	1.00	1.00	1.00	1.00
10^{-5}	m_2	5.01	5.01	5.01	5.01	5.01
	m_3	5.01	5.01	5.01	5.01	5.01
	m_4	1.00	1.00	3.00	1.00	1.00
	m_5	1.00	1.00	1.00	1.00	1.00
10^{0}	m_2	5.01	5.01	5.01	5.01	5.01
	m_3	5.01	5.01	5.01	5.01	5.01
	m_4	1.00	1.00	1.00	2.33	1.00
	m_5	1.00	1.00	1.00	1.00	1.00
10^{5}	m_2	5.01	5.01	5.01	5.01	5.01
	m_3	5.01	5.01	5.01	5.01	5.01
	m_4	1.00	1.00	1.00	1.00	1.00
	m_5	1.00	1.00	1.00	1.00	1.00

Table 7
The ratio of computing times ($c = 10^{-15}$, $d = 10^{-15}$).

Method \ Dimension	500	1000	2000	4000	6000	12 000	24 000
M1	1.0	1.0	1.0	1.0	1.0	1.0	1.0
M2	0.7	0.49	0.36	0.30	0.29	0.27	0.26
M3	0.6	0.64	0.56	0.53	0.52	0.51	0.50
M4	0.8	0.71	0.60	0.55	0.53	0.52	0.51
M5	1.0	0.98	0.84	0.79	0.79	0.76	0.77

Next, Ogita–Oishi's algorithm is introduced. Let Q be

$$Q := \texttt{fastNN}(|A|, R_B).$$

Ogita–Oishi's algorithm outputs an enclosure of $A[B]$ as

$$A[B] \subseteq [\mathrm{fl}_{\triangledown}(AM_B - Q),\ \mathrm{fl}_{\triangle}(AM_B + Q)]. \tag{27}$$

Evaluation (27) involves two matrix products: $\mathrm{fl}_{\triangledown}(AM_B)$ and $\mathrm{fl}_{\triangle}(AM_B)$.

Table 8
The ratio of computing times ($c = 10^{-15}$, $d = 10^5$).

Method \ Dimension	500	1000	2000	4000	6000	12 000	24 000
M1	1.0	1.0	1.0	1.0	1.0	1.0	1.0
M2	0.44	0.26	0.13	0.07	0.05	0.02	0.01
M3	0.73	0.63	0.57	0.53	0.52	0.51	0.50
M4	0.70	0.51	0.37	0.31	0.29	0.27	0.26
M5	0.76	0.53	0.39	0.32	0.30	0.30	0.26

Table 9
The ratio of computing times ($c = 10^5$, $d = 10^5$).

Method \ Dimension	500	1000	2000	4000	6000	12 000	24 000
M1	1.0	1.0	1.0	1.0	1.0	1.0	1.0
M2	0.44	0.27	0.13	0.07	0.05	0.02	0.01
M3	0.75	0.66	0.57	0.53	0.52	0.51	0.50
M4	0.77	0.54	0.40	0.33	0.30	0.28	0.26
M5	0.97	0.77	0.63	0.57	0.55	0.53	0.52

Algorithm 11 (*Ogita and Oishi [22]*)**.** For matrices $A \in \mathbb{F}^{m\times n}$ and $[B] \in \mathbb{IF}^{n\times p}$, the following algorithm returns an enclosure $[C] \in \mathbb{IR}^{m\times p}$ of $A[B]$:

```
function [C] = fastInRePI(A, [B])
    [M_B, R_B] = Is2Mr([B]);
    setround(1);
    Q = fastNN(abs(A), R_B);
    T̄ = A * M_B + Q;
    setround(−1);
    T̲ = A * M_B − Q;
    [C] = infsup(T̲, T̄);
end
```

4.2. Proposed algorithms

We propose two algorithms to compute an enclosure of $A[B]$. One algorithm requires only one matrix product. The other requires two matrix products. Define

$$P := \mathrm{fl}_\triangle(\overline{\gamma}_n \mathtt{fastNN}(|A|, |M_B|)).$$

Then, $A[B]$ can be enclosed as follows:

$$A[B] \subseteq \langle \mathrm{fl}(AM_B), \mathrm{fl}_\triangle(P + Q)\rangle. \tag{28}$$

It involves only one matrix product $\mathrm{fl}(M_A M_B)$. We denote an algorithm computing (28) by `PIM1(A, [B], α, β)`.

Algorithm 12. For matrices $A \in \mathbb{F}^{m\times n}$ and $[B] \in \mathbb{IF}^{n\times p}$, the following algorithm returns an enclosure $[C] \in \mathbb{IF}^{m\times p}$ of $A[B]$. The constants α and β are arguments of Algorithm 5.

```
function [C] = PIM1(A, [B], α, β)
    [M_B, R_B] = Is2Mr([B]);
    [M_B, R_B] = SP_check(M_B, R_B, α, β);
    setround(0);
    T = A * M_B;
    setround(1);
    P = gamma(n) * fastNN(abs(A), abs(M_B));
    Q = fastNN(abs(A), R_B);
    [C] = midrad(T, P + Q);
end
```

Next, we introduce an algorithm with two matrix products. Let q_1, q_2 be

$$q_1 = \mathrm{fl}\left(\sum_{i=1}^{m}\sum_{j=1}^{p} P_{ij}\right), \qquad q_2 = \mathrm{fl}\left(\sum_{i=1}^{m}\sum_{j=1}^{p} Q_{ij}\right).$$

If q_1 is larger than q_2, then our algorithm computes the enclosure of $A[B]$ by computing (27). Otherwise, our algorithm outputs the enclosure as

$$A[B] \subseteq \langle \mathrm{fl}(AM_B), \mathrm{fl}_{\triangle}(|A|(\overline{\gamma}_n|M_B| + R_B))\rangle. \tag{29}$$

Evaluation (29) involves two matrix products: $\mathrm{fl}(AM_B)$ and $\mathrm{fl}_{\triangle}(|A|(\gamma_n|M_B| + R_B))$. We denote an algorithm computing (28) by PIM2(A, [B], α, β).

Algorithm 13. For matrices $A \in \mathbb{F}^{m\times n}$ and $[B] \in \mathbb{IF}^{n\times p}$, the following algorithm returns an enclosure $[C] \in \mathbb{IF}^{m\times p}$ of $A[B]$:

```
function [C] = PIM2(A, [B], α, β)
    [M_B, R_B] = Is2Mr([B]);
    [M_B, R_B] = SP_check(M_B, R_B, α, β);
    setround(1);
    P = gamma(n) * fastNN(A, abs(M_B));
    Q = fastNN(abs(A), R_B);
    if q_1 > q_2
        T̄ = A * M_B + Q;
        setround(−1);
        T̲ = A * M_B − Q;
        [C] = infsup(T̲, T̄);
    else
        setround(0);
        T = A * M_B;
        setround(1);
        R = abs(A) * (gamma(n) * abs(M_B) + R_B);
        [C] = midrad(T, R);
    end
end
```

4.3. Numerical examples

We generate $B \in \mathbb{IF}^{n\times n}$ as

$$H = \mathtt{gallery}(\text{'randsvd'}, n, cnd, 3, n, n, 1), \qquad B = \mathtt{midrad}(H, c * \mathtt{abs}(H)),$$

where c is a small and positive constant. The matrix H is one of Higham's randsvd test matrices [19] which is supported in MATLAB.[6] The argument *cnd* is the 2-norm condition number of the matrix. Let A be an approximate inverse matrix of H computed by MATLAB built-in function inv(H). We compare upper bounds for the norms of

- M6: $[C_6] = I -$ midrad_PI(A, [B]), using Algorithm 10,
- M7: $[C_7] = I -$ fastInRePI(A, [B]), using Algorithm 11,
- M8: $[C_8] = I -$ PIM1(A, [B], 0.01, n), using Algorithm 12,
- M9: $[C_9] = I -$ PIM2(A, [B], 0.01, n), using Algorithm 13,

where I denotes the identity matrix. The upper bounds are computed by sup(norm([C_6], int)) on INTLAB. If the upper bound of the norm is less than 1, then it is guaranteed that $[B]$ is nonsingular. We set the dimension n as 5000. All examples in this subsection are tested on Intel Xeon X5550 2.67 GHz, MATLAB 2010a and INTLAB version 6. Each item in Tables 10–12 shows the upper bounds of each norm with various *cnd* and c.

[6] See the detail by typing 'help private/randsvd' on MATLAB. Before generating a matrix, we execute randn('state', 0), and rand('state', 0).

Table 10
Comparison of the upper bounds of $\|I - A[B]\|_\infty$ ($cnd = 10^4$).

c	M6	M7	M8	M9
10^{-8}	0.019	0.084	0.084	0.019
10^{-7}	0.189	0.84	0.84	0.189
$5 \cdot 10^{-7}$	0.946	4.17	4.17	0.946
10^{-6}	1.89	8.35	8.35	1.89

Table 11
Comparison of the upper bounds of $\|I - A[B]\|_\infty$ ($cnd = 10^8$).

c	M6	M7	M8	M9
10^{-12}	0.010	0.042	0.065	0.016
10^{-11}	0.101	0.418	0.441	0.107
$5 \cdot 10^{-11}$	0.506	2.08	2.11	0.512
10^{-10}	1.01	4.17	4.20	1.01

Table 12
Comparison of the upper bounds of $\|I - A[B]\|_\infty$ ($cnd = 10^{12}$).

c	M6	M7	M8	M9
10^{-16}	$9.25 \cdot 10^{-1}$	$9.58 \cdot 10^{-1}$	$1.55 \cdot 10^{2}$	$9.58 \cdot 10^{-1}$
10^{-15}	$9.84 \cdot 10^{-1}$	$1.20 \cdot 10^{0}$	$1.55 \cdot 10^{2}$	$1.20 \cdot 10^{0}$

Table 13
Comparison of the ratio of computing times ($cnd = 10^4$, $c = 10^{-15}$).

Method \ Dimension	500	1000	2000	4000	6000	12 000	24 000
M6	1.0	1.0	1.0	1.0	1.0	1.0	1.0
M7	0.81	0.74	0.71	0.69	0.68	0.69	0.67
M8	0.65	0.50	0.42	0.38	0.36	0.35	0.34
M9	0.94	0.83	0.75	0.71	0.70	0.68	0.68

It is confirmed by Tables 10–12 that $[C_6]$ is the tightest interval of all. The result by `PIM1` is comparable to others in these examples. In addition, it is confirmed that the accuracy of the result by `PIM2` is better than that by `fastInRePI`.[7] However, even if we set $c = 0$ for the matrix with $cnd = 10^{12}$, the approach M8 cannot verify the non-singularity of the matrix.

Table 13 shows the ratio of computing times with $cnd = 10^4$, $c = 10^{-15}$ for each algorithm with various n (if we set another cnd and c, the result is not so different). The displayed number in Table 13 excludes time for computing the approximate inverse. The result shows that our algorithms are efficient for large matrices.

5. Conclusion

In this paper, we have developed new algorithms for computing interval matrix products. Our algorithms first predict which terms become dominant radii of the resultant intervals. Based on that, we choose an appropriate method to avoid the overestimation of the interval. As a result, our algorithms are sometimes comparable to Oishi–Rump's algorithm in terms of accuracy although our algorithms work faster when the inner dimension of the matrices is large.

Acknowledgment

This research was supported by CREST program, Japan Science and Technology Agency (JST).

References

[1] A. Neumaier, Interval Methods for Systems of Equations, Cambridge University Press, 2008.
[2] S.M. Rump, Verification methods: rigorous result using floating-point arithmetic, Acta Numerica 19 (2010) 287–449.
[3] S.M. Rump, Computer-assisted proofs and self-validating methods, in: B. Einarsson (Ed.), Handbook on Accuracy and Reliability in Scientific Computation, SIAM, 2005, pp. 95–240.
[4] S.M. Rump, INTLAB—INTerval LABoratory, in: Tibor Csendes (Ed.), Developments in Reliable Computing, Kluwer Academic Publishers, Dordrecht, 1999, pp. 77–104.
[5] Int4Sci toolbox: a scilab interface for interval analysis. http://www-sop.inria.fr/teams/coprin/logiciels/Int4Sci/.

[7] If $c = 10^{-16}$, then the result by $[C_7]$ is the same to that by $[C_9]$.

[6] http://www.math.uni-wuppertal.de/~xsc/xsc/cxsc.html.
[7] b4m: a free interval arithmetic toolbox for MATLAB. http://www.ti3.tu-harburg.de/zemke/b4m/.
[8] S. Oishi, Fast enclosure of matrix eigenvalues and singular values via rounding mode controlled computation, Linear Algebra and its Applications 324 (2001) 133–146.
[9] T. Ogita, S. Oishi, Fast inclusion of interval matrix multiplication, Reliable Computing 11 (3) (2005) 191–205.
[10] S.M. Rump, Fast interval matrix multiplication, 2011, submitted for publication.
[11] MATLAB Programming version 7, the mathworks.
[12] ANSI/IEEE, IEEE Standard for Binary Floating Point Arithmetic, Std 754-2008 ed., IEEE, New York, 2008.
[13] ANSI/IEEE, IEEE Standard for Binary Floating Point Arithmetic, Std 754-1985 ed., IEEE, New York, 1985.
[14] K. Goto, R. Van De Geijn, High-performance implementation of the level-3 BLAS, ACM Transactions on Mathematical Software 35 (1) (2008) Article No. 4 (20 pages).
[15] J.J. Dongarra, C.R. Whaley, Automatically turned linear algebra software (ATLAS), in: Proceeding of SC'98 Conference, IEEE, 1998.
[16] S.M. Rump, Computational error bounds for multiple or nearly multiple eigenvalues, Linear Algebra and its Applications 324 (2001) 209–226.
[17] S.M. Rump, Fast and parallel interval arithmetic, BIT. Numerical Mathematics 39 (3) (1999) 539–560.
[18] G.H. Golub, C.F. Van Loan, Matrix Computations, third ed., The Johns Hopkins University Press, 1996.
[19] N.J. Higham, Accuracy and Stability of Numerical Algorithms, second ed., SIAM Publications, Philadelphia, 2002.
[20] T. Sunaga, Theory of an interval algebra and its application to numerical analysis, RAAG Memoirs 2 (1958) 29–46.
[21] R.E. Moore, Interval Analysis, Prentice-Hall, Englewood Cliffs, 1966.
[22] S. Oishi, S.M. Rump, Fast verification of solutions of matrix equations, Numerische Mathematik 90 (4) (2002) 755–773.