Tight interval inclusions with compensated algorithms

Stef Graillat, Fabienne Jézéquel

Tight interval inclusions with compensated algorithms

Stef Graillat and Fabienne Jézéquel

Abstract—Compensated algorithms consist in computing the rounding errors of individual operations and then adding them later on to the computed result. This makes it possible to increase the accuracy of the computed result efficiently. Computing the rounding error of an individual operation is possible through the use of a so-called *error-free transformation*. In this article, we show that it is possible to use compensated algorithms for having tight interval inclusion. We study compensated algorithms for summation, dot product and polynomial evaluation. We prove that the use of directed rounding makes it possible to get narrow inclusions with compensated algorithms. This is due to the fact that error-free transformations are no more exact but still sufficiently accurate to improve the numerical quality of results.

Index Terms—interval arithmetic, directed rounding, compensated algorithms, error-free transformations, floating-point arithmetic, numerical validation, rounding errors, summation algorithms, dot product, Horner scheme.

1 Introduction

In June 2018, researchers at the US Department of Energy's Oak Ridge National Laboratory broke the exascale barrier, achieving on the *Summit* supercomputer[1] a peak throughput of 1.88 exaflops (*i.e.* $1.88\,10^{18}$ floating-point operations per second). Unfortunately, with exascale computing, or more generally with high performance computing, a large number of rounding errors may be generated. Indeed, nearly all floating-point operations imply a small rounding which can accumulate along the computation and finally an incorrect result may be produced. As a consequence, it is crucial to propose methods and tools for numerical validation and accurate computation.

To improve the numerical quality of results, one can increase the working precision. In addition to the widely used *binary32* and *binary64* formats, the IEEE 754-2008 standard [1] defines the *binary128* format, also called quadruple precision, that is implemented in compilers such as the GNU compiler *gcc* and the Intel compiler *icc*. Moreover arbitrary precision libraries exist: one can cite ARPREC [2] and MPFR [3]. The computing precision can also be extended thanks to expansions, unevaluated sums of standard floating-point numbers. The QD package [4] provides the *double-double* and the *quad-double* data types, that consist of respectively two and four *binary64* floating-point numbers. One can also use arbitrary length expansions [5], [6], [7]. If a simple enough computation is performed, its accuracy can be improved thanks to compensated algorithms [8], [9], [10]. These algorithms are based on error-free transformations (EFTs) that make it possible to compute the rounding errors of some elementary operations like addition and multiplication exactly.

Interval arithmetic [11], [12] is a well known approach to control the validity of numerical results. It briefly consists in performing floating-point operations on intervals instead of scalars. These operations give a 100% certain result, represented as an interval containing the exact result. The main advantage of this approach lies in the guaranteed error bounds it provides.

In this paper we show how to compute tight interval inclusions with compensated algorithms. To obtain garanteed interval bounds, directed rounding should be used. However EFTs are intended to be used with rounding to nearest. Therefore we study the behaviour of EFTs with directed rounding. Results presented in [13], [14], [15] are completed in this paper. In particular, concerning the EFT for the multiplication without FMA (*Fused-Multiply-and-Add* operator) we bound the difference between the rounding error and the correction computed with directed rounding. In this paper we also show that EFTs executed with directed rounding provide garanteed bounds on the results of additions and multiplications. We complete results established in [13], [15] on the behaviour with directed rounding of compensated algorithms based on these EFTs. Then we show that, thanks to compensated algorithms executed with directed rounding, tight interval inclusions can be computed for summation, dot product, and polynomial evaluation with Horner scheme.

The outline of this article is as follows. In Sect. 2 we give some definitions and notations used in the sequel. In Sect. 3 we show the impact on a directed rounding mode on EFTs and prove that garanteed interval bounds can be obtained thanks to EFTs executed with directed rounding. In Sect. 4, 5, and 6 we study the behaviour with directed rounding of compensated algorithms for respectively summation, dot product, and polynomial evaluation and show how they can provide narrow inclusions. Numerical experiments carried out using INTLAB [16] are presented in Sect. 7. Finally,

- *S. Graillat is with Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, LIP6, F-75005 Paris, France. E-mail: Stef.Graillat@lip6.fr*
- *Fabienne Jézéquel is with Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, LIP6, F-75005 Paris, France and with Université Panthéon-Assas, 12 place du Panthéon, 75231 Paris CEDEX 05, France. E-mail: Fabienne.Jezequel@lip6.fr*

1. URL address: http://www.olcf.ornl.gov/summit

conclusions and perspectives on this work are given in Sect. 8.

2 Definitions and notations

In this paper, we assume to work with a binary floating-point arithmetic adhering to IEEE 754-2008 floating-point standard [1] and we suppose that no overflow occurs. The error bounds for the compensated summation that are presented in Sect. 4 remain valid in the presence of underflow. For the other compensated algorithms considered in this article (dot product and Horner scheme) we assume that no underflow occurs so as to present simpler error bounds.

The set of floating-point numbers is denoted by $\mathbb{F}$, the bound on relative error for round to nearest by $\mathbf{u}$. With the IEEE 754 *binary64* format (double precision), we have $\mathbf{u} = 2^{-53}$ and with the *binary32* format (single precision), $\mathbf{u} = 2^{-24}$.

We denote by $\mathrm{fl}_*(\cdot)$ the result of a floating-point computation, where all operations inside parentheses are done in floating-point working precision with a directed rounding (that is to say toward $-\infty$ or $+\infty$). Floating-point operations in IEEE 754 satisfy [17]
For $\circ = \{+, -\}$, $\exists\, \varepsilon_1 \in \mathbb{R}$, $\varepsilon_2 \in \mathbb{R}$ such that

$$\mathrm{fl}_*(a \circ b) = (a \circ b)(1+\varepsilon_1) = (a \circ b)/(1+\varepsilon_2) \text{ with } |\varepsilon_\nu| \leq 2\mathbf{u}.$$

As a consequence, for $\circ = \{+, -\}$,

$$|a \circ b - \mathrm{fl}_*(a \circ b)| \leq 2\mathbf{u}|a \circ b| \text{ and } |a \circ b - \mathrm{fl}_*(a \circ b)| \leq 2\mathbf{u}|\mathrm{fl}_*(a \circ b)|.$$

We use standard notations for error estimations. The quantities γ_n are defined as usual [17] by

$$\gamma_n(\mathbf{u}) := \frac{n\mathbf{u}}{1 - n\mathbf{u}} \quad \text{for } n \in \mathbb{N},$$

where it is implicitly assumed that $n\mathbf{u} < 1$.

Remark 1. *We give the following relations on γ_n, that will be frequently used in the sequel of the paper. For any positive integer n, $n\mathbf{u} \leq \gamma_n(\mathbf{u})$, $\gamma_n(\mathbf{u}) \leq \gamma_{n+1}(\mathbf{u})$, $(1+\mathbf{u})\gamma_n(\mathbf{u}) \leq \gamma_{n+1}(\mathbf{u})$, $2n\mathbf{u}(1+\gamma_{2n-2}(\mathbf{u})) \leq \gamma_{2n}(\mathbf{u})$.*

Remark 2. *Recently, it has been shown that classic Wilkinson-type error bounds for summation, dot product and polynomial evaluation [18], [19], [20] can be slightly improved by replacing the factor $\gamma_n(\mathbf{u})$ by $n\mathbf{u}$ with no condition on n (for summation, dot product and Horner scheme). It is likely that the error bounds gien in this paper could also be slightly improved by replacing all the $\gamma_n(\mathbf{u})$ by $n\mathbf{u}$. However the proofs for improving the bounds would be more complicated and tricky, and would not be useful for this paper. We just aim at showing that the relative accuracy is in $\mathcal{O}(\mathbf{u})$ for classic algorithms and in $\mathcal{O}(\mathbf{u}^2)$ for compensated algorithms with directed roundings.*

3 Error-free transformations with directed rounding

3.1 Error-free transformations for addition

EFTs exist for the sum of two floating-point numbers with rounding to nearest: `FastTwoSum` [21], given as Algorithm 1, which requires a test and 3 floating-point operations, and `TwoSum` [22], given as Algorithm 2, which requires 6 floating-point operations. These algorithms compute both the floating-point sum x of two numbers a and b and the associated rounding error y such that $x + y = a + b$ when using rounding to the nearest. This is no more true with directed rounding. Indeed, with directed rounding, the rounding error may not be exactly representable (see [23] page 125).

We will study the behaviour of `FastTwoSum` and `TwoSum` with directed rounding. In the rest of this section, any arithmetic operation is rounded using the fl_* function defined in Sect. 2. In the Propositions presented in this section, and also in Sect. 4.2, we assume underflow may occur because, in this case, additions or subtractions generate no rounding error if subnormal numbers are available [24].

3.1.1 `FastTwoSum` *with directed rounding*

With rounding to nearest, the `FastTwoSum` EFT, given in Algorithm 1, computes the floating-point sum x of two numbers a and b and its associated rounding error y.

function $[x, y]$ = `FastTwoSum`(a, b)

1: **if** $|b| > |a|$ **then**
2: exchange a and b
3: **end if**
4: $x \leftarrow a + b$
5: $z \leftarrow c - a$
6: $y \leftarrow b - z$

Algorithm 1: Error-free transformation for the sum of two floating-point numbers with rounding to nearest

In [25], it is shown that the floating-point number z in Algorithm 1 is computed exactly with directed rounding. This property is recalled as Proposition 3.1.

Proposition 3.1. *The floating-point number z provided by Algorithm 1 using directed rounding is computed exactly, i.e. $z = x - a$.*

In general the correction y computed by Algorithm 1 using directed rounding is different from the rounding error e on the sum of a and b. In Proposition 3.2, we bound the difference between e and y.

Proposition 3.2. *Let x and y be the floating-point addition of a and b and the correction both computed by Algorithm 1 using directed rounding. Let e be the error on x: $a + b = x + e$. Then*

$$|e - y| \leq 4\mathbf{u}^2|a + b| \quad \text{and} \quad |e - y| \leq 4\mathbf{u}^2|x|.$$

Proof. From Proposition 3.1, z is computed exactly. However with directed rounding, y may not be computed exactly. So $\delta \in \mathbb{R}$ exists such that

$$y = b - z + \delta \tag{1}$$

and

$$|\delta| \leq 2\mathbf{u}|b - z|. \tag{2}$$

From Proposition 3.1, we deduce

$$|\delta| \leq 2\mathbf{u}|a + b - x| \tag{3}$$

Let e be the error on the floating-point addition of a and b, then

$$a + b = x + e \tag{4}$$

with

$$|e| \leq 2\mathbf{u}|a+b| \quad \text{and} \quad |e| \leq 2\mathbf{u}|x|. \tag{5}$$

From Equations 3 and 4, we deduce a bound on $|\delta| = |e-y|$:

$$|\delta| \leq 4\mathbf{u}^2|a+b| \quad \text{and} \quad |\delta| \leq 4\mathbf{u}^2|x|. \tag{6}$$

□

In Proposition 3.3 we establish a relation between the error e and the correction y if Algorithm 1 is executed with directed rounding.

Proposition 3.3. *Let x and y be the floating-point addition of a and b and the correction both computed by Algorithm 1 using directed rounding. Let e be the error on x: $a+b=x+e$.*

- *If computations are performed with rounding toward $+\infty$ then $e \leq y$.*
- *If computations are performed with rounding toward $-\infty$ then $y \leq e$.*

Proof. We always have by definition $a+b=x+e$.

- Let us assume computations are performed with rounding toward $+\infty$.
  In this case, we have $a+b \leq x$. Moreover from Proposition 3.1, we know that $z = x-a$ and still with rounding toward $+\infty$, we have $b-z \leq y$. As a consequence, we have $b-(x-a) \leq y$ and so $a+b-x \leq y$ which means that $e \leq y$.
- Let us assume computations are performed with rounding toward $-\infty$.
  In this case, we have $x \leq a+b$. Moreover from Proposition 3.1, we know that $z = x-a$ and still with rounding toward $-\infty$, we have $y \leq b-z$. As a consequence, we have $y \leq b-(x-a)$ and so $y \leq a+b-x$ which means that $y \leq e$.

□

3.1.2 `TwoSum` *with directed rounding*

With rounding to nearest, the `TwoSum` EFT, given in Algorithm 2, computes the floating-point sum x of two numbers a and b and its associated rounding error y.

function $[x,y]$ = `TwoSum`(a,b)

1: $x \leftarrow a+b$
2: $d \leftarrow x-a$
3: $f \leftarrow b-d$
4: $g \leftarrow x-d$
5: $h \leftarrow a-g$
6: $y \leftarrow f+h$

Algorithm 2: Error-free transformation for the sum of two floating-point numbers with rounding to nearest

We recall here a result of [14].

Theorem 3.4 ([14, Thm. 4.1])**.** *Let x and y be the floating-point addition of a and b and the correction both computed by Algorithm 2 using directed rounding. Let e be the error on x: $a+b=x+e$. Then*

$$|e-y| \leq 4\mathbf{u}^2|a+b| \qquad \textit{and} \qquad |e-y| \leq 4\mathbf{u}^2|x|.$$

Proposition 3.6 has been established using Sterbenz's lemma [26] which is recalled below. As a remark, Sterbenz's lemma is valid with directed rounding.

Lemma 3.5 (Sterbenz)**.** *In a floating-point system with subnormal numbers available, if c and d are finite floating-point numbers such that $d/2 \leq c \leq 2d$, then $c-d$ is exactly representable.*

In Proposition 3.6 we establish a relation between the error e and the correction y if Algorithm 2 is executed with directed rounding.

Proposition 3.6. *Let x and y be the floating-point addition of a and b and the correction both computed by Algorithm 2 using directed rounding. Let e be the error on x: $a+b=x+e$.*

- *If computations are performed with rounding toward $+\infty$ then $e \leq y$.*
- *If computations are performed with rounding toward $-\infty$ then $y \leq e$.*

Proof. Without loss of generality, we can assume that $a \geq 0$. We will separate the proof into three different cases: $|b| \geq a$, $-a < b \leq -a/2$ and $-a/2 < b < a$.

- case 1: $|b| \geq a$
  In this case, the lines 1, 2 and 3 correspond exactly to `FastTwoSum` (Algorithm 1). It follows that $d = x-a$ and so $f = \mathrm{fl}(a+b-x)$, $g=a$, $h=0$ and $y=f$. As a consequence, $y = \mathrm{fl}(e)$. So if we use rounding toward $+\infty$ then $e \leq y$ and if we use rounding toward $-\infty$ then $y \leq e$.
- case 2: $-a < b \leq -a/2$
  Using Sterbenz's lemma, it follows that $x = a+b$ and so $d=b$, $f=0$, $g=a$, $h=0$ and $y=0$. So in this case, we have $e=y=0$.
- case 3: $-a/2 < b < a$
  It follows from [14, Thm 4.1] that computations in lines 3 and 4 are exact due to Sterbenz's lemma. As a consequence, $f = b-d$ and $g = x-d$. Let us now assume we use rounding toward $+\infty$. As a consequence, $f+h \leq y$ and $a-g \leq h$ so $f+a-g \leq y$. Using the fact that $f = b-d$ and $g = x-d$, we obtain that $e = a+b-x \leq y$.
  Let us now assume we use rounding toward $-\infty$. We have $y \leq f+h$ and $h \leq a-g$ so $y \leq f+a-g$. Using the fact that $f = b-d$ and $g = x-d$, we obtain that $y \leq a+b-x = e$.

This concludes the proof. □

3.2 Error-free transformations for multiplication

3.2.1 `TwoProdFMA` *with directed rounding*

With any rounding mode, the `TwoProdFMA` EFT, given in Algorithm 3, computes both the floating-point product x of two numbers a and b and the associated rounding error y, provided that no underflow occurs. If this property holds, the floating-point numbers x and y computed by the `TwoProdFMA` algorithm satisfy: $x+y = a \times b$.

The `TwoProdFMA` algorithm is based on the *Fused-Multiply-and-Add* (`FMA`) operator that enables a floating-point multiplication followed by an addition to be performed as a single floating-point operation. For $a,b,c \in \mathbb{F}$,

function $[x, y]$ = TwoProdFMA(a, b)

1: $x \leftarrow a \times b$
2: $y \leftarrow \mathtt{FMA}(a, b, -x)$

Algorithm 3: Error-free transformation for the product of two floating-point numbers using an FMA

$\mathtt{FMA}(a, b, c)$ is an approximation of $a \times b + c \in \mathbb{R}$ that satisfies, if no underflow occurs:

$$\mathtt{FMA}(a, b, c) = (a \times b + c)(1 + \varepsilon_1) = (a \times b + c)/(1 + \varepsilon_2)$$

where $|\varepsilon_\nu| \leq \mathbf{u}$ with rounding to nearest and $|\varepsilon_\nu| \leq 2\mathbf{u}$ with directed rounding. The FMA operation is supported by numerous processors such as AMD or Intel processors starting with respectively the Bulldozer or the Haswell architecture and by the Intel Xeon Phi coprocessor. It is also supported by AMD and NVidia GPUs (Graphics Processing Units) since 2010.

3.2.2 TwoProduct *with directed rounding*

If no FMA is available, with rounding to nearest, the TwoProduct EFT from Veltkamp (see [21]), given in Algorithm 5, computes the product x of two floating-point numbers a and b and its associated rounding error y. The TwoProduct algorithm requires the Split algorithm [21], given in Algorithm 4. Let p be given by $\mathbf{u} = 2^{-p}$ and let us define $s = \lceil p/2 \rceil$. For example, if the working precision is IEEE 754 double precision, then $p = 53$ and $s = 27$. Algorithm 4 splits a floating-point number $a \in \mathbb{F}$ into two parts x and y such that

$$a = x + y \quad \text{with } |y| \leq |x|. \tag{7}$$

Both parts x and y have at most $s - 1$ non-zero bits.

We present here the behaviour of Algorithms 4 and 5 with directed rounding. Let $r \in \mathbb{R}$ be positive and $\mathrm{fl}(r)$ be a faithful correct rounding (to nearest, toward $+\infty$ or $-\infty$). We denote $\mathrm{ufp}(r) = 2^{\lfloor \log_2(r) \rfloor}$ if $r \neq 0$ and $\mathrm{ufp}(0) = 0$ as introduced in [27]. As a consequence, $\mathrm{ufp}(r) = 2^k$ with $k \in \mathbb{N}$. It is easy to show that if $\sigma = 2^k$, $k \in \mathbb{Z}$ and $r \in \mathbb{R}$ such that $r \in 2\mathbf{u}\sigma\mathbb{Z}$ and $|r| \leq 2\sigma$ then $r \in \mathbb{F}$. If $r \in \mathbb{R}$ and $\widetilde{r} := \mathrm{fl}(r) \in \mathbb{F}$ then we always have $\mathrm{ufp}(r) \leq \mathrm{ufp}(\widetilde{r})$ and $|\widetilde{r} - r| \leq 2\mathbf{u}\,\mathrm{ufp}(r) \leq 2\mathbf{u}\,\mathrm{ufp}(\widetilde{r})$.

function $[x, y]$ = Split(a)

1: $c \leftarrow (2^s + 1) \times a$
2: $d \leftarrow c - a$
3: $x \leftarrow c - d$
4: $y \leftarrow a - x$

Algorithm 4: Error-free split of a floating-point number into two parts

Lemma 3.7. *Assume that computations are done with a directed rounding mode (either toward $+\infty$ or $-\infty$). Let $a \in \mathbb{F}$ and $[x, y] = \mathtt{Split}(a)$. Then we have $a = x + y$ and*

- *the significand of x fits in $p - s$ bits;*
- *the significand of y fits in s bits.*

Proof. We can assume that a is not a power of 2 and $a > 0$. Otherwise, all the operations are exact and the result is clear. Let us define $\sigma = \mathrm{ufp}(a)$ so that $\mathrm{ufp}(a) < a < 2\,\mathrm{ufp}(a)$ that is to say $\sigma < a < 2\sigma$. As a is a floating-point number, we also have $\sigma(1 + 2\mathbf{u}) \leq a \leq 2\sigma(1 - \mathbf{u})$. It implies that $a \in 2\mathbf{u}\sigma\mathbb{Z}$ and $\mathrm{ufp}(2^s a) = 2^s\sigma$. By definition, we have that $2^s a$ is a floating-point number and $c = \mathrm{fl}((2^s a) + a)$. As $s \geq 2$ and $a > 0$, we either have $\mathrm{ufp}(c) = 2^s\sigma$ or $\mathrm{ufp}(c) = 2^{s+1}\sigma$.

1) $\mathrm{ufp}(c) = 2^s\sigma$
As $c - a < c$ since $a > 0$, we know that $d \leq c$ and so $\mathrm{ufp}(d) \leq \mathrm{ufp}(c) = 2^s\sigma$. Moreover since $\sigma(1 + 2\mathbf{u}) \leq a \leq 2\sigma(1 - \mathbf{u})$, we have $2^s\sigma(1 + 2\mathbf{u}) + a \leq 2^s a + a$. As $c = \mathrm{fl}((2^s a) + a)$, it follows that $c \geq 2^s a + a - 2\mathbf{u}\,\mathrm{ufp}(c) = 2^s a + a - 2^{s+1}\mathbf{u}\sigma$. Combining the two previous inequalities implies $c \geq 2^s\sigma + a$. As a consequence, $d = \mathrm{fl}(c - a) \geq 2^s\sigma$. We can conclude that necessarily $\mathrm{ufp}(d) = 2^s\sigma$.
As $c = \mathrm{fl}((2^s a) + a)$, we also have $c = (2^s + 1)a + e_1$ with $|e_1| \leq 2\mathbf{u}\,\mathrm{ufp}(c) \leq 2^{s+1}\mathbf{u}\sigma$. Moreover, $c \in 2\mathbf{u}\,\mathrm{ufp}(c)\mathbb{Z}$ and so $c \in 2^{s+1}\mathbf{u}\sigma\mathbb{Z}$. As $d = \mathrm{fl}(c - a)$ and $\mathrm{ufp}(d) = 2^s\sigma$ then necessarily $d = c - a + e_2$ with $|e_2| \leq 2\mathbf{u}\,\mathrm{ufp}(d) = 2^{s+1}\mathbf{u}\sigma$. We also have $d \in 2\mathbf{u}\,\mathrm{ufp}(d)\mathbb{Z}$ and so $d \in 2^{s+1}\mathbf{u}\sigma$.
As long as $s \geq 2$, $d = 2^s a + e_1 + e_2$ and $c = 2^s a + a + e_1$ are within a factor 2 and so using Sterbenz's lemma yields to the fact that $x = c - d$ (no rounding error during the addition). As a consequence, $x = a - e_2$. We know that $c \in 2^{s+1}\mathbf{u}\sigma\mathbb{Z}$ and $d \in 2^{s+1}\mathbf{u}\sigma\mathbb{Z}$ so $x = c - d \in 2^{s+1}\mathbf{u}\sigma\mathbb{Z}$. Moreover $|x| \leq |a| + |e_2| < 2\sigma + 2^{s+1}\mathbf{u}\sigma$. As $x \in 2^{s+1}\mathbf{u}\sigma\mathbb{Z}$ and $|x| < 2\sigma + 2^{s+1}\mathbf{u}\sigma$ implies that $|x| \leq 2\sigma$.
Since $x = a - e_2$ and a are very close, Sterbenz's lemma says that $y = a - x$ is exact and so $y = a - x = e_2$. It follows that $|y| \leq 2^{s+1}\mathbf{u}\sigma$ and $y \in 2\mathbf{u}\sigma\mathbb{Z}$ since $a, x \in 2\mathbf{u}\sigma\mathbb{Z}$.
Thus we have $x + y = a$ and since $x \leq 2\sigma$ and $x \in 2^{s+1}\mathbf{u}\sigma\mathbb{Z}$, this implies that x fits in $p - s$ bits. Besides, $|y| \leq 2^{s+1}\mathbf{u}\sigma$ and $y \in 2\mathbf{u}\sigma\mathbb{Z}$ implies that y fits in s bits.

2) $\mathrm{ufp}(c) = 2^{s+1}\sigma$
In that case, we either have $\mathrm{ufp}(d) = 2^{s+1}\sigma$ or $\mathrm{ufp}(d) = 2^s\sigma$. If $\mathrm{ufp}(d) = 2^s\sigma$ then $d \in 2^{s+1}\mathbf{u}\sigma$ and as $c \in 2\mathbf{u}\,\mathrm{ufp}(c)\mathbb{Z}$ and $2^s\sigma \leq \mathrm{ufp}(c)$ then $c \in 2^{s+1}\mathbf{u}\sigma\mathbb{Z}$ and the proof is similar to the previous case. Let us assume that $a \leq 2\sigma(1 - 3\mathbf{u})$. As $c \leq 2^s a + a + 2\mathbf{u}\,\mathrm{ufp}(c)$ so $c - a \leq 2^{s+1}\sigma(1 - 3\mathbf{u}) + 2^{s+2}\mathbf{u}\sigma$ which can be rewritten into $c - a \leq 2^{s+1}\sigma(1 - \mathbf{u})$. As $d = \mathrm{fl}(c - a)$ then $d \leq 2^{s+1}\sigma(1 - \mathbf{u})$ and so $\mathrm{ufp}(d) < 2^{s+1}\sigma$. As a consequence, the case $\mathrm{ufp}(d) = 2^{s+1}\sigma$ can only happen if $a = 2\sigma(1 - 2\mathbf{u})$ or $a = 2\sigma(1 - \mathbf{u})$.
If we use rounding toward $-\infty$:
If $a = 2\sigma(1 - \mathbf{u})$ then $2^s a + a = 2\sigma(2^s + 1)(1 - \mathbf{u})$ and so $c = 2\sigma(2^s + 1 - 2^{s+1}\mathbf{u})$ and so $c - a = 2\sigma(2^s - (2^{s+1} - 1)\mathbf{u})$ so $\mathrm{ufp}(d) = 2^s\sigma$ and this has been proved before.
If $a = 2\sigma(1 - 2\mathbf{u})$ then $2^s a + a = 2\sigma(2^s + 1)(1 - 2\mathbf{u})$ and so $c = 2\sigma(2^s + 1 - 2^{s+2}\mathbf{u})$ and so $c - a = 2\sigma(2^s - (2^{s+2} - 2)\mathbf{u})$ so $\mathrm{ufp}(d) = 2^s\sigma$ and this has been proved before.

If we use rounding toward $+\infty$:
If $a = 2\sigma(1-\mathbf{u})$ then $2^s a + a = 2\sigma(2^s+1)(1-\mathbf{u})$ and so $c = 2\sigma(2^s+1)$ and so $c - a = 2\sigma(2^s+\mathbf{u})$ and so $d = 2\sigma 2^s(1+2\mathbf{u})$. It follows that $x = 2\sigma(1-2^{s+1}\mathbf{u})$ and $y = 2\sigma(2^{s+1}-1)\mathbf{u}$
If $a = 2\sigma(1-2\mathbf{u})$ then $2^s a + a = 2\sigma(2^s+1)(1-2\mathbf{u})$ and so $c = 2\sigma(2^s + 1 - 2^{s+1}\mathbf{u})$ and so $c - a = 2\sigma(2^s - 2^{s+1}\mathbf{u} + 2\mathbf{u})$ and so $\mathrm{ufp}(d) = 2^s\sigma$ and this has been proved before.

This concludes the proof. □

With rounding to nearest, Algorithm 5 computes the product x of two floating-point numbers a and b and its associated rounding error y, *i. e.* such that $a \times b = x + y$.

function $[x, y]$ = `TwoProduct`(a, b)

1: $x \leftarrow a \times b$
2: $[a_1, a_2] \leftarrow$ `Split`(a)
3: $[b_1, b_2] \leftarrow$ `Split`(b)
4: $t_1 \leftarrow -x + (a_1 \times b_1)$
5: $t_2 \leftarrow t_1 + (a_1 \times b_2)$
6: $t_3 \leftarrow t_2 + (a_2 \times b_1)$
7: $y \leftarrow t_3 + (a_2 \times b_2)$

Algorithm 5: Error-free transformation of the product of two floating-point numbers with rounding to nearest

With directed rounding, `TwoProduct` does not necessarily return the generated rounding error even if this one is always a floating-point number. Indeed, a counter-example in rounding toward $-\infty$ can be chosen as follows. Let $a = 1+2\mathbf{u}$ and $b = 1+2\mathbf{u}$, then $x = 1+4\mathbf{u}$ and the rounding error is $4\mathbf{u}^2$ but `TwoProduct`(a, b) returns $y = 0$. In Proposition 3.8, we bound the difference between the rounding error e and the correction y computed by Algorithm 5 with directed rounding.

Proposition 3.8. *Let x and y be the floating-point product of a and b and the correction both computed by Algorithm 5 using directed rounding. Let e be the error on x: $a \times b = x + e$. Then*

$$|e - y| \le 8\mathbf{u}^2|a \times b| \quad \textit{and} \quad |e - y| \le 8\mathbf{u}^2|x|.$$

Proof. Let us denote $\sigma_1 := \mathrm{ufp}(a)$ and $\sigma_2 := \mathrm{ufp}(b)$. By definition of the splitting, the products a_1b_1, a_1b_2, a_2b_1 are exactly representable but this is not necessarily the case for a_2b_2. From `Split` algorithm, we have that $|a - a_1| \le 2^{s+1}\mathbf{u}\sigma_1$, $|b - b_1| \le 2^{s+1}\mathbf{u}\sigma_2$ and $|a_1| \le 2\sigma_1$, $|b_1| \le 2\sigma_2$. From

$$(ab - a_1b_1) = (a - a_1)b + (b - b_1)a_1,$$

we get that $|ab - a_1b_1| \le 2^{s+3}\mathbf{u}\sigma_1\sigma_2$. Moreover, it is clear that $|ab - x| \le 2\mathbf{u}\,\mathrm{ufp}(ab)$. As $\mathrm{ufp}(ab) \le 2\sigma_1\sigma_2$ then $|ab - x| \le 4\mathbf{u}\sigma_1\sigma_2$. As a consequence, $|a_1b_1 - x| \le 4\mathbf{u}\sigma_1\sigma_2 + 2^{s+3}\mathbf{u}\sigma_1\sigma_2$. It follows that a_1b_1 and x are very close and so by Sterbenz's lemma, we know that $t_1 = -x + a_1b_1$ is exact.

As $ab = a_1b_1 + a_1b_2 + a_2b_1 + a_2b_2$, we have

$$\begin{aligned} |t_1 + a_1b_2| &= |-x + a_1b_1 + a_1b_2| \\ &= |-x + ab + (a_1b_1 + a_1b_2 - ab)| \\ &\le |ab - x| + |a_2b_2 + a_2b_1|. \end{aligned}$$

As $|ab - x| \le 4\mathbf{u}\sigma_1\sigma_2$ and $|a_2b_2 + a_2b_1| \le |a_2||b| \le 2^{s+2}\mathbf{u}\sigma_1\sigma_2$, it follows that

$$|t_1 + a_1b_2| \le 2^{s+2}\mathbf{u}\sigma_1\sigma_2 + 4\mathbf{u}\sigma_1\sigma_2 < 2^{s+3}\mathbf{u}\sigma_1\sigma_2.$$

Since $a_1 \in 2^{s+1}\mathbf{u}\sigma_1\mathbb{Z}$ and $b_2 \in 2\mathbf{u}\sigma_2\mathbb{Z}$, it follows that $a_1b_2 \in 2^{s+2}\mathbf{u}^2\sigma_1\sigma_2\mathbb{Z}$ and so $t_1 + a_1b_2 \in 2^{s+2}\mathbf{u}^2\sigma_1\sigma_2$. This and $|t_1 + a_1b_2| < 2^{s+3}\mathbf{u}\sigma_1\sigma_2$ implies that $t_1 + a_1b_2$ is exactly representable.

It follows that $t_2 = -x + a_1b_1 + a_1b_2$ so $t_2 + a_2b_1 = -x + ab - a_2b_2$. As a consequence,

$$|t_2 + a_2b_1| \le |ab - x| + |a_2b_2| \le 4\mathbf{u}\sigma_1\sigma_2 + 2^{2s+2}\mathbf{u}^2\sigma_1\sigma_2.$$

As $s = \lceil p/2 \rceil$ and $\mathbf{u} = 2^{-p}$, it follows that $2^{2s}\mathbf{u} \le 2$, it follows that $|t_2 + a_2b_1| \le 4\mathbf{u}\sigma_1\sigma_2 + 2^3\mathbf{u}\sigma_1\sigma_2 \le 12\mathbf{u}\sigma_1\sigma_2$. As $t_2 + a_2b_1 \in 2^{s+2}\mathbf{u}^2\sigma_1\sigma_2\mathbb{Z}$, implies that $t_2 + a_2b_1$ is exactly representable.

It follows that $t_3 = t_2 + a_2b_1 = -x + a_1b_1 + a_1b_2 + a_2b_1 = -x + ab - a_2b_2$ and so

$$t_3 + a_2b_2 = ab - x.$$

So the error on the floating-point product (which is a floating-point number) is bounded by $\mathrm{fl}(t_3 + a_2b_2)$ if we used rounding toward $+\infty$. Moreover a_2b_2 have at most $p+1$ bits and $a_2b_2 \in 4\mathbf{u}^2\sigma_1\sigma_2\mathbb{Z}$ so $r = \mathrm{fl}(a_2b_2) \in 8\mathbf{u}^2\sigma_1\sigma_2\mathbb{Z}$ and so $t_3 + r \in 8\mathbf{u}^2\sigma_1\sigma_2\mathbb{Z}$ and $|t_3 + r| \le 4\mathbf{u}\sigma_1\sigma_2$. So $t_3 + r$ is exactly representable. So it follows that

$$|(ab - x) - y| = |a_2b_2 - r| \le 2\mathbf{u}|a_2b_2|.$$

As $|a_2b_2| \le 2^{s+2}\mathbf{u}^2\sigma_1\sigma_2$ and as $2^{2s} \le 2$, we obtain that

$$|(ab - x) - y| = |a_2b_2 - r| \le 8\mathbf{u}^2\sigma_1\sigma_2.$$

As we know that $\sigma_1\sigma_2 \le \mathrm{ufp}(ab) \le 2\sigma_1\sigma_2$, it follows that

$$|(ab - x) - y| \le 8\mathbf{u}^2\,\mathrm{ufp}(ab).$$

□

In Proposition 3.9 we establish a relation between the error e and the correction y if Algorithm 5 is executed with directed rounding.

Proposition 3.9. *Let x and y be the floating-point product of a and b and the correction both computed by Algorithm 5 using directed rounding. Let e be the error on x: $a \times b = x + e$.*

- *If computations are performed with rounding toward $+\infty$ then $e \le y$.*
- *If computations are performed with rounding toward $-\infty$ then $y \le e$.*

Proof. From the previous theorem, we know that $e = t_3 + a_2b_2$ and $y = \mathrm{fl}(t_3 + \mathrm{fl}(a_2b_2))$. As a consequence, if we perform computations with rounding toward $+\infty$ then $e \le y$ and if we perform computations with rounding toward $-\infty$ then $y \le e$. □

4 Accurate summation

In this section we recall how to obtain interval inclusions for summation using the classical iterative algorithm. Then we present how to compute narrow inclusions thanks to compensated algorithms.

function res = Sum(p)
1: $s_1 \leftarrow p_1$
2: **for** $i = 2$ to n **do**
3: $\quad s_i \leftarrow s_{i-1} + p_i$
4: **end for**
5: res $\leftarrow s_n$

Algorithm 6: Summation of n floating-point numbers $p = \{p_i\}$

4.1 Classic summation

The classic algorithm for summation is the iterative Algorithm 6.

The error generated by Algorithm 6 with directed rounding is given in [17] and is recalled in Proposition 4.1.

Proposition 4.1. *Let us suppose Algorithm 6 is applied to floating-point numbers $p_i \in \mathbb{F}$, $1 \leq i \leq n$. Let $s := \sum p_i$ and $S := \sum |p_i|$.*

With directed rounding, if $n\mathbf{u} < \frac{1}{2}$, then

$$|\texttt{res} - s| \leq \gamma_{n-1}(2\mathbf{u})S. \tag{8}$$

In Corollary 4.2 Equation 8 is rewritten in terms of the condition number on $\sum p_i$:

$$\operatorname{cond}\left(\sum p_i\right) = \frac{S}{|s|}.$$

Corollary 4.2. *With directed rounding, if $n\mathbf{u} < \frac{1}{2}$, the result* res *of Algorithm 6 satisfies*

$$\frac{|\texttt{res} - s|}{|s|} \leq \gamma_{n-1}(2\mathbf{u}) \operatorname{cond}\left(\sum p_i\right).$$

Because $\gamma_{n-1}(2\mathbf{u}) \approx 2(n-1)\mathbf{u}$, the bound for the relative error is essentially $2n\mathbf{u}$ times the condition number. If the condition number is large (greater than $1/\mathbf{u}$) then the result of Algorithm 6 has no more correct digits. Compensated algorithms, that evaluate more accurately the sum of floating-point numbers, are presented in Sect. 4.2.

Algorithm 7 shows how to compute an inclosure of $\sum_{i=1}^{n} p_i$. It is given with the MATLAB syntax. With the argument -1 (resp. 1), the `setround` function enables one to perform the next instructions with rounding to $-\infty$ (resp. $+\infty$). The same algorithm could also be writtten in a programming language sush as C++ using the `fesetround` function to change the rounding mode.

```
setround(-1)
Sinf = Sum(p)
setround(1)
Ssup = Sum(p)
```

Algorithm 7: Computation of interval bounds Sinf and Ssup with the classic summation algorithm Sum

As shown for example in [28], we have the following enclosure.

Proposition 4.3. *Let $p = \{p_i\}$ be a vector of n floating-point numbers. If* Sinf *and* Ssup *are computed using Algorithm 7, then we have*

$$\texttt{Sinf} \leq \sum_{i=1}^{n} p_i \leq \texttt{Ssup}.$$

4.2 Compensated summation with directed rounding

A compensated algorithm to evaluate accurately the sum of n floating-point numbers is presented as Algorithm 8 (FastCompSum) [29], [30]. This sum is corrected thanks to an error-free transformation used for each individual summation. Although FastTwoSum is called in Algorithm 8, with rounding to nearest the same result can be obtained using another error-free transformation (TwoSum).

function res = FastCompSum(p)
1: $\pi_1 \leftarrow p_1$
2: $\sigma_1 \leftarrow 0$
3: **for** $i = 2$ to n **do**
4: $\quad [\pi_i, q_i] \leftarrow \texttt{FastTwoSum}(\pi_{i-1}, p_i)$
5: $\quad \sigma_i \leftarrow \sigma_{i-1} + q_i$
6: **end for**
7: res $\leftarrow \pi_n + \sigma_n$

Algorithm 8: Compensated summation of n floating-point numbers $p = \{p_i\}$ using FastTwoSum

With directed rounding, Algorithm 1 (FastTwoSum) is not an error-free transformation. The error generated by Algorithm 8 with directed rounding is given in [13] and is recalled in Proposition 4.4.

Proposition 4.4. *Let us suppose Algorithm* FastCompSum *is applied, with directed rounding, to floating-point numbers $p_i \in \mathbb{F}$, $1 \leq i \leq n$. Let $s := \sum p_i$ and $S := \sum |p_i|$. If $n\mathbf{u} < \frac{1}{2}$, then*

$$|\texttt{res} - s| \leq 2\mathbf{u}|s| + 2(1+2\mathbf{u})\gamma_n^2(2\mathbf{u})S. \tag{9}$$

From Proposition 4.4, a bound for the relative error on the result of Algorithm 8 (FastCompSum) obtained with directed rounding is deduced in Corollary 4.5.

Corollary 4.5. *With directed rounding, if $n\mathbf{u} < \frac{1}{2}$, then, the result* res *of Algorithm 8 (*FastCompSum*) satisfies*

$$\frac{|\texttt{res} - s|}{|s|} \leq 2\mathbf{u} + 2(1+2\mathbf{u})\gamma_n^2(2\mathbf{u}) \operatorname{cond}\left(\sum p_i\right).$$

From Corollary 4.5, because $\gamma_n(2\mathbf{u}) \approx 2n\mathbf{u}$, the relative error bound is essentially $(n\mathbf{u})^2$ times the condition number plus the unavoidable rounding $2\mathbf{u}$ due to the working precision. The computation is carried out almost as with twice the working precision ($\mathbf{u}^2$).

Algorithm 9 shows how to compute with MATLAB the FastCompSum algorithm with rounding to $-\infty$, and then with rounding to $+\infty$.

```
setround(-1)
Sinf = FastCompSum(p)
setround(1)
Ssup = FastCompSum(p)
```

Algorithm 9: Computation of interval bounds Sinf and Ssup with the compensated summation algorithm FastCompSum

In Proposition 4.6 we show that Algorithm 9 provides an inclosure of $\sum_{i=1}^{n} p_i$. Thanks to the FastCompSum algorithm,

the results provided by Algorithm 9 are almost as accurate as if the classical summation was computed in twice the working precision.

Proposition 4.6. *Let $p = \{p_i\}$ be a vector of n floating-point numbers. If* `Sinf` *and* `Ssup` *are computed using Algorithm 9, then we have*

$$\texttt{Sinf} \leq \sum_{i=1}^{n} p_i \leq \texttt{Ssup}.$$

Proof. Let e_i be the error on the floating-point addition of π_{i-1} and p_i $(i = 2, ..., n)$. We know that $s = \sum_{i=1}^{n} p_i = \pi_n + \sum_{i=1}^{n} e_i$ where $\pi_i + e_i = \pi_{i-1} + p_i$.

- Let us assume computations are performed with rounding toward $+\infty$.
  From Proposition 3.2, it follows that $e_i \leq q_i$. As a consequence, we have $s \leq \pi_n + \sum_{i=1}^{n} q_i$. As we use rounding toward $+\infty$, we have $\sum_{i=1}^{n} q_i \leq \sigma_n$ so $s \leq \pi_n + \sigma_n$. As we always use rounding toward $+\infty$, we also have $s \leq \texttt{res} := \texttt{Ssup}$.
- Let us assume computations are performed with rounding toward $-\infty$.
  From Proposition 3.2, it follows that $q_i \leq e_i$. As a consequence, we have $\pi_n + \sum_{i=1}^{n} q_i \leq s$. As we use rounding toward $-\infty$, we have $\sigma_n \leq \sum_{i=1}^{n} q_i$ so $\pi_n + \sigma_n \leq s$. As we always use rounding toward $-\infty$, we also have $\texttt{Sinf} := \texttt{res} \leq s$.

□

A compensated summation algorithm based on `TwoSum` is given in Algorithm 10 (`CompSum`). This algorithm was introduced in [9].

function res = CompSum(p)

1: $\pi_1 \leftarrow p_1$
2: $\sigma_1 \leftarrow 0$
3: **for** $i = 2$ to n **do**
4: $\quad [\pi_i, q_i] \leftarrow \texttt{TwoSum}(\pi_{i-1}, p_i)$
5: $\quad \sigma_i \leftarrow \sigma_{i-1} + q_i$
6: **end for**
7: $\texttt{res} \leftarrow \pi_n + \sigma_n$

Algorithm 10: Compensated summation of n floating-point numbers $p = \{p_i\}$ using `TwoSum`

Proposition 4.7 shows that the error bound established for the `FastCompSum` algorithm is also valid for `CompSum`.

Proposition 4.7. *Let us suppose Algorithm* `CompSum` *is applied, with directed rounding, to floating-point numbers $p_i \in \mathbb{F}$, $1 \leq i \leq n$. Let $s := \sum p_i$ and $S := \sum |p_i|$. If $n\mathbf{u} < \frac{1}{2}$, then*

$$|\texttt{res} - s| \leq 2\mathbf{u}|s| + 2(1 + 2\mathbf{u})\gamma_n^2(2\mathbf{u})S. \tag{10}$$

Proof. The error bounds for `FastTwoSum` and `TwoSum` are the same as shown in Propositions 3.2 and 3.6. As the consequence, the proof is similar to the one for `FastCompSum` (see Proposition 4.4). □

Algorithm 11 shows how to compute the `CompSum` algorithm with rounding to $-\infty$, and then with rounding to $+\infty$.

```
setround(-1)
Sinf = CompSum(p)
setround(1)
Ssup = CompSum(p)
```

Algorithm 11: Computation of interval bounds `Sinf` and `Ssup` with the compensated summation algorithm `CompSum`

Proposition 4.8 shows that Algorithm 11 provides an inclosure of $\sum_{i=1}^{n} p_i$. The results of Algorithm 11, like those of Algorithm 9, are almost as accurate as if the classical summation was computed in twice the working precision.

Proposition 4.8. *Let $p = \{p_i\}$ be a vector of n floating-point numbers. If* `Sinf` *and* `Ssup` *are computed using Algorithm 11, then we have*

$$\texttt{Sinf} \leq \sum_{i=1}^{n} p_i \leq \texttt{Ssup}.$$

Proof. The proof is similar to the one of Proposition 4.6. □

5 Accurate dot product

In this section we recall how to obtain inclusions of dot products using the classic dot product algorithm. Then we show that tighter inclusions can be computed using compensated dot product algorithms executed with directed rounding. In this section, we assume that no underflow occurs.

5.1 Classic dot product

The classic algorithm for computing a dot product is Algorithm 12.

function res = Dot(x, y)

1: $s_1 \leftarrow x_1 y_1$
2: **for** $i = 2$ to n **do**
3: $\quad s_i \leftarrow x_i \times y_i + s_{i-1}$
4: **end for**
5: $\texttt{res} \leftarrow s_n$

Algorithm 12: Classic dot product of $x = \{x_i\}$ and $y = \{y_i\}$, $1 \leq i \leq n$

The error generated by Algorithm 12 with directed rounding is recalled in Proposition 5.1.

Proposition 5.1. *Let floating point numbers $x_i, y_i \in \mathbb{F}, 1 \leq i \leq n$, be given and denote by $\texttt{res} \in \mathbb{F}$ the result computed by Algorithm 12* (`Dot`)*. With directed rounding, if $n\mathbf{u} < \frac{1}{2}$, we have*

$$|\texttt{res} - x^T y| \leq \gamma_n(2\mathbf{u})|x^T||y|. \tag{11}$$

Proof. The proof can be found in Higham [17, p.63]. □

We can rewrite the previous inequality in terms of the condition number of the dot product defined by

$$\text{cond}(x^T y) = 2\frac{|x|^T|y|}{|x^T y|}.$$

Corollary 5.2. *With directed rounding, if* $n\mathbf{u} < \frac{1}{2}$*, the result* res *of Algorithm 12 satisfies*

$$\frac{|\texttt{res} - x^T y|}{|x^T y|} \leq \frac{1}{2}\gamma_n(2\mathbf{u})\,\mathrm{cond}(x^T y).$$

Because $\gamma_n(2\mathbf{u}) \approx 2n\mathbf{u}$, the bound for the relative error is essentially $n\mathbf{u}$ times the condition number.

Algorithm 13 shows how to compute the Dot algorithm with rounding to $-\infty$, and then with rounding to $+\infty$.

```
setround(-1)
Dinf = Dot(x,y)
setround(1)
Dsup = Dot(x,y)
```

Algorithm 13: Computation of interval bounds Dinf and Dsup with the classic dot product algorithm Dot

As shown for example in [28], we have the following enclosure.

Proposition 5.3. *Let floating-point numbers* $x_i, y_i \in \mathbb{F}, 1 \leq i \leq n$*, be given. If* Dinf *and* Dsup *are computed using Algorithm 13, then we have*

$$\texttt{Dinf} \leq x^T y \leq \texttt{Dsup}.$$

5.2 Compensated dot product with directed rounding and FMA

A compensated dot product algorithm [9] that uses the TwoProdFMA EFT is recalled as Algorithm 14 (CompDotFMA).

function res = CompDotFMA(x, y)

1: $[p, s] \leftarrow$ TwoProdFMA(x_1, y_1)
2: **for** $i = 2$ to n **do**
3: $\quad [h, r] \leftarrow$ TwoProdFMA(x_i, y_i)
4: $\quad [p, q] \leftarrow$ TwoSum(p, h)
5: $\quad s \leftarrow s + (q + r)$
6: **end for**
7: res $\leftarrow p + s$

Algorithm 14: Compensated dot product of $x = \{x_i\}$ and $y = \{y_i\}$, $1 \leq i \leq n$ with FMA.

A bound for the absolute error on the result res of Algorithm 14 with directed rounding is given in Proposition 5.4.

Proposition 5.4. *Let floating-point numbers* $x_i, y_i \in \mathbb{F}, 1 \leq i \leq n$*, be given and denote by* $\texttt{res} \in \mathbb{F}$ *the result computed by Algorithm 14 with directed rounding. If* $(n+1)\mathbf{u} < \frac{1}{2}$*, then,*

$$|\texttt{res} - x^T y| \leq 2\mathbf{u}|x^T y| + 2\gamma^2_{n+1}(2\mathbf{u})|x^T||y|.$$

Proof. In [15], a similar algorithm has been analyzed with directed rounding, except FastTwoSum was used instead of TwoSum here. Because the error bounds are the same in Proposition 3.2 and Theorem 3.4, the error bound in Proposition 5.4 is the same as in [15]. □

From Proposition 5.4, a bound for the relative error on the result of Algorithm 14 obtained with directed rounding is deduced in Corollary 5.5.

Corollary 5.5. *With directed rounding, if* $(n+1)\mathbf{u} < \frac{1}{2}$*, then, the result* res *of Algorithm 14 satisfies*

$$\frac{|\texttt{res} - x^T y|}{|x^T y|} \leq 2\mathbf{u} + \gamma^2_{n+1}(2\mathbf{u})\,\mathrm{cond}\left(x^T y\right).$$

From Corollary 5.5, the relative error bound on the result of Algorithm 14 computed with directed rounding is essentially $(n\mathbf{u})^2$ times the condition number plus the rounding $2\mathbf{u}$ due to the working precision. The result obtained with Algorithm 14 is almost as accurate as if the classic dot product was computed in twice the working precision.

Algorithm 15 shows how to compute with MATLAB the CompDotFMA algorithm with rounding to $-\infty$, and then with rounding to $+\infty$.

```
setround(-1)
Dinf = CompDotFMA(x,y)
setround(1)
Dsup = CompDotFMA(x,y)
```

Algorithm 15: Computation of interval bounds Dinf and Dsup with the compensated dot product algorithm CompDotFMA

In Proposition 5.6 we show that Algorithm 15 provides an inclosure of the dot product. For the proof we rewrite this algorithm into the following equivalent one.

function res=CompDotFMA(x, y)

1: $[p_1, s_1] \leftarrow$ TwoProdFMA(x_1, y_1)
2: **for** $i = 2$ to n **do**
3: $\quad [h_i, r_i] \leftarrow$ TwoProdFMA(x_i, y_i)
4: $\quad [p_i, q_i] \leftarrow$ TwoSum(p_{i-1}, h_i)
5: $\quad s_i \leftarrow s_{i-1} + (q_i + r_i)$
6: **end for**
7: res $\leftarrow p_n + s_n$

Algorithm 16: Equivalent formulation of Algorithm 14

Proposition 5.6. *Let floating-point numbers* $x_i, y_i \in \mathbb{F}, 1 \leq i \leq n$*, be given. If* Dinf *and* Dsup *are computed using Algorithm 15, then we have*

$$\texttt{Dinf} \leq x^T y \leq \texttt{Dsup}.$$

Proof. Let e_i be the error on the floating-point addition of p_{i-1} and h_i $(i = 2, ..., n)$. We know that $x^T y = p_n + s_1 + \sum_{i=2}^{n}(e_i + r_i)$ where $p_i + e_i = p_{i-1} + h_i$ (see Proposition 4.5 in [15]).

- Let us assume computations are performed with rounding toward $+\infty$.
  From Proposition 3.6, it follows that $e_i \leq q_i$. As a consequence, we have $x^T y \leq p_n + s_1 + \sum_{i=2}^{n}(q_i + r_i)$. As we use rounding toward $+\infty$, we have $s_1 + \sum_{i=2}^{n}(q_i + r_i) \leq s_n$ so $x^T y \leq p_n + s_n$. As we always use rounding toward $+\infty$, we also have $x^T y \leq \texttt{res} := \texttt{Dsup}$.
- Let us assume computations are performed with rounding toward $-\infty$.

From Proposition 3.6, it follows that $q_i \leq e_i$. As a consequence, we have $p_n + s_1 + \sum_{i=2}^{n}(q_i + r_i) \leq x^T y$. As we use rounding toward $-\infty$, we have $s_n \leq s_1 + \sum_{i=2}^{n}(q_i + r_i)$ so $p_n + s_n \leq x^T y$. As we always use rounding toward $-\infty$, we also have $\texttt{Dinf} := \texttt{res} \leq x^T y$.

□

5.3 Compensated dot product with directed rounding without FMA

If an FMA is not easily available, as it is the case with MATLAB, a compensated dot product algorithm similar to Algorithm 14 can be written by replacing TwoProdFMA by TwoProduct. This compensated dot product algorithm with no FMA is given as Algorithm 17 in a formulation convenient for the proofs of Propositions 5.7 and 5.8.

function res = CompDot(x, y)

1: $[p_1, s_1] \leftarrow$ TwoProduct(x_1, y_1)
2: **for** $i = 2$ to n **do**
3: $\quad [h_i, r_i] \leftarrow$ TwoProduct(x_i, y_i)
4: $\quad [p_i, q_i] \leftarrow$ TwoSum(p_{i-1}, h_i)
5: $\quad s_i \leftarrow s_{i-1} + (q_i + r_i)$
6: **end for**
7: res $\leftarrow p_n + s_n$

Algorithm 17: Compensated dot product of $x = \{x_i\}$ and $y = \{y_i\}$, $1 \leq i \leq n$ without FMA.

Proposition 5.7. *Let floating-point numbers $x_i, y_i \in \mathbb{F}, 1 \leq i \leq n$, be given and denote by $\texttt{res} \in \mathbb{F}$ the result computed by Algorithm 17 (CompDot) with directed rounding. If $(n+1)\mathbf{u} < \frac{1}{2}$, then,*

$$|\texttt{res} - x^T y| \leq 2\mathbf{u}|x^T y| + 2(1 + 2\mathbf{u})\gamma_{n+1}^2(2\mathbf{u})|x^T||y|.$$

Proof. Thanks to the TwoProduct algorithm, we have

$$p_1 + t_1 = x_1 y_1, \tag{12}$$

with $|t_1 - s_1| \leq 4\mathbf{u}^2|x_1 y_1|$ and for $i \geq 2$,

$$h_i + t_i = x_i y_i, \tag{13}$$

with $|t_i - r_i| \leq 4\mathbf{u}^2|x_i y_i|$. From Proposition 3.2, it follows that

$$p_i + e_i = p_{i-1} + h_i \quad \text{with} \quad |q_i - e_i| \leq 4\mathbf{u}^2|p_{i-1} + h_i|. \tag{14}$$

Therefore from Equation 13, we deduce that

$$e_i + t_i = (p_{i-1} + h_i - p_i) + (x_i y_i - h_i) = x_i y_i + p_{i-1} - p_i.$$

Then from Equation 12, we derive

$$s_1 + \sum_{i=2}^{n}(e_i + t_i) = (x_1 y_1 - p_1) + \left(\sum_{i=2}^{n} x_i y_i + p_1 - p_n\right) = x^T y - p_n. \tag{15}$$

We know that $|t_i| \leq 2\mathbf{u}|x_i y_i|$ and $|t_i - r_i| \leq 4\mathbf{u}^2|x_i y_i|$ for $i \geq 2$. As a consequence, for $i \geq 2$,

$$|r_i| \leq [2\mathbf{u} + 8\mathbf{u}^2]|x_i y_i|.$$

Therefore, we have

$$\sum_{i=2}^{n}|r_i| \leq [2\mathbf{u} + 8\mathbf{u}^2]\sum_{i=2}^{n}|x_i y_i|,$$

and

$$|s_1| + \sum_{i=2}^{n}|r_i| \leq [2\mathbf{u} + 8\mathbf{u}^2]|x^T||y|. \tag{16}$$

Let us denote $\alpha_i := e_i - q_i$ so that

$$|\alpha_i| \leq 4\mathbf{u}^2|\pi_{i-1} + p_i|. \tag{17}$$

Let us first evaluate an upper bound on $\sum_{i=2}^{n}|\alpha_i|$ and an upper bound for $\sum_{i=2}^{n}|e_i|$ and then an upper bound on $\sum_{i=2}^{n}|q_i|$. Let us show by induction that

$$\sum_{i=2}^{n}|\alpha_i| \leq 2\mathbf{u}\gamma_{n-1}(2\mathbf{u})\sum_{i=1}^{n}|h_i|. \tag{18}$$

By convention, we define $h_1 := p_1$. We know that if $n = 2$,

$$p_2 + e_2 = p_1 + h_2 = h_1 + h_2. \tag{19}$$

Therefore

$$|\alpha_2| \leq 4\mathbf{u}^2\left(|h_1| + |h_2|\right) \leq 2\mathbf{u}\gamma_1(2\mathbf{u})\left(|h_1| + |h_2|\right) \tag{20}$$

Let us assume that Equation 18 is true for n and that an extra floating-point number h_{n+1} is added. Then

$$p_{n+1} = \mathrm{fl}_*(p_n + h_{n+1}), \tag{21}$$

$$p_{n+1} = \mathrm{fl}_*\left(\sum_{i=1}^{n+1} h_i\right). \tag{22}$$

From [17],

$$|p_{n+1}| \leq (1 + \gamma_n(2\mathbf{u}))\sum_{i=1}^{n+1}|h_i|. \tag{23}$$

Let e_{n+1} be the error on the floating-point addition of p_n and h_{n+1}:

$$p_{n+1} + e_{n+1} = p_n + h_{n+1}. \tag{24}$$

From Proposition 3.8,

$$|\alpha_{n+1}| \leq 4\mathbf{u}^2|p_{n+1}| \leq 4\mathbf{u}^2\left(1 + \gamma_n(2\mathbf{u})\right)\sum_{i=1}^{n+1}|h_i| \tag{25}$$

Hence, assuming that Equation 18 is true for n,

$$\sum_{i=2}^{n+1}|\alpha_i| \leq \left(2\mathbf{u}\gamma_{n-1}(2\mathbf{u}) + 4\mathbf{u}^2(1 + \gamma_n(2\mathbf{u}))\right)\sum_{i=1}^{n+1}|h_i| \tag{26}$$

From the fact that a direct calculation shows that $\gamma_{n-1}(2\mathbf{u}) + 2\mathbf{u}(1 + \gamma_n(2\mathbf{u})) \leq \gamma_n(2\mathbf{u})$, we can deduce

$$\sum_{i=2}^{n+1}|\alpha_i| \leq 2\mathbf{u}\gamma_n(2\mathbf{u})\sum_{i=1}^{n+1}|h_i| \tag{27}$$

Therefore by induction Equation 18 is true.

Let us now find an upper bound for $\sum_{i=2}^{n}|e_i|$. Let us show by induction that

$$\sum_{i=2}^{n} |e_i| \leq \gamma_{n-1}(2\mathbf{u}) \sum_{i=1}^{n} |h_i| \tag{28}$$

We know that if $n = 2$,

$$p_2 + e_2 = p_1 + h_2 = h_1 + h_2. \tag{29}$$

Therefore

$$|e_2| \leq \gamma_1(2\mathbf{u}) \left(|h_1| + |h_2|\right) \tag{30}$$

Let us assume that Equation 28 is true for n and that an extra floating-point number h_{n+1} is added.

From Equations 21 to 24,

$$|e_{n+1}| \leq 2\mathbf{u}|p_{n+1}| \leq 2\mathbf{u}\left(1 + \gamma_n(2\mathbf{u})\right) \sum_{i=1}^{n+1} |h_i| \tag{31}$$

Hence, assuming that Equation 28 is true for n,

$$\sum_{i=2}^{n+1} |e_i| \leq \left(\gamma_{n-1}(2\mathbf{u}) + 2\mathbf{u}(1 + \gamma_n(2\mathbf{u}))\right) \sum_{i=1}^{n+1} |h_i| \tag{32}$$

By a calculation, we deduce

$$\sum_{i=2}^{n+1} |e_i| \leq \gamma_n(2\mathbf{u}) \sum_{i=1}^{n+1} |h_i| \tag{33}$$

Therefore by induction Equation 28 is true.

Let us evaluate an upper bound on $\sum_{i=2}^{n} |q_i|$:

$$\sum_{i=2}^{n} |q_i| \leq \sum_{i=2}^{n} |e_i| + \sum_{i=2}^{n} |q_i - e_i| = \sum_{i=2}^{n} |e_i| + \sum_{i=2}^{n} |\alpha_i| \tag{34}$$

From Equations 17 and 28,

$$\sum_{i=2}^{n} |q_i| \leq \gamma_{n-1}(2\mathbf{u}) \sum_{i=1}^{n} |h_i| + 2\mathbf{u}\gamma_{n-1}(2\mathbf{u}) \sum_{i=1}^{n} |h_i| \tag{35}$$

Therefore

$$\sum_{i=2}^{n} |q_i| \leq \left(\gamma_{n-1}(2\mathbf{u}) + 2\mathbf{u}\gamma_{n-1}(2\mathbf{u})\right) \sum_{i=1}^{n} |h_i| \tag{36}$$

We then deduce

$$\sum_{i=2}^{n} |q_i| \leq \gamma_n(2\mathbf{u}) \sum_{i=1}^{n} |h_i| \tag{37}$$

As a consequence, we have

$$\sum_{i=2}^{n} |e_i| \leq \gamma_n(2\mathbf{u}) |x^T||y|. \tag{38}$$

and

$$\sum_{i=2}^{n} |q_i| \leq \gamma_{n+1}(2\mathbf{u}) |x^T||y|. \tag{39}$$

For later use, we evaluate an upper bound on the following expression

$$\begin{aligned} &|s_1 + \sum_{i=2}^{n} (q_i + r_i) - s_n| \\ &= \left| s_1 + \sum_{i=2}^{n} (q_i + r_i) - \mathrm{fl}_*\left(s_1 + \sum_{i=2}^{n} (q_i + r_i)\right) \right|. \end{aligned}$$

From Proposition 4.1, it follows that

$$|s_1 + \sum_{i=2}^{n} (q_i + r_i) - s_n| \leq \gamma_{n-1}(2\mathbf{u}) \left(|s_1| + \sum_{i=2}^{n} |\mathrm{fl}_*\left(q_i + r_i\right)| \right). \tag{40}$$

Furthermore, because a directed rounding mode is used, we have

$$\sum_{i=2}^{n} |\mathrm{fl}_*\left(q_i + r_i\right)| \leq (1 + 2\mathbf{u}) \sum_{i=2}^{n} |q_i + r_i|.$$

Therefore from Equation 40, we deduce that

$$|s_1 + \sum_{i=2}^{n} (q_i + r_i) - s_n| \leq (1 + 2\mathbf{u})\gamma_{n-1}(2\mathbf{u}) \left(|s_1| + \sum_{i=2}^{n} |q_i + r_i| \right),$$

and, so

$$|s_1 + \sum_{i=2}^{n} (q_i + r_i) - s_n| \leq \gamma_n(2\mathbf{u}) \left(|s_1| + \sum_{i=2}^{n} |q_i + r_i| \right).$$

From Equations 16 and 39, it follows that

$$|s_1 + \sum_{i=2}^{n} (q_i + r_i) - s_n| \leq \gamma_n(2\mathbf{u}) \left(2\mathbf{u} + 8\mathbf{u}^2 + \gamma_{n+1}(2\mathbf{u})\right) |x^T||y|. \tag{41}$$

We deduce from Equation 15 that

$$|\left(x^T y - p_n\right) - s_n| = \left| s_1 + \sum_{i=2}^{n} (e_i + t_i) - s_n \right|.$$

As a consequence, it yields

$$\begin{aligned} &|x^T y - p_n - s_n| \\ &= \left| s_1 + \sum_{i=2}^{n} (q_i + r_i) - s_n + \sum_{i=2}^{n} (e_i - q_i) + \sum_{i=2}^{n} (t_i - r_i) \right|, \end{aligned}$$

and

$$\begin{aligned} &|x^T y - p_n - s_n| \\ &\leq \left| s_1 + \sum_{i=2}^{n} (q_i + r_i) - s_n \right| + \sum_{i=2}^{n} |e_i - q_i| + \sum_{i=2}^{n} |t_i - r_i|. \end{aligned}$$

Therefore, we deduce that

$$|x^T y - p_n - s_n| \leq [\gamma_n(2\mathbf{u})(4\mathbf{u} + \gamma_{n+1}(2\mathbf{u}) + 8\mathbf{u}^2) + 4\mathbf{u}^2] |x^T||y|. \tag{42}$$

Because $n > 2$ and $\mathbf{u}$ is small,

$$|x^T y - p_n - s_n| \leq 2\gamma_{n+1}(2\mathbf{u})^2 |x^T||y|. \tag{43}$$

Because Algorithm 16 is executed with a directed rounding mode, it follows that

$$|\mathtt{res} - x^T y| = |(1 + \varepsilon)(p_n + s_n) - x^T y| \quad \text{with} \quad |\varepsilon| \leq 2\mathbf{u}.$$

Therefore, we have

$$|\mathtt{res} - x^T y| = |\varepsilon x^T y + (1 + \varepsilon)(p_n + s_n - x^T y)|,$$

and

$$|\mathtt{res} - x^T y| \leq 2\mathbf{u}|x^T y| + (1 + 2\mathbf{u})|p_n + s_n - x^T y|.$$

Then from Equation 43, it follows that

$$|\mathtt{res} - x^T y| \leq 2\mathbf{u}|x^T y| + 2(1 + 2\mathbf{u})\gamma_{n+1}(2\mathbf{u})^2 |x^T||y|.$$

```
setround(-1)
Dinf = CompDot(x,y)
setround(1)
Dsup = CompDot(x,y)
```

Algorithm 18: Computation of interval bounds `Dinf` and `Dsup` with the compensated dot product algorithm `CompDot`

□

Algorithm 18 shows how to compute with MATLAB the CompDotFMA algorithm with rounding to $-\infty$, and then with rounding to $+\infty$.

In Proposition 5.8, we show that Algorithm 17 provides an inclosure of the dot product.

Proposition 5.8. *Let floating-point numbers $x_i, y_i \in \mathbb{F}, 1 \leq i \leq n$, be given. If* `Dinf` *and* `Dsup` *are computed using Algorithm 18, then we have*

$$\texttt{Dinf} \leq x^T y \leq \texttt{Dsup}.$$

Proof. Let e_i be the error on the floating-point addition of π_{i-1} and h_i $(i = 2, ..., n)$. We know that $x^T y = p_n + s_1 + \sum_{i=2}^{n}(e_i + t_i)$ where $\pi_i + e_i = \pi_{i-1} + h_i$ and $h_i + t_i = x_i \times y_i$ (see Proposition 4.5 in [15]).

- Let us assume computations are performed with rounding toward $+\infty$.
  From Proposition 3.6, it follows that $e_i \leq q_i$. From Propisition 3.9 it follows that $t_i \leq r_i$. As a consequence, we have $x^T y \leq p_n + s_1 + \sum_{i=2}^{n}(q_i + r_i)$. As we use rounding toward $+\infty$, we have $s_1 + \sum_{i=2}^{n}(q_i + r_i) \leq s_n$ so $x^T y \leq p_n + s_n$. As we always use rounding toward $+\infty$, we also have $x^T y \leq \texttt{res} := \texttt{Dsup}$.
- Let us assume computations are performed with rounding toward $-\infty$.
  From Proposition 3.6, it follows that $q_i \leq e_i$. rom Propisition 3.9 it follows that $r_i \leq t_i$. As a consequence, we have $p_n + s_1 + \sum_{i=2}^{n}(q_i + r_i) \leq x^T y$. As we use rounding toward $-\infty$, we have $s_n \leq s_1 + \sum_{i=2}^{n}(q_i + r_i)$ so $p_n + s_n \leq x^T y$. As we always use rounding toward $-\infty$, we also have $\texttt{Dinf} := \texttt{res} \leq x^T y$.

□

6 Accurate Horner scheme

In this section we recall how to obtain inclusions of a polynomial evaluation using the classic Horner scheme. Then we show that tighter inclusions can be computed using a compensated Horner scheme executed with directed rounding. In this section, we assume that no underflow occurs.

6.1 Classic Horner scheme

The classical method for evaluating a polynomial

$$p(x) = \sum_{i=0}^{n} a_i x^i$$

function $\texttt{res} = \texttt{Horner}(p, x)$

1: $s_n \leftarrow a_n$
2: **for** $i = n - 1$ downto 0 **do**
3: $\quad s_i \leftarrow s_{i+1} \times x + a_i$
4: **end for**
5: $\texttt{res} \leftarrow s_0$

Algorithm 19: Polynomial evaluation with Horner's scheme

is the Horner scheme which consists of Algorithm 19.

Whatever the rounding mode, a forward error bound on the result of Algorithm 19 is (see [17, p. 95]):

$$|p(x) - \texttt{res}| \leq \gamma_{2n}(2\mathbf{u}) \sum_{i=0}^{n} |a_i||x|^i = \gamma_{2n}(2\mathbf{u})\widetilde{p}(|x|)$$

where $\widetilde{p}(x) = \sum_{i=0}^{n} |a_i| x^i$. The relative error on the result can be expressed in terms of the condition number of the polynomial evaluation defined by

$$\text{cond}(p, x) = \frac{\sum_{i=0}^{n} |a_i||x|^i}{|p(x)|} = \frac{\widetilde{p}(|x|)}{|p(x)|}. \tag{44}$$

Thus we have

$$\frac{|p(x) - \texttt{res}|}{|p(x)|} \leq \gamma_{2n}(2\mathbf{u})\,\text{cond}(p, x).$$

If an `FMA` instruction is available, then the statement $s_i \leftarrow s_{i+1} \times x + a_i$ in Algorithm 19 can be rewritten as $s_i \leftarrow \texttt{FMA}(s_{i+1}, x, a_i)$ which slightly improves the error bound (see [17]).

Algorithm 20 presents how to compute an inclosure of $p(x)$ if $x \geq 0$. If $x \leq 0$, $\texttt{Horner}(\bar{p}, -x)$ is computed with $\bar{p}(x) = \sum_{i=0}^{n} a_i (-1)^i x^i$.

```
setround(-1)
Einf = Horner(p,x)
setround(1)
Esup = Horner(p,x)
```

Algorithm 20: Computation of interval bounds `Einf` and `Esup` with the classic Horner scheme for $x \geq 0$

As for dot product and summation with directed rounding ([28]), the following enclosure holds.

Proposition 6.1. *Consider a polynomial p of degree n with floating-point coefficients, and a floating-point value x. If* `Einf` *and* `Esup` *are computed using Algorithm 20, then*

$$\texttt{Einf} \leq p(x) \leq \texttt{Esup}.$$

6.2 Compensated Horner scheme with directed rounding

A compensated Horner scheme [10], [31] is recalled as Algorithm 21 (`CompHorner`).

The error generated by Algorithm 21 with directed rounding is given in [15] and is recalled in Proposition 6.2.

function res $=$ CompHorner(p, x)

1: $s_n \leftarrow a_n$
2: $r_n \leftarrow 0$
3: **for** $i = n-1$ down to 0 **do**
4: $\quad [p_i, \pi_i] \leftarrow$ TwoProdFMA(s_{i+1}, x)
5: $\quad [s_i, \sigma_i] \leftarrow$ FastTwoSum(p_i, a_i)
6: $\quad r_i \leftarrow r_{i+1} \times x + (\pi_i + \sigma_i)$
7: **end for**
8: res $\leftarrow s_0 + r_0$

Algorithm 21: Polynomial evaluation with a compensated Horner scheme

Proposition 6.2. *Consider a polynomial p of degree n with floating-point coefficients, and a floating-point value x. With directed rounding, the forward error in the compensated Horner algorithm is such that*

$$|\texttt{CompHorner}(p, x) - p(x)| \leq 2\mathbf{u}|p(x)| + 2\gamma_{2n+1}(2\mathbf{u})^2 \widetilde{p}(|x|).$$

Combining this error bound with the condition number (44) for the polynomial evaluation gives

$$\frac{|\texttt{CompHorner}(p, x) - p(x)|}{|p(x)|} \leq 2\mathbf{u} + 2\gamma_{2n+1}(2\mathbf{u})^2 \operatorname{cond}(p, x).$$

Because $\gamma_{2n+1}(2\mathbf{u}) \approx 4n\mathbf{u}$, the bound for the relative error of the computed result is essentially $(n\mathbf{u})^2$ times the condition number of the polynomial evaluation, plus the unavoidable term $2\mathbf{u}$ for rounding the result to the working precision. The computed result is almost as accurate as if it was computed by the classic Horner algorithm with twice the working precision, and then rounded to the working precision.

Algorithm 22 presents how to compute an inclosure of $p(x)$ if $x \geq 0$. Like with Algorithm 20, if $x \leq 0$, CompHorner$(\bar{p}, -x)$ is computed with $\bar{p}(x) = \sum_{i=0}^{n} a_i (-1)^i x^i$.

```
setround(-1)
Einf = CompHorner(p,x)
setround(1)
Esup = CompHorner(p,x)
```

Algorithm 22: Computation of interval bounds Einf and Esup with the compensated Horner scheme CompHorner for $x \geq 0$

In Proposition 6.3 we show that Algorithm 22 provides an inclosure of $p(x)$. The results of Algorithm 22 are almost as accurate as if the classical Horner scheme was computed in twice the working precision.

Proposition 6.3. *Consider a polynomial p of degree n with floating-point coefficients, and a floating-point value x. If* Einf *and* Esup *are computed using Algorithm 22, then*

$$\texttt{Einf} \leq p(x) \leq \texttt{Esup}.$$

Proof. We analyze the impact of a directed rounding mode on Algorithm 21 (CompHorner).

Let τ_i be the rounding error in the floating-point addition of p_i and a_i (τ_i is not necessarily a floating-point number):

$$s_i + \tau_i = p_i + a_i.$$

It follows that $s_{i+1} \times x = p_i + \pi_i$ and $p_i + a_i = s_i + \tau_i$ with $|\tau_i - \sigma_i| \leq 2\mathbf{u}\tau_i$. As a consequence, we have

$$s_i = s_{i+1} \times x - \pi_i - \tau_i \quad \text{for } i = 0, \ldots, n-1.$$

By induction, we deduce that

$$p(x) = s_0 + p_\pi(x) + p_\tau(x),$$

with

$$s_0 = \mathrm{fl}_*(p(x)), \quad p_\pi(x) = \sum_{i=0}^{n-1} \pi_i x^i, \quad \text{and} \quad p_\tau(x) = \sum_{i=0}^{n-1} \tau_i x^i.$$

- Let us assume computations are performed with rounding toward $+\infty$.
  From Proposition 3.2, it follows that $\tau_i \leq \sigma_i$. As a consequence, we have
  $$p(x) \leq s_0 + \sum_{i=0}^{n-1} \pi_i x^i + \sum_{i=0}^{n-1} \sigma_i x^i.$$
  As we use rounding toward $+\infty$, we have $p(x) \leq s_0 + r_0 =$ res $:=$ Esup.
- Let us assume computations are performed with rounding toward $-\infty$.
  From Proposition 3.2, it follows that $\sigma_i \leq \tau_i$. As a consequence, we have
  $$s_0 + \sum_{i=0}^{n-1} \pi_i x^i + \sum_{i=0}^{n-1} \sigma_i x^i \leq p(x).$$
  As we use rounding toward $-\infty$, we have Einf $:=$ res $= s_0 + r_0 \leq p(x)$.

□

A similar result can be obtained with CompHorner2 (Algorithm 23) by using TwoProduct instead of TwoProdFMA and TwoSum instead of FastTwoSum.

function res $=$ CompHorner2(p, x)

1: $s_n \leftarrow a_n$
2: $r_n \leftarrow 0$
3: **for** $i = n-1$ down to 0 **do**
4: $\quad [p_i, \pi_i] \leftarrow$ TwoProduct(s_{i+1}, x)
5: $\quad [s_i, \sigma_i] \leftarrow$ TwoSum(p_i, a_i)
6: $\quad r_i \leftarrow r_{i+1} \times x + (\pi_i + \sigma_i)$
7: **end for**
8: res $\leftarrow s_0 + r_0$

Algorithm 23: Polynomial evaluation with a compensated Horner scheme without FMA

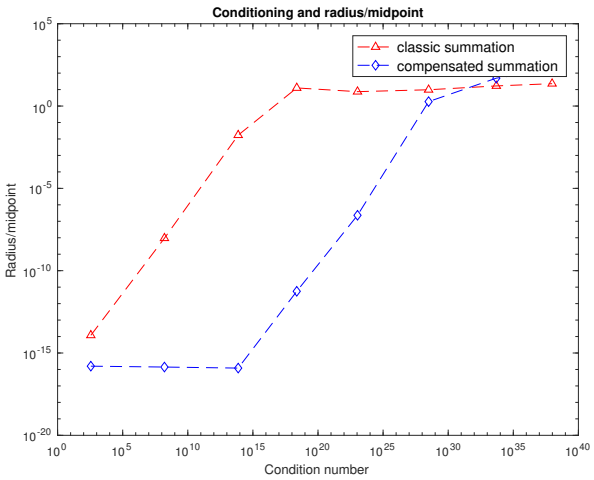

Fig. 1: Classic and compensated summation computed with interval arithmetic

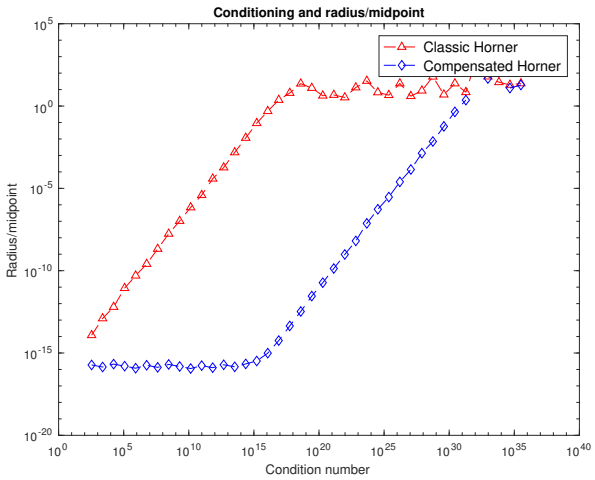

Fig. 3: Classic and compensated Horner scheme computed with interval arithmetic

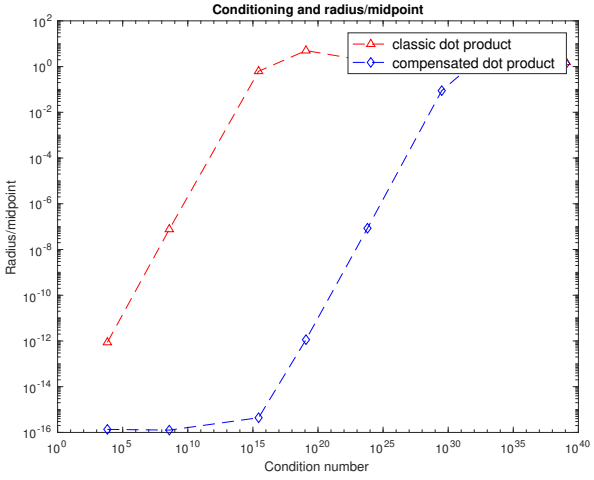

Fig. 2: Classic and compensated dot product computed with interval arithmetic

7 Numerical results

In this section, we present results computed with interval arithmetic using the classic and the compensated algorithms for summation, dot product and Horner scheme. With the compensated algorithms, the interval bounds have been computed as described in the previous sections. The numerical experiments have been carried out on a laptop with an Intel Core i5 processor at 2.9 GHz with 16 Gb of RAM. We used MATLAB R2016b with INTLAB v10 [16]. The computation has been performed with the *binary64* (double precision) format of the IEEE 754-2008 standard [1]. Figures 1 to 3 display the radius over the midpoint of interval results obtained for various condition numbers.

From Figures 1 to 3, with the classic algorithms, if the condition number increases, the radius over the midpoint of the computed interval also increases, which means that the numerical quality of the result decreases. If the condition number reaches about 10^{15}, the computed result has no more correct digits. With the compensated algorithms, if the condition number remains less than about 10^{15}, the numerical quality of the computed result is very satisfactory. If the condition number increases from about 10^{15} to 10^{30}, the numerical quality of the result decreases. If the condition number reaches about 10^{30}, the result has no more correct digits. As expected, the interval results obtained with the compensated algorithms are almost as accurate as if they were computed in twice the working precision. Tight interval inclusions have been computed thanks to compensated algorithms.

8 Conclusion and perspectives

In this paper we have shown that tight inclusions can be computed for summation, dot product, and polynomial evaluation thanks to compensated algorithms executed with directed rounding. The results obtained are almost as accurate as if they were computed using twice the working precision. The approach chosen in this paper consists in executing the compensated algorithms entirely with rounding toward $-\infty$, and then with rounding toward $+\infty$. An advantage of this approach lies in the fact that the original compensated algorithms can be used, possibly from a library usually executed with rounding to nearest.

Another approach would consist in computing the results once with rounding to nearest and the corrections with rounding toward $-\infty$, and then with rounding toward $+\infty$. This approach would be more memory consuming than the approach presented in this paper. However it would perform better in terms of execution time. It would be interesting to compare the two approaches.

K-fold compensated algorithms enable one to compute summation and dot product as in K-fold precision [9]. Priest's EFT [8] for the addition and `TwoProdFMA` both compute the generated rounding error whatever the rounding mode. The impact of a directed rounding mode on K-fold compensated algorithms based on these EFTs has been shown in [15]. Another perspective would consist in studying K-fold compensated algorithms to see if they can

provide for summation and dot product narrow inclusions, as in K-fold precision.

As a future work, we could also determine of it would be possible to obtain tight inclusions using other compensated algorithms, such as compensated exponentiation [32], compensated Newton's scheme [33], [34], the compensated evaluation of elementary symmetric functions [35], or the compensated algorithm for solving triangular systems [36].

Acknowledgments

This work was partly supported by the project FastRelax ANR-14-CE25-0018-01.

References

[1] IEEE Computer Society, *IEEE Standard for Floating-Point Arithmetic*. IEEE Standard 754-2008, Aug. 2008.

[2] D. H. Bailey, Y. Hida, X. S. Li, and B. Thompson, "Arprec: An arbitrary precision computation package," Tech. Rep., 2002.

[3] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann, "Mpfr: A multiple-precision binary floating-point library with correct rounding," *ACM Trans. Math. Softw.*, vol. 33, no. 2, Jun. 2007. [Online]. Available: http://doi.acm.org/10.1145/1236463.1236468

[4] Y. Hida, X. Li, and D. Bailey, "Library for double-double and quad-double arithmetic," NERSC Division, Lawrence Berkeley National Laboratory, USA, Tech. Rep., 2008, http://www.davidhbailey.com/dhbpapers/qd.pdf.

[5] D. M. Priest, "Algorithms for arbitrary precision floating point arithmetic," in *Proceedings of the 10th IEEE Symposium on Computer Arithmetic (Arith-10)*, P. Kornerup and D. W. Matula, Eds. IEEE Computer Society Press, Los Alamitos, CA, Jun. 1991, pp. 132–144.

[6] J. R. Shewchuk, "Adaptive precision floating-point arithmetic and fast robust geometric predicates," *Discrete & Computational Geometry*, vol. 18, pp. 305–368, 1997.

[7] M. Joldes, J.-M. Muller, V. Popescu, and W. Tucker, "CAMPARY: Cuda Multiple Precision Arithmetic Library and Applications," in *5th International Congress on Mathematical Software (ICMS)*, Berlin, Germany, Jul. 2016. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01312858

[8] D. Priest, "On properties of floating point arithmetics: Numerical stability and the cost of accurate computations," Ph.D. dissertation, Mathematics Department, University of California, Berkeley, CA, USA, Nov. 1992, ftp://ftp.icsi.berkeley.edu/pub/theory/priest-thesis.ps.Z.

[9] T. Ogita, S. M. Rump, and S. Oishi, "Accurate sum and dot product," *SIAM Journal on Scientific Computing*, vol. 26, no. 6, pp. 1955–1988, 2005.

[10] S. Graillat, P. Langlois, and N. Louvet, "Algorithms for accurate, validated and fast polynomial evaluation," *Japan J. Indust. Appl. Math.*, vol. 2-3, no. 26, pp. 191–214, 2009, special issue on State of the Art in Self-Validating Numerical Computations.

[11] G. Alefeld and J. Herzberger, *Introduction to interval analysis*. Academic Press, 1983.

[12] U. Kulisch, *Advanced Arithmetic for the Digital Computer*. Springer-Verlag, Wien, 2002.

[13] S. Graillat, F. Jézéquel, and R. Picot, "Numerical validation of compensated summation algorithms with stochastic arithmetic," *Electronic Notes in Theoretical Computer Science*, vol. 317, pp. 55–69, 2015.

[14] S. Boldo, S. Graillat, and J.-M. Muller, "On the robustness of the 2Sum and Fast2Sum algorithms," *ACM Trans. Math. Softw.*, vol. 44, no. 1, pp. 4:1–4:14, Jul. 2017.

[15] S. Graillat, F. Jézéquel, and R. Picot, "Numerical validation of compensated algorithms with stochastic arithmetic," *Applied Mathematics and Computation*, vol. 329, pp. 339 – 363, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0096300318300985

[16] S. Rump, "INTLAB - INTerval LABoratory," in *Developments in Reliable Computing*, T. Csendes, Ed. Kluwer Academic Publishers, Dordrecht, 1999, pp. 77–104, http://www.ti3.tu-harburg.de/rump/intlab.

[17] N. Higham, *Accuracy and stability of numerical algorithms*, 2nd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 2002.

[18] C.-P. Jeannerod and S. M. Rump, "Improved error bounds for inner products in floating-point arithmetic," *SIAM Journal on Matrix Analysis and Applications*, vol. 34, no. 2, pp. 338–344, 2013. [Online]. Available: http://dx.doi.org/10.1137/120894488

[19] S. M. Rump, "Error estimation of floating-point summation and dot product," *BIT. Numerical Mathematics*, vol. 52, no. 1, pp. 201–220, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10543-011-0342-4

[20] S. M. Rump, F. Bünger, and C.-P. Jeannerod, "Improved error bounds for floating-point products and Horner's scheme," *BIT. Numerical Mathematics*, vol. 56, no. 1, pp. 293–307, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10543-015-0555-z

[21] T. Dekker, "A floating-point technique for extending the available precision," *Numerische Mathematik*, vol. 18, no. 3, pp. 224–242, 1971. [Online]. Available: http://dx.doi.org/10.1007/BF01397083

[22] D. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd ed. Boston, MA, USA: Addison-Wesley, 1997.

[23] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of Floating-Point Arithmetic*. Birkhäuser, Boston, 2010.

[24] J. Hauser, "Handling floating-point exceptions in numeric programs," *ACM Trans. Program. Lang. Syst.*, vol. 18, no. 2, pp. 139–174, 1996.

[25] J. Demmel and H. D. Nguyen, "Fast reproducible floating-point summation," in *21st IEEE Symposium on Computer Arithmetic, Austin, TX, USA, April 7-10*, 2013, pp. 163–172.

[26] P. Sterbenz, *Floating-point computation*, ser. Prentice-Hall series in automatic computation. Prentice-Hall, 1973. [Online]. Available: http://books.google.fr/books?id=MKpQAAAAMAAJ

[27] S. M. Rump, T. Ogita, and S. Oishi, "Accurate floating-point summation. I. Faithful rounding," *SIAM J. Sci. Comput.*, vol. 31, no. 1, pp. 189–224, 2008.

[28] S. M. Rump, "Verification methods: rigorous results using floating-point arithmetic," *Acta Numer.*, vol. 19, pp. 287–449, 2010.

[29] A. Neumaier, "Rundungsfehleranalyse einiger Verfahren zur Summation endlicher Summen," *ZAMM (Zeitschrift für Angewandte Mathematik und Mechanik)*, vol. 54, pp. 39–51, 1974.

[30] M. Pichat, "Correction d'une somme en arithmétique à virgule flottante," *Numerische Mathematik*, vol. 19, pp. 400–406, 1972.

[31] S. Graillat, N. Louvet, and P. Langlois, "Compensated Horner scheme," Équipe de recherche DALI, Laboratoire LP2A, Université de Perpignan Via Domitia, France, 52 avenue Paul Alduy, 66860 Perpignan cedex, France, Research Report 04, July 2005.

[32] S. Graillat, "Accurate floating point product and exponentiation," *IEEE Transactions on Computers*, vol. 58, no. 7, pp. 994–1000, 2009.

[33] ——, "Accurate simple zeros of polynomials in floating point arithmetic," *Computers & Mathematics with Applications*, vol. 56, no. 4, pp. 1114–1120, 2008.

[34] H. Jiang, S. Graillat, C. Hu, S. Lia, X. Liao, L. Cheng, and F. Su, "Accurate evaluation of the k-th derivative of a polynomial," *Journal of Computational and Applied Mathematics*, vol. 191, pp. 28–47, 2013.

[35] H. Jiang, S. Graillat, and R. Barrio, "Accurate and fast evaluation of elementary symmetric functions," in *Proceedings of the 21st IEEE Symposium on Computer Arithmetic, Austin, TX, USA, April 7-10*, 2013, pp. 183–190.

[36] N. Louvet, "Algorithmes compensés en arithmétique flottante : précision, validation, performances," PhD thesis, Université de Perpignan Via Domitia, nov 2007.