Efficient implementation of interval matrix multiplication

Hong Diep Nguyen

Efficient implementation of interval matrix multiplication

NGUYEN Hong Diep
INRIA
LIP(UMR 5668 CNRS - ENS de Lyon - INRIA - UCBL)
Université de Lyon
hong.diep.nguyen@ens-lyon.fr

April 1, 2010

Abstract

The straightforward implementation of interval matrix product suffers from poor efficiency, far from the performances of highly optimized floating-point implementations. In this paper, we show how to reduce the interval matrix multiplication to 9 floating-point matrix products - for performance issues - without sacrificing the quality of the result. We show that, compared to the straightforward implementation, the overestimation factor is at most 1.18.

1 Introduction

Interval arithmetic is a means to obtain guaranteed results: enclosures of the results are computed. Nonetheless, it suffers from lower performance than non-guaranteed floating-point computations. Indeed, the factor of performance between interval arithmetic and floating-point arithmetic in theory is four. It is even much worse in practice, especially when a large number of operations is involved.

In this paper we will study the case of matrix multiplication. Our implementation, based on the natural algorithm, provides good results at a cost of very low performance. The idea is to exploit existing libraries which are well optimized for floating-point matrix operations such as BLAS, ATLAS, etc. Rump proposed a fast algorithm which uses only four floating-point matrix products [2]. It returns a result wider than the result obtained by replacing each floating-poing opertaion between two numbers by its interval counterpart: the factor of overestimation in the worst case of this algorithm is 1.5. This paper proposes a new algorithm which costs **nine** floating-point matrix products with the factor of overestimation in the worst case of 1.18.

This paper is organized as follows. Section 2 briefly presents interval arithmetic with some basic operations. These are extended to matrix operations which are studied in section 3. Section 4 reminds the idea of Rump algorithm. Section 5 is devoted to our proposed algorithm.

Notations

In this paper, bold-face lower-case letters represent scalar intervals and bold-face upper-case letters represent interval matrices. Below are some additional notations.

$[\circ]$ an expression computed by interval arithmetic,

$(\circ)_{\downarrow}$, $(\circ)_{\uparrow}$ expressions computed by floating-point arithmetic with downward and upward rounding mode respectively,

$[i, s]$ an interval bounded by i and s,

$\{m, r\}$ an interval whose mid-point is m and radius is r,

$\underline{\mathbf{a}}$, $\overline{\mathbf{a}}$ lower and upper bound of $\mathbf{a}$,

$\mathtt{mag}(\mathbf{a})$ maximal magnitude of $\mathbf{a}$: $\mathtt{mag}(\mathbf{a}) \stackrel{def}{=} \mathtt{max}\{|a|, a \in \mathbf{a}\} = \mathtt{max}(|\underline{\mathbf{a}}|, |\overline{\mathbf{a}}|)$

2 Interval Arithmetic

Interval are used to represent connected closed sets of real values. Interval arithmetic defines operations between intervals. The result of an operation between intervals is also an interval containing all the possible results between all possible pairs of real values taken from input intervals: $\mathbf{r} = \mathbf{a}[\circ]\mathbf{b} = \{a \circ b, a \in \mathbf{a}, b \in \mathbf{b}\}$.

Due to the monotonicity property, the sum of two intervals $\mathbf{r} = \mathbf{a} + \mathbf{b}$ can be computed by the sum of their two respective lower and upper bounds:

$$\underline{\mathbf{r}} = \underline{\mathbf{a}} + \underline{\mathbf{b}} \qquad \overline{\mathbf{r}} = \overline{\mathbf{a}} + \overline{\mathbf{b}} \tag{1}$$

Interval multiplication $\mathbf{r} = \mathbf{a} * \mathbf{b}$ is formed by taking the minimum and maximum value of the four products between two pairs of bounds of the two input intervals.

$$\left\{ \begin{array}{rcl} \underline{\mathbf{r}} & = & \mathtt{min}(\underline{\mathbf{a}} * \underline{\mathbf{b}}, \underline{\mathbf{a}} * \overline{\mathbf{b}}, \overline{\mathbf{a}} * \underline{\mathbf{b}}, \overline{\mathbf{a}} * \overline{\mathbf{b}}) \\ \overline{\mathbf{r}} & = & \mathtt{max}(\underline{\mathbf{a}} * \underline{\mathbf{b}}, \underline{\mathbf{a}} * \overline{\mathbf{b}}, \overline{\mathbf{a}} * \underline{\mathbf{b}}, \overline{\mathbf{a}} * \overline{\mathbf{b}}) \end{array} \right. \tag{2}$$

Hence, in theory the factor, in terms of performance, between interval arithmetic and floating-point arithmetic is 4.

Implementation

Intervals are implemented on computers using floating-point numbers. To ensure the inclusion property of the results, rounding errors must be taken into account.

For interval addition $\mathbf{r} = \mathbf{a} + \mathbf{b}$, the lower bound must be computed with downward rounding mode, and the upper bound with upward rounding mode.

$$\underline{\mathbf{r}} = (\underline{\mathbf{a}} + \underline{\mathbf{b}})_{\downarrow} \qquad \overline{\mathbf{r}} = (\overline{\mathbf{a}} + \overline{\mathbf{b}})_{\uparrow} \tag{3}$$

Interval product $\mathbf{r} = \mathbf{a} * \mathbf{b}$ is computed following (2) by four floating-point products. Nevertheless, to tackle rounding errors, each floating-point product must be computed twice, once with upward and once with downward rounding mode. Thus in total, it requires eight floating-point products.

$$\begin{cases} \underline{\mathbf{r}} &= \mathtt{min}((\underline{\mathbf{a}} * \underline{\mathbf{b}})_{\downarrow}, (\underline{\mathbf{a}} * \overline{\mathbf{b}})_{\downarrow}, (\overline{\mathbf{a}} * \underline{\mathbf{b}})_{\downarrow}, (\overline{\mathbf{a}} * \overline{\mathbf{b}})_{\downarrow}) \\ \overline{\mathbf{r}} &= \mathtt{max}((\underline{\mathbf{a}} * \underline{\mathbf{b}})_{\uparrow}, (\underline{\mathbf{a}} * \overline{\mathbf{b}})_{\uparrow}, (\overline{\mathbf{a}} * \underline{\mathbf{b}})_{\uparrow}, (\overline{\mathbf{a}} * \overline{\mathbf{b}})_{\uparrow}) \end{cases} \tag{4}$$

We can reduce the number of floating-point products by inspecting the sign of each component. But testing the sign is costly also.

Particular cases

If a and b are centered in zero

ie $\underline{\mathbf{a}} = -\overline{\mathbf{a}}$ and $\underline{\mathbf{b}} = -\overline{\mathbf{b}}$ then $\underline{\mathbf{a}} + \underline{\mathbf{b}} = -(\overline{\mathbf{a}} + \overline{\mathbf{b}})$. Hence, the sum $\mathbf{r} = \mathbf{a} + \mathbf{b}$ can be computed by:

$$\overline{\mathbf{r}} = \left(\overline{\mathbf{a}} + \overline{\mathbf{b}}\right)_{\uparrow} \qquad \underline{\mathbf{r}} = -\overline{\mathbf{r}} \tag{5}$$

If b is centered in zero

then $\mathtt{max}(x * \underline{\mathbf{b}}, x * \overline{\mathbf{b}}) = |x| * \overline{\mathbf{b}}$ and $\mathtt{min}(x * \underline{\mathbf{b}}, x * \overline{\mathbf{b}}) = -|x| * \overline{\mathbf{b}}$ for all $x \in \mathbb{R}$. Hence according to (2), $\mathbf{r} = \mathbf{a} * \mathbf{b}$ can be computed by:

$$\begin{aligned} \overline{\mathbf{r}} &= \mathtt{max}(|\underline{\mathbf{a}}|, |\overline{\mathbf{a}}|) * \overline{\mathbf{b}} \\ &= \mathtt{mag}(\mathbf{a}) * \overline{\mathbf{b}} \\ \underline{\mathbf{r}} &= -\mathtt{max}(|\underline{\mathbf{a}}|, |\overline{\mathbf{a}}|) * \overline{\mathbf{b}} \\ &= -\overline{\mathbf{r}} \end{aligned}$$

Using floating-point arithmetic, $\mathbf{r}$ can be computed by only one floating-point product in upward rounding mode:

$$\overline{\mathbf{r}} = \left(\mathtt{mag}(\mathbf{a}) * \overline{\mathbf{b}}\right)_{\uparrow} \qquad \underline{\mathbf{r}} = -\overline{\mathbf{r}} \tag{6}$$

3 Interval matrix operations

Let's now study the case of matrix operations. Suppose that each interval matrix here is of dimension $n \times n$ and is represented by two floating-point matrices, one for its lower bound and the other for its upper bound.

The addition of two interval matrices $\mathbf{C} = \mathbf{A} + \mathbf{B}$ can be computed by performing scalar additions between corresponding elements of the two matrices. Thus

$$\underline{\mathbf{C}} = (\underline{\mathbf{A}} + \underline{\mathbf{B}})_{\downarrow} \qquad \overline{\mathbf{C}} = \left(\overline{\mathbf{A}} + \overline{\mathbf{B}}\right)_{\uparrow}$$

For the case of interval matrix multiplication, each element of the result matrix is computed by:

$$\mathbf{C}_{i,j} = \sum_k \mathbf{A}_{i,k} * \mathbf{B}_{k,j} \tag{7}$$

Using this natural formula, the computation of each product element requires n interval multiplications and n interval additions, or, following (3) and (4), it requires $8n$ floating-point products and $2n$ floating-point additions. Hence, the overall cost in theory is $8n_*^3 + 2n_+^3$ floating-point operations. Nonetheless, this cost does not take into account neither $\mathtt{min}, \mathtt{max}$ functions nor the cost of rounding mode changes.

Particular cases

A is centered in zero

ie $\mathbf{A}_{i,k}$ is centered in zero for all i, k. From (6) we get

$$\begin{array}{lrcl} & \mathbf{A}_{i,k} * \mathbf{B}_{k,j} & = & \left[-\overline{\mathbf{A}}_{i,k} * \mathtt{mag}\left(\mathbf{B}\right)_{k,j}, \overline{\mathbf{A}}_{i,k} * \mathtt{mag}\left(\mathbf{B}\right)_{k,j}\right] \\ \Rightarrow & \mathbf{C}_{i,j} & = & \left[-\sum_k \overline{\mathbf{A}}_{i,k} * \mathtt{mag}\left(\mathbf{B}\right)_{k,j}, \sum_k \overline{\mathbf{A}}_{i,k} * \mathtt{mag}\left(\mathbf{B}\right)_{k,j}\right] \\ \Rightarrow & \overline{\mathbf{C}} & = & \left(\overline{\mathbf{A}} * \mathtt{mag}\left(\mathbf{B}\right)\right)_{\uparrow} \\ & \underline{\mathbf{C}} & = & -\overline{\mathbf{C}} \end{array}$$

Hence $\mathbf{A} * \mathbf{B}$ can be computed by one floating-point matrix product with upward rounding mode.

A is non-negative

It means that $0 \leq \underline{\mathbf{A}}_{i,k}$ and $0 \leq \overline{\mathbf{A}}_{i,k}$ for all i, k. Hence

$$\left\{\begin{array}{lcl} \mathtt{max}(\underline{\mathbf{A}}_{i,k} * \underline{\mathbf{B}}_{k,j}, \underline{\mathbf{A}}_{i,k} * \overline{\mathbf{B}}_{k,j}) & = & \underline{\mathbf{A}}_{i,k} * \overline{\mathbf{B}}_{k,j} \\ \mathtt{min}(\underline{\mathbf{A}}_{i,k} * \underline{\mathbf{B}}_{k,j}, \underline{\mathbf{A}}_{i,k} * \overline{\mathbf{B}}_{k,j}) & = & \underline{\mathbf{A}}_{i,k} * \underline{\mathbf{B}}_{k,j} \\ \mathtt{max}(\overline{\mathbf{A}}_{i,k} * \underline{\mathbf{B}}_{k,j}, \overline{\mathbf{A}}_{i,k} * \overline{\mathbf{B}}_{k,j}) & = & \overline{\mathbf{A}}_{i,k} * \overline{\mathbf{B}}_{k,j} \\ \mathtt{min}(\overline{\mathbf{A}}_{i,k} * \underline{\mathbf{B}}_{k,j}, \overline{\mathbf{A}}_{i,k} * \overline{\mathbf{B}}_{k,j}) & = & \overline{\mathbf{A}}_{i,k} * \underline{\mathbf{B}}_{k,j} \end{array}\right.$$

Denote by $x^{pos} = \mathtt{max}(x, 0)$ and $x^{neg} = \mathtt{min}(x, 0)$ then

$$\left\{\begin{array}{lcl} \mathtt{max}(\underline{\mathbf{A}}_{i,k} * \overline{\mathbf{B}}_{k,j}, \overline{\mathbf{A}}_{i,k} * \overline{\mathbf{B}}_{k,j}) & = & \overline{\mathbf{A}}_{i,k} * \overline{\mathbf{B}}_{k,j}^{pos} + \underline{\mathbf{A}}_{i,k} * \overline{\mathbf{B}}_{k,j}^{neg} \\ \mathtt{min}(\underline{\mathbf{A}}_{i,k} * \underline{\mathbf{B}}_{k,j}, \overline{\mathbf{A}}_{i,k} * \underline{\mathbf{B}}_{k,j}) & = & \underline{\mathbf{A}}_{i,k} * \underline{\mathbf{B}}_{k,j}^{pos} + \overline{\mathbf{A}}_{i,k} * \underline{\mathbf{B}}_{k,j}^{neg} \end{array}\right.$$

$$\Rightarrow \left\{\begin{array}{lcl} \overline{\mathbf{A}_{i,k} * \mathbf{B}_{k,j}} & = & \overline{\mathbf{A}}_{i,k} * \overline{\mathbf{B}}_{k,j}^{pos} + \underline{\mathbf{A}}_{i,k} * \overline{\mathbf{B}}_{k,j}^{neg} \\ \underline{\mathbf{A}_{i,k} * \mathbf{B}_{k,j}} & = & \underline{\mathbf{A}}_{i,k} * \underline{\mathbf{B}}_{k,j}^{pos} + \overline{\mathbf{A}}_{i,k} * \underline{\mathbf{B}}_{k,j}^{neg} \end{array}\right.$$

$$\Rightarrow \left\{\begin{array}{lcl} \overline{\mathbf{C}} & = & \left(\overline{\mathbf{A}} * \overline{\mathbf{B}}^{pos} + \underline{\mathbf{A}} * \overline{\mathbf{B}}^{neg}\right)_{\uparrow} \\ \underline{\mathbf{C}} & = & \left(\underline{\mathbf{A}} * \underline{\mathbf{B}}^{pos} + \underline{\mathbf{A}} * \underline{\mathbf{B}}^{neg}\right)_{\downarrow} \end{array}\right.$$

In this case, an interval matrix product can be computed by 4 floating-point matrix products.

Similarly, if $\mathbf{A}$ is non positive, 4 floating-point matrix products suffice to compute an interval matrix product.

A does not contain zero in its interior

We can then split $\mathbf{A}$ into positive and negative part by

$$\begin{array}{rcl} \mathbf{a}^{pos} & = & \begin{cases} \{a \in \mathbf{a} \mid a \geq 0\} & \texttt{if} \quad \overline{\mathbf{a}} \geq 0 \\ 0 & \texttt{if} \quad \overline{\mathbf{a}} < 0 \end{cases} \\ \mathbf{a}^{neg} & = & \begin{cases} \{a \in \mathbf{a} \mid a \leq 0\} & \texttt{if} \quad \underline{\mathbf{a}} \leq 0 \\ 0 & \texttt{if} \quad \underline{\mathbf{a}} > 0 \end{cases} \end{array}$$

$\mathbf{A}$ does not contain zero in its interior, thus either $\mathbf{A}^{pos}_{i,j} = \mathbf{A}_{i,j}, \mathbf{A}^{neg}_{i,j} = 0$ or $\mathbf{A}^{pos}_{i,j} = 0, \mathbf{A}^{neg}_{i,j} = \mathbf{A}_{i,j}$. Hence

$$\begin{array}{rcl} \mathbf{A}_{i,j} * \mathbf{B}_{j,k} & = & \mathbf{A}_{i,j} * \mathbf{B}^{pos}_{j,k} + \mathbf{A}_{i,j} * \mathbf{B}^{neg}_{j,k} \\ \Rightarrow \mathbf{A} * \mathbf{B} & = & \mathbf{A}^{pos} * \mathbf{B} + \mathbf{A}^{neg} * \mathbf{B} \end{array}$$

In total, it costs eight floating-point matrix products.

For these three particular cases, if all operations are performed in infinite precision or if there is no rounding error, then the computed result is exact.

In general, when there is no assumption about the input intervals then it would not be efficient to use the natural algorithm. One solution is to exploit floating-point matrix multiplication because there are available libraries which are well optimised for floating-point matrix operations.

4 Rump's algorithm

Rump proposes an algorithm which makes use of floating-point operations for speed [2, 3]. This algorithm is based on the midpoint-radius representation of intervals.

Let $\mathbf{A} = \{m_A, r_A\}$ and $\mathbf{B} = \{m_B, r_B\}$ be two interval matrices, with m_A, m_B their midpoints and r_A, r_B their radius respectively. The product $\mathbf{A} * \mathbf{B}$ is enclosed by an interval matrix $\mathbf{C}$ whose midpoint and radius are computed by:

$$\begin{array}{rcl} m_C & = & m_A * m_B \\ r_C & = & (|m_A| + r_A) * r_B + r_A * |m_B| \end{array}$$

In fact, because of rounding errors, $m_A * m_B$ cannot be computed exactly. Thus it must be computed with both upward and downward rounding mode to obtain en enclosure of the midpoint. The radius r_C must also be computed with upward rounding to ensure the inclusion property.

$$\begin{array}{rcl} \mathbf{mC} & = & \left[(m_A * m_B)_{\downarrow}, (m_B * m_B)_{\uparrow} \right] \\ r_C & = & ((|m_A| + r_A) * r_B + r_A * |m_B|)_{\uparrow} \end{array}$$

Finally, the result can be easily converted back to endpoints representation by two floating-point additions.

$$\overline{\mathbf{C}} = \left(\overline{\mathbf{mC}} + r_C\right)_{\uparrow} \qquad \underline{\mathbf{C}} = \left(\underline{\mathbf{mC}} - r_C\right)_{\downarrow}$$

In total, this algorithm uses **four** floating-point products.

Over-estimation

Without taking into account of rounding errors, Rump's algorithm always provides over-estimating results. The factor of over-estimation in the worst case is 1.5.

For example, by definition $[0,2] * [0,4] \stackrel{def}{=} [0,8]$.

Meanwhile, Rump's algorithm gives

$$\begin{aligned} [0,2] * [0,4] &= \{1,1\} * \{2,2\} \\ &= \{1*2, (1+1)*2 + 1*2\} \\ &= \{2,6\} \\ &= [-4,8] \end{aligned}$$

The over-estimation factor in this case is $12/8 = 1.5$.

5 Proposition

Rump algorithm can be considered as decomposing $\mathbf{A}$ and $\mathbf{B}$ into a sum of two components representing its midpoint and radius respectively. $\mathbf{A} * \mathbf{B}$ is then replaced by its development, which is a sum of four sub-products. Due to the sub-distributive property of interval product, the result yielded by this sum is an enclosure of the original product.

Our idea is to find another decomposition such that sub-products can be efficiently computed, and the over-estimation is small.

As we can see in Section 3, an interval product can be efficiently computed when one of the two multipliers is centered in zero or does not contain zero.

Proposition 1. *Let $\boldsymbol{A}$ be an interval matrix. If $\boldsymbol{A}$ is decomposed into two interval matrices $\boldsymbol{A}^0$ and $\boldsymbol{A}^*$ which satisfy:*

- $\boldsymbol{A}^0_{i,j} = 0, \boldsymbol{A}^*_{i,j} = \boldsymbol{A}_{i,j}$ *if* $\underline{\boldsymbol{A}}_{i,j} * \overline{\boldsymbol{A}}_{i,j} \geq 0$,
- $\boldsymbol{A}^0_{i,j} = \left[\underline{\boldsymbol{A}}_{i,j}, -\underline{\boldsymbol{A}}_{i,j}\right], \boldsymbol{A}^*_{i,j} = \left[0, \underline{\boldsymbol{A}}_{i,j} + \boldsymbol{A}_{i,j}\right]$ *if* $\underline{\boldsymbol{A}}_{i,j} < 0 < \left|\underline{\boldsymbol{A}}_{i,j}\right| \leq \overline{\boldsymbol{A}}_{i,j}$,
- $\boldsymbol{A}^0_{i,j} = \left[-\overline{\boldsymbol{A}}_{i,j}, \overline{\boldsymbol{A}}_{i,j}\right], \boldsymbol{A}^*_{i,j} = \left[\underline{\boldsymbol{A}}_{i,j} + \boldsymbol{A}_{i,j}, 0\right]$ *if* $\underline{\boldsymbol{A}}_{i,j} < 0 < \overline{\boldsymbol{A}}_{i,j} < \left|\underline{\boldsymbol{A}}_{i,j}\right|$

then

- $\boldsymbol{A}^0$ *is centered in zero,*
- $\boldsymbol{A}^*$ *does not contain zero in its interior, and*

- $\boldsymbol{A}^0 + \boldsymbol{A}^* = \boldsymbol{A}$.

Proof. Easily deduced from the formula. □

Proposition 2. *Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be two interval matrices and $(\boldsymbol{A}^0, \boldsymbol{A}^*)$ a decomposition of $\boldsymbol{A}$ by Proposition 1. Let $\boldsymbol{C}$ be an interval matrix computed by*

$$\boldsymbol{C} = \boldsymbol{A}^0 * \boldsymbol{B} + \boldsymbol{A}^* * \boldsymbol{B} \tag{8}$$

Then $\boldsymbol{A} * \boldsymbol{B}$ is contained in $\boldsymbol{C}$. Denote $\boldsymbol{C} \stackrel{def}{=} \boldsymbol{A} \circledast \boldsymbol{B}$.

Proof. Following Proposition 1:

$$\mathbf{A} = \mathbf{A}^0 + \mathbf{A}^*$$

Interval multiplication is sub-distributive, hence

$$\begin{aligned} \mathbf{A} * \mathbf{B} &\subseteq \mathbf{A}^0 * \mathbf{B} + \mathbf{A}^* * \mathbf{B} \\ &\subseteq \mathbf{C} \end{aligned}$$

□

$\mathbf{A}^0$ is centered in zero and $\mathbf{A}^*$ does not contain zero, so according to Section 3, $\mathbf{A}^0 * \mathbf{B}$ and $\mathbf{A}^* * \mathbf{B}$ can be computed using 1 and 8 floating-point matrix products respectively. Hence, the overall cost is 9 floating-point matrix products. The following section will study the over-estimation factor of this operation in the worst case.

Over-estimation factor

Let us first consider the product of two scalar intervals $\mathbf{a}$ and $\mathbf{b}$ with $\mathbf{a}$ being decomposed, following Proposition 1, into two parts: $\mathbf{a} = \mathbf{a}^0 + \mathbf{a}^*$. Suppose that all calculations here are performed in infinite precision. It means that rounding errors will not be taken into account.

If a is centered in zero

then $\mathbf{a}^* = 0 \rightarrow \mathbf{a} * \mathbf{b} = \mathbf{a}^0 * \mathbf{b}$. Thus the result is exact.

If a does not contain zero in its interior

then $\mathbf{a}^0 = 0 \rightarrow \mathbf{a} * \mathbf{b} = \mathbf{a}^* * \mathbf{b}$. Thus the result is exact too.

If a contains zero

It means that $\underline{\mathbf{a}} < 0 < \overline{\mathbf{a}}$. The case $\overline{\mathbf{a}} < |\underline{\mathbf{a}}|$ is contrary to the case $\overline{\mathbf{a}} > |\underline{\mathbf{a}}|$, hence without loss of generality we suppose that $\overline{\mathbf{a}} > |\underline{\mathbf{a}}|$. In this case $\mathbf{a}^0 = [\underline{\mathbf{a}}, -\underline{\mathbf{a}}]$ and $\mathbf{a}^* = [0, \overline{\mathbf{a}} + \underline{\mathbf{a}}]$. Following Proposition 2

$$\begin{aligned} \mathbf{a} \circledast \mathbf{b} &= \mathbf{a}^0 * \mathbf{b} + \mathbf{a}^* * \mathbf{b} \\ &= [\underline{\mathbf{a}}, -\underline{\mathbf{a}}] * \mathbf{b} + [0, \overline{\mathbf{a}} + \underline{\mathbf{a}}] * \mathbf{b} \\ &= [\underline{\mathbf{a}} * \mathtt{mag}(\mathbf{b}), -\underline{\mathbf{a}} * \mathtt{mag}(\mathbf{b})] + [0, \overline{\mathbf{a}} + \underline{\mathbf{a}}] * \mathbf{b} \end{aligned}$$

- If $\mathbf{b} \geq 0$, then $\mathtt{mag}(\mathbf{b}) = \overline{\mathbf{b}}$

$$\begin{aligned} \mathbf{a} \circledast \mathbf{b} &= [\underline{\mathbf{a}} * \mathtt{mag}(\mathbf{b}), -\underline{\mathbf{a}} * \mathtt{mag}(\mathbf{b})] + [0, \overline{\mathbf{a}} + \underline{\mathbf{a}}] * \mathbf{b} \\ &= \left[\underline{\mathbf{a}} * \overline{\mathbf{b}}, -\underline{\mathbf{a}} * \overline{\mathbf{b}}\right] + \left[0, (\overline{\mathbf{a}} + \underline{\mathbf{a}}) * \overline{\mathbf{b}}\right] \\ &= \left[\underline{\mathbf{a}} * \overline{\mathbf{b}}, \overline{\mathbf{a}} * \overline{\mathbf{b}}\right] \end{aligned}$$

  Meanwhile, $\mathbf{b} \geq 0 \rightarrow \mathbf{a} * \mathbf{b} = \left[\underline{\mathbf{a}} * \overline{\mathbf{b}}, \overline{\mathbf{a}} * \overline{\mathbf{b}}\right]$. Hence $\mathbf{a} \circledast \mathbf{b} = \mathbf{a} * \mathbf{b}$. It means that there is no overestimation.

- Idem for the case $\mathbf{b} \leq 0$.

- If $\overline{\mathbf{b}} > 0 > \underline{\mathbf{b}}$ and $\overline{\mathbf{b}} \geq |\underline{\mathbf{b}}|$ then $\mathtt{mag}(\mathbf{b}) = \overline{\mathbf{b}}$. Hence

$$\begin{aligned} \mathbf{a} \circledast \mathbf{b} &= [\underline{\mathbf{a}} * \mathtt{mag}(\mathbf{b}), -\underline{\mathbf{a}} * \mathtt{mag}(\mathbf{b})] + [0, \overline{\mathbf{a}} + \underline{\mathbf{a}}] * \mathbf{b} \\ &= \left[\underline{\mathbf{a}} * \overline{\mathbf{b}}, -\underline{\mathbf{a}} * \overline{\mathbf{b}}\right] + \left[(\overline{\mathbf{a}} + \underline{\mathbf{a}}) * \underline{\mathbf{b}}, (\overline{\mathbf{a}} + \underline{\mathbf{a}}) * \overline{\mathbf{b}}\right] \\ &= \left[(\overline{\mathbf{a}} + \underline{\mathbf{a}}) * \underline{\mathbf{b}} + \underline{\mathbf{a}} * \overline{\mathbf{b}}, \overline{\mathbf{a}} * \overline{\mathbf{b}}\right] \end{aligned}$$

$$\begin{aligned} \Rightarrow \mathtt{diam}(\mathbf{a} \circledast \mathbf{b}) &= \overline{\mathbf{a}} * \overline{\mathbf{b}} - ((\overline{\mathbf{a}} + \underline{\mathbf{a}}) * \underline{\mathbf{b}} + \underline{\mathbf{a}} * \overline{\mathbf{b}}) \\ &= \overline{\mathbf{a}} * \overline{\mathbf{b}} * (1 - \underline{\mathbf{a}}/\overline{\mathbf{a}} - \underline{\mathbf{b}}/\overline{\mathbf{b}} - \underline{\mathbf{a}}/\overline{\mathbf{a}} * \underline{\mathbf{b}}/\overline{\mathbf{b}}) \end{aligned}$$

  As $\overline{\mathbf{a}} > 0 > \underline{\mathbf{a}} > -\overline{\mathbf{a}}$ and $\overline{\mathbf{b}} > 0 > \underline{\mathbf{b}} > -\overline{\mathbf{b}}$

$$\begin{aligned} \mathbf{a} * \mathbf{b} &= \left[\mathtt{min}(\overline{\mathbf{a}} * \underline{\mathbf{b}}, \underline{\mathbf{a}} * \overline{\mathbf{b}}), \overline{\mathbf{a}} * \overline{\mathbf{b}}\right] \\ \Rightarrow \mathtt{diam}(\mathbf{a} * \mathbf{b}) &= \overline{\mathbf{a}} * \overline{\mathbf{b}} - \mathtt{min}(\overline{\mathbf{a}} * \underline{\mathbf{b}}, \underline{\mathbf{a}} * \overline{\mathbf{b}}) \\ &= \overline{\mathbf{a}} * \overline{\mathbf{b}} * (1 - \mathtt{min}(\underline{\mathbf{a}}/\overline{\mathbf{a}}, \underline{\mathbf{b}}/\overline{\mathbf{b}})) \end{aligned}$$

Denote by $\left\{ \begin{array}{lcl} M &=& \mathtt{max}(|\underline{\mathbf{a}}|/\overline{\mathbf{a}}, |\underline{\mathbf{b}}|/\overline{\mathbf{b}}) \\ m &=& \mathtt{min}(|\underline{\mathbf{a}}|/\overline{\mathbf{a}}, |\underline{\mathbf{b}}|/\overline{\mathbf{b}}) \end{array} \right.$ then

$$\left\{ \begin{array}{lcl} 0 < m \leq M \leq 1 & & \\ \mathtt{min}(\underline{\mathbf{a}}/\overline{\mathbf{a}}, \underline{\mathbf{b}}/\overline{\mathbf{b}}) &=& -M \\ \underline{\mathbf{a}}/\overline{\mathbf{a}} + \underline{\mathbf{b}}/\overline{\mathbf{b}} &=& -m - M \\ \underline{\mathbf{a}}/\overline{\mathbf{a}} * \underline{\mathbf{b}}/\overline{\mathbf{b}} &=& m * M \end{array} \right.$$

The factor of overestimation is computed by:

$$\begin{aligned}\frac{\mathtt{diam}(\mathbf{a} \circledast \mathbf{b})}{\mathtt{diam}(\mathbf{a} * \mathbf{b})} &= \frac{1 - \underline{\mathbf{a}}/\overline{\mathbf{a}} - \underline{\mathbf{b}}/\overline{\mathbf{b}} - \underline{\mathbf{a}}/\overline{\mathbf{a}} * \underline{\mathbf{b}}/\overline{\mathbf{b}}}{1 - \mathtt{min}(\underline{\mathbf{a}}/\overline{\mathbf{a}}, \underline{\mathbf{b}}/\overline{\mathbf{b}})} \\ &= \frac{1 + M + m - Mm}{1 + M} \\ &= \frac{1 + M + m(1 - M)}{1 + M} \\ &\leq \frac{1 + M + M(1 - M)}{1 + M}\end{aligned}$$

Inspecting the last function of unknown M between 0 and 1, we have that its maximum is $4 - 2\sqrt{2} \approx 1.18$ and this maximum is reached for $m = M = \sqrt{2} - 1$, or $\underline{\mathbf{a}}/\overline{\mathbf{a}} = \underline{\mathbf{b}}/\overline{\mathbf{b}} = 1 - \sqrt{2}$.

- Idem for the case $\overline{\mathbf{b}} > 0 > \underline{\mathbf{b}}$ and $\overline{\mathbf{b}} < |\underline{\mathbf{b}}|$

Let's now extend to the case of matrix multiplication. Each element of the result matrix is computed by

$$\begin{aligned}\mathbf{C}_{i,j} &= \sum_k \mathbf{A}^0_{i,k} * \mathbf{B}_{k,j} + \sum_k \mathbf{A}^*_{i,k} * \mathbf{B}_{k,j} \\ &= \sum_k (\mathbf{A}^0_{i,k} * \mathbf{B}_{k,j} + \mathbf{A}^*_{i,k} * \mathbf{B}_{k,j})\end{aligned}$$

Moreover

$$\begin{aligned}\mathtt{diam}(\mathbf{a} + \mathbf{b}) &= (\overline{\mathbf{a}} + \overline{\mathbf{b}}) - (\underline{\mathbf{a}} + \underline{\mathbf{b}}) \\ &= (\overline{\mathbf{a}} - \underline{\mathbf{a}}) + (\overline{\mathbf{b}} - \underline{\mathbf{b}}) \\ &= \mathtt{diam}(\mathbf{a}) + \mathtt{diam}(\mathbf{b}) \\ \Rightarrow \mathtt{diam}(\mathbf{C}_{i,j}) &= \mathtt{diam}\left(\sum_k (\mathbf{A}^0_{i,k} * \mathbf{B}_{k,j} + \mathbf{A}^*_{i,k} * \mathbf{B}_{k,j})\right) \\ &= \sum_k \mathtt{diam}(\mathbf{A}^0_{i,k} * \mathbf{B}_{k,j} + \mathbf{A}^*_{i,k} * \mathbf{B}_{k,j})\end{aligned}$$

With the assumption of no rounding error, the over-estimation factor in the worst case for this scalar product is $4 - 2\sqrt{2}$. Hence

$$\begin{aligned}\mathtt{diam}(\mathbf{C}_{i,j}) &\leq (4 - 2\sqrt{2}) * \sum_k \mathtt{diam}(\mathbf{A}_{i,k} * \mathbf{B}_{k,j}) \\ &\leq (4 - 2\sqrt{2}) * \mathtt{diam}\left(\sum_k \mathbf{A}_{i,k} * \mathbf{B}_{k,j}\right) \\ \Rightarrow \mathtt{diam}(\mathbf{C}) &\leq (4 - 2\sqrt{2}) * \mathtt{diam}(\mathbf{A} * \mathbf{B})\end{aligned}$$

Hence, the over-estimation factor in the worst case of interval matrix product is also $4 - 2\sqrt{2} \approx 1.18$.

6 Conclusion

The algorithm presented in this paper implements the product of interval matrices using floating-point operations. It constitutes a trade-off between the performances of optimized floating-point libraries, and a slight overestimation of the result. We have proven that this overestimation factor is at most 1.18, which means that the width of computed result is always less than 1.18 times the width of exact result. In particular, if one of the two multipliers does not contain zero in its interior then the computed result is exact using exact arithmetic.

The performance of the algorithm given above relies entirely on the performance of the employed floating-point library. Such a library must support directed rounding modes to be of use. This requirement is not really an issue, since libraries such as LAPACK or ATLAS, which are renowned for their performances, support directed roundings.

However, the use of directed rounding modees restricts the algorithm that can be used as a basic algorithm to the natural one. Indeed, fast algorithms for matrix products [1], which have a complexity below $\mathcal{O}(n^3)$, do not preserve the monotonicity of operations (they use substraction as well as addition and multiplication) and they cannot be used. Furthermore, such algorithms rely on algebraic properties such as $(x + y) - y = x$, which do not hold in interval arithmetic.

References

[1] M. Ceberio and V. Kreinovich. Fast multiplication of interval matrices (interval version of strassen's algorithm). *Reliable Computing*, 10(3):241–243, 2004.

[2] S. M. Rump. Fast and parallel interval arithmetic. *BIT*, 39(3):534–554, 1999.

[3] S. M. Rump. Computer-assisted proofs and self-validating methods. In B. Einarsson, editor, *Handbook on Accuracy and Reliability in Scientific Computation*, chapter 10, pages 195–240. SIAM, 2005.