Applied Mathematics and Computation 329 (2018) 339–363

Contents lists available at ScienceDirect

Applied Mathematics and Computation

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

Numerical validation of compensated algorithms with stochastic arithmetic

Stef Graillat[a,*], Fabienne Jézéquel[a,b], Romain Picot[a,c]

[a] Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, LIP6, Paris F-75005, France
[b] Université Panthéon-Assas, 12 place du Panthéon, Paris, F-75005, France
[c] EDF R&D, 7 boulevard Gaspard Monge, Palaiseau, F-91120, France

ARTICLE INFO

Keywords:
CADNA
Compensated algorithms
Discrete stochastic arithmetic
Error-free transformations
Floating-point arithmetic
Numerical validation
Rounding errors

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 validate the result of compensated algorithms using stochastic arithmetic. We study compensated algorithms for summation, dot product and polynomial evaluation. We prove that the use of the random rounding mode inherent to stochastic arithmetic does not change much the accuracy of compensated methods. 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.

1. Introduction

Computing power rapidly increases and Exascale computing (10^{18} floating-point operations per second) should be reached in a few years. Such a computing power also means a large number of rounding errors. 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 fundamental to be able to give some information about the numerical quality of the computed results. By numerical quality, we mean here the number of significant digits of the computed result that are not affected by rounding errors.

A well-known solution to assert the numerical quality is to use the numerical library called CADNA[1] that implements Discrete Stochastic Arithmetic (DSA) and makes it possible to provide a confidence interval of the computed result [1–3]. DSA requires several executions of the user program with a random rounding mode that consists in rounding any result to plus or minus infinity with the same probability.

If the accuracy of the computed result is not sufficient, it is necessary to increase the precision of the computation. For simple enough calculations, a possible technique is the use of *compensated algorithms* (see [4]). 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. We now assume a floating-point arithmetic adhering to the IEEE 754-2008

* Corresponding author.
E-mail addresses: stef.graillat@lip6.fr (S. Graillat), fabienne.jezequel@lip6.fr (F. Jézéquel), romain.picot@lip6.fr (R. Picot).
[1] URL address: http://cadna.lip6.fr

https://doi.org/10.1016/j.amc.2018.02.004
0096-3003/

standard [5]. In that case, when using rounding to nearest, the rounding error of an addition is a floating-point number that can be computed exactly via an EFT. But EFTs are no longer valid when used with directed rounding (rounding to plus or minus infinity). Indeed, if we use directed rounding, the error of a floating-point addition is not necessarily a floating-point number. However, directed rounding is required in DSA. As a consequence, it is not clear whether we can use DSA to validate some numerical codes that heavily rely on the use of error-free transformations.

In this article, we show that compensated algorithms enable one to increase the results accuracy even with directed rounding. Whatever the faithful rounding mode chosen, compensated summation, dot product, and Horner scheme algorithms provide a result almost as accurate as if it was computed with twice the working precision. Concerning compensated summation, although part of this work has been done in [6], for completeness some previously obtained results are recalled in Section 3. We also show in this article that compensated algorithms *as in K*-fold precision for summation and dot product [7] provide satisfactory results even with directed rounding: the accuracy obtained is almost as if the working precision was multiplied by *K*. Therefore compensated algorithms can improve accuracy even if they are executed with DSA. Furthermore DSA enables one to estimate the numerical quality of results of compensated algorithms. This satisfactory behavior of compensated algorithms with DSA is confirmed by the numerical experiments described in this article.

This outline of this article is as follows. In Section 2 we give some definitions and notations used in the sequel. In the next sections, we show the impact of a directed rounding mode on compensated algorithms. Sections 3–7 are successively devoted to the error analysis with directed rounding of compensated summation, compensated dot product, compensated Horner scheme, summation *as in K*-fold precision, and dot product *as in K*-fold precision. Finally, numerical experiments carried out using the CADNA library are presented in Section 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 [5] and we suppose that no overflow occurs. The error bounds for the compensated summation that are presented in Sections 3 and 6 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 [8]

$\exists\, \varepsilon_1 \in \mathbb{R},\ \varepsilon_2 \in \mathbb{R}$ such that

$$\mathrm{fl}_*(a \circ b) = (a \circ b)(1+\varepsilon_1) = (a \circ b)/(1+\varepsilon_2) \text{ for } \circ = \{+,-\} \text{ and } |\varepsilon_v| \leq 2\mathbf{u}. \tag{2.1}$$

As a consequence,

$$|a \circ b - \mathrm{fl}_*(a \circ b)| \leq 2\mathbf{u}|a \circ b| \text{ and } |a \circ b - \mathrm{fl}_*(a \circ b)| \leq 2\mathbf{u}|\mathrm{fl}_*(a \circ b)| \text{ for } \circ = \{+,-\}. \tag{2.2}$$

We use standard notations for error estimations. The quantities γ_n are defined as usual [8] 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 $n\mathbf{u} < 1$.

To keep track of the $(1+\varepsilon)$ factors in our error analysis, we use the relative error counters introduced by Stewart [9]. For a positive integer n, $\langle n \rangle$ denotes the following product

$$\langle n \rangle(\mathbf{u}) = \prod_{i=1}^{n} (1+\varepsilon_i)^{\rho_i} \quad \text{with} \quad \rho_i = \pm 1 \quad \text{and} \quad |\varepsilon_i| \leq \mathbf{u} \quad (i = 1, \ldots, n).$$

The relative error counters satisfy $\langle j \rangle(\mathbf{u})\langle k \rangle(\mathbf{u}) = \langle j \rangle(\mathbf{u})/\langle k \rangle(\mathbf{u}) = \langle j+k \rangle(\mathbf{u})$. When $\langle n \rangle$ denotes any error counter, then there exists a quantity θ_n such that

$$\langle n \rangle(\mathbf{u}) = 1 + \theta_n(\mathbf{u}) \quad \text{and} \quad |\theta_n(\mathbf{u})| \leq \gamma_n(\mathbf{u}).$$

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,

$$n\mathbf{u} \leq \gamma_n(\mathbf{u}), \quad \gamma_n(\mathbf{u}) \leq \gamma_{n+1}(\mathbf{u}), \quad (1+\mathbf{u})\gamma_n(\mathbf{u}) \leq \gamma_{n+1}(\mathbf{u}), \quad 2n\mathbf{u}(1+\gamma_{2n-2}(\mathbf{u})) \leq \gamma_{2n}(\mathbf{u}).$$

Remark 2. Recently, it has been shown that classic Wilkinson-type error bounds for summation, dot product and polynomial evaluation [10–12] can be slightly improved by replacing the factor $\gamma_n(\mathbf{u})$ by $n\mathbf{u}$ with no condition on n (for summation and dot product). In the sequel, it is likely that all the error bounds could also be improved by replacing all the $\gamma_n(\mathbf{u})$ by $n\mathbf{u}$. However, it is clear that $\gamma_n(\mathbf{u})$ is very close to $n\mathbf{u}$. Moreover, the proofs for improving the bounds would be more complicated and tricky, and would not be useful for the 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.

In the next sections, we aim at analysing the effects of the random rounding mode required by DSA on compensated algorithms *a priori* intended to be used with rounding to nearest. Therefore we will present the impact of a directed rounding mode on the accuracy of results provided by compensated algorithms.

3. Accurate summation

This section presents the accuracy obtained with the classic summation algorithm and with various compensated summation algorithms, using either rounding to nearest or directed rounding.

3.1. Classic summation

The classic algorithm for summation is the recursive Algorithm 1.

Algorithm 1: Summation of n floating-point numbers $p = \{p_i\}$.

function $\texttt{res} = \texttt{Sum}(p)$

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: $\texttt{res} \leftarrow s_n$

The error generated by Algorithm 1 is recalled in Proposition 3.1.

Proposition 3.1 [8]. *Let us suppose Algorithm 1 is applied to floating-point numbers $p_i \in \mathbb{F}$, $1 \le i \le n$. Let $s := \Sigma p_i$ and $S := \Sigma |p_i|$. With rounding to nearest, if $n\mathbf{u} < 1$, then*

$$|\texttt{res} - s| \le \gamma_{n-1}(\mathbf{u})S. \tag{3.3}$$

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

$$|\texttt{res} - s| \le \gamma_{n-1}(2\mathbf{u})S. \tag{3.4}$$

In Corollary 3.2 Eqs. (3.3) and (3.4) are rewritten in terms of the condition number on Σp_i:

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

Corollary 3.2. *With rounding to nearest, if $n\mathbf{u} < 1$, the result* res *of Algorithm 1 satisfies*

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

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

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

Because $\gamma_{n-1}(\mathbf{u}) \approx (n-1)\mathbf{u}$, the bound for the relative error is essentially $n\mathbf{u}$ times the condition number. This accuracy is sometimes not sufficient in practice. Indeed, when the condition number is large (greater than $1/\mathbf{u}$) then the recursive algorithm does not even return one correct digit. Algorithms to evaluate more accurately the sum of floating-point numbers are presented in the sequel of this section.

3.2. Compensated summation with rounding to nearest

Error-free transformations exist for the sum of two floating-point numbers with rounding to nearest: `TwoSum` [13] which requires 6 floating-point operations and `FastTwoSum` [14], given as Algorithm 2, which requires a test and 3 floating-point operations. These algorithms compute both the floating-point sum c of two numbers a and b and the associated rounding error d such that $c + d = a + b$. Another algorithm, proposed by Priest in [15, p. 14–15] and given later as Algorithm 4 (`PriestTwoSum`), although more costly, computes with any rounding mode an error-free transformation for the sum of two floating-point numbers.

A compensated algorithm to evaluate accurately the sum of n floating-point numbers is presented as Algorithm 3 (`FastCompSum`) [16,17]. This sum is corrected thanks to an error-free transformation used for each individual summation. Although `FastTwoSum` is called in Algorithm 3, with rounding to nearest the same result can be obtained using another error-free transformation (`TwoSum` or `PriestTwoSum`).

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

function $[c, d] = \texttt{FastTwoSum}(a, b)$

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

Algorithm 3: Compensated summation of n floating-point numbers $p = \{p_i\}$ using `FastTwoSum`.

function $\texttt{res} = \texttt{FastCompSum}(p)$

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

The error on the result `res` of Algorithm 3 obtained with rounding to nearest is analysed in [7]. A bound for the absolute error is recalled in Proposition 3.3 and a bound for the relative error in Corollary 3.4.

Proposition 3.3 [7]**.** *Let us suppose Algorithm 3 (`FastCompSum`) is applied, with rounding to nearest, to floating-point numbers $p_i \in \mathbb{F}$, $1 \le i \le n$. Let $s := \Sigma p_i$ and $S := \Sigma |p_i|$. If $n\mathbf{u} < 1$, then, also in the presence of underflow,*

$$|\texttt{res} - s| \le \mathbf{u}|s| + \gamma_{n-1}^2(\mathbf{u})S.$$

Corollary 3.4 [7]**.** *With rounding to nearest, if $n\mathbf{u} < 1$, then, also in the presence of underflow, the result `res` of Algorithm 3 (`FastCompSum`) satisfies*

$$\frac{|\texttt{res} - s|}{|s|} \le \mathbf{u} + \gamma_{n-1}^2(\mathbf{u})\text{cond}\left(\sum \mathrm{p_i}\right).$$

From Corollary 3.4, because $\gamma_{n-1}(\mathbf{u}) \approx (n-1)\mathbf{u}$, the bound for the relative error on the result is essentially $(n\mathbf{u})^2$ times the condition number plus the rounding $\mathbf{u}$ due to the working precision. The second term on the right hand side reflects that the computation is carried out almost as with twice the working precision ($\mathbf{u}^2$). The first term represents the rounding back into the working precision.

3.3. Compensated summation with directed rounding

We recall here the impact of a directed rounding mode on Algorithm 3 (`FastCompSum`). With directed rounding, Algorithm 2 (`FastTwoSum`) is not an error-free transformation. A bound on the difference between the floating-point number d computed by Algorithm 2 and the error e due to the floating-point addition is recalled in Proposition 3.5.

Proposition 3.5 [6]**.** *Let c and d be the floating-point addition of a and b and the correction both computed by Algorithm 2 (`FastTwoSum`) using directed rounding. Let e be the error on c: $a + b = c + e$. Then*

$$|e - d| \le 2\mathbf{u}|e|.$$

Eq. (3.5) in Lemma 3.6, established in [6], is recalled for later use.

Lemma 3.6 [6]**.** *Let us suppose Algorithm3 (`FastCompSum`) is applied, with directed rounding, to floating-point numbers $p_i \in \mathbb{F}$, $1 \le i \le n$. For $i = 2, \ldots, n$, let e_i be the error on the floating-point addition of π_{i-1} and p_i: $\pi_i + e_i = \pi_{i-1} + p_i$. If $n\mathbf{u} < \frac{1}{2}$, then*

$$\sum_{i=2}^{n} |e_i| \le \gamma_{n-1}(2\mathbf{u}) \sum_{i=1}^{n} |p_i|. \tag{3.5}$$

A bound for the absolute error on the result of Algorithm 3 (`FastCompSum`) obtained with directed rounding is recalled in Proposition 3.7.

Proposition 3.7 [6]. *Let us suppose Algorithm 3 (*`FastCompSum`*) is applied, with directed rounding, to floating-point numbers* $p_i \in \mathbb{F}$, $1 \le i \le n$. *Let* $s := \Sigma p_i$ *and* $S := \Sigma |p_i|$. *If* $n\mathbf{u} < \frac{1}{2}$, *then, also in the presence of underflow,*

$$|\mathtt{res} - s| \le 2\mathbf{u}|s| + 2(1+2\mathbf{u})\gamma_n^2(2\mathbf{u})S.$$

From Proposition 3.7, a bound for the relative error on the result of Algorithm 3 (`FastCompSum`) obtained with directed rounding is deduced in Corollary 3.8.

Corollary 3.8. *With directed rounding, if* $n\mathbf{u} < \frac{1}{2}$, *then, also in the presence of underflow, the result* `res` *of Algorithm 3 (*`FastCompSum`*) satisfies*

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

From Corollary 3.8, 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 inevitable rounding $2\mathbf{u}$ due to the working precision.

The impact of a directed rounding mode on the compensated summation based on the `PriestTwoSum` algorithm is analysed here. The error bounds obtained will be used in Section 6. The `PriestTwoSum` algorithm [15, p. 14–15] is recalled as Algorithm 4.

Algorithm 4: Error-free transformation for the sum of two floating-point numbers with any rounding mode.

function $[c, d]$ = `PriestTwoSum`(a, b)

1: **if** $|b| > |a|$ **then**
2: exchange a and b
3: **end if**
4: $c \leftarrow a + b$
5: $e \leftarrow c - a$
6: $g \leftarrow c - e$
7: $h \leftarrow g - a$
8: $f \leftarrow b - h$
9: $d \leftarrow f - e$
10: **if** $d + e \neq f$ **then**
11: $c \leftarrow a$
12: $d \leftarrow b$
13: **end if**

A compensated summation algorithm based on `PriestTwoSum` is given as Algorithm 5 .

Algorithm 5: Compensated summation of n floating-point numbers $p = \{p_i\}$ using `PriestTwoSum`.

function `res` = `PriestCompSum`(p)

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

Lemma 3.9 is given for later use in Section 6.

Lemma 3.9. *Let us suppose Algorithm 5 (*`PriestCompSum`*) is applied, with directed rounding, to floating-point numbers* $p_i \in \mathbb{F}$, $1 \le i \le n$. *Let* $s := \Sigma p_i$ *and* $S := \Sigma |p_i|$. *If* $n\mathbf{u} < \frac{1}{2}$, *then*

$$\sum_{i=2}^{n} |q_i| + |\pi_n| \le |s| + 2\gamma_{n-1}(2\mathbf{u})S. \tag{3.6}$$

Proof. As

$$\sum_{i=2}^{n} |q_i| + |\pi_n| = \sum_{i=2}^{n} |q_i| + |s - \sum_{i=2}^{n} q_i|,$$

we have

$$\sum_{i=2}^{n} |q_i| + |\pi_n| \le |s| + 2\sum_{i=2}^{n} |q_i|. \tag{3.7}$$

From Lemma 3.6, we deduce that

$$\sum_{i=2}^{n} |q_i| \leq \gamma_{n-1}(2\mathbf{u}) \sum_{i=1}^{n} |p_i| = \gamma_{n-1}(2\mathbf{u})S. \tag{3.8}$$

Finally Eq. (3.6) is obtained from Eqs. (3.7) and (3.8). □

A bound for the absolute error on the result of Algorithm 5 (`PriestCompSum`) obtained with directed rounding is given in Proposition 3.10.

Proposition 3.10. *Let us suppose Algorithm* `PriestCompSum` *is applied, with directed rounding, to floating-point numbers* $p_i \in \mathbb{F}$, $1 \leq i \leq n$. *Let* $s := \Sigma p_i$ *and* $S := \Sigma |p_i|$. *If* $n\mathbf{u} < \frac{1}{2}$, *then, also in the presence of underflow,*

$$|\mathtt{res} - s| \leq 2\mathbf{u}|s| + \gamma_{n-1}^2(2\mathbf{u})S. \tag{3.9}$$

The proof is similar to the one given in [7] for compensated summation with rounding to nearest.

Proof. Because $\sigma_n = \mathrm{fl}_*(\sum_{i=2}^{n} q_i)$, we have

$$|\sigma_n - \sum_{i=2}^{n} q_i| \leq \gamma_{n-2}(2\mathbf{u}) \sum_{i=2}^{n} |q_i|.$$

Therefore, from Eq. (3.8), we deduce that

$$|\sigma_n - \sum_{i=2}^{n} q_i| \leq \gamma_{n-2}(2\mathbf{u})\gamma_{n-1}(2\mathbf{u})S.$$

As Algorithm 5 is executed with directed rounding, it yields

$$\mathtt{res} = \mathrm{fl}_*(\pi_n + \sigma_n) = (1+\varepsilon)(\pi_n + \sigma_n) \quad \text{with} \quad |\varepsilon| \leq 2\mathbf{u},$$

$$|\mathtt{res} - s| = |\mathrm{fl}_*(\pi_n + \sigma_n) - s|,$$

$$|\mathtt{res} - s| = |(1+\varepsilon)(\pi_n + \sigma_n - s) + \varepsilon s|,$$

$$|\mathtt{res} - s| = |(1+\varepsilon)\left(\pi_n + \sum_{i=2}^{n} q_i - s\right) + (1+\varepsilon)\left(\sigma_n - \sum_{i=2}^{n} q_i\right) + \varepsilon s|.$$

Since

$$s = \sum_{i=1}^{n} p_i = \pi_n + \sum_{i=2}^{n} q_i, \tag{3.10}$$

then

$$|\mathtt{res} - s| \leq (1+2\mathbf{u})|\sigma_n - \sum_{i=2}^{n} q_i| + 2\mathbf{u}|s|,$$

$$|\mathtt{res} - s| \leq (1+2\mathbf{u})\gamma_{n-2}(2\mathbf{u})\gamma_{n-1}(2\mathbf{u})S + 2\mathbf{u}|s|. \tag{3.11}$$

We have

$$(1+2\mathbf{u})\gamma_{n-1}(2\mathbf{u}) < \gamma_n(2\mathbf{u}). \tag{3.12}$$

Indeed, it is clear that

$$\gamma_n(2\mathbf{u}) - (1+2\mathbf{u})\gamma_{n-1}(2\mathbf{u}) = 2\mathbf{u}\left(1 + \frac{2n\mathbf{u}}{(1-2(n-1)\mathbf{u})(1-2n\mathbf{u})}\right),$$

and then

$$\gamma_n(2\mathbf{u}) - (1+2\mathbf{u})\gamma_{n-1}(2\mathbf{u}) > 0.$$

Finally Eq. (3.9) can be deduced from Eqs. (3.11) and (3.12). □

From Proposition 3.10, a bound for the relative error on the result of Algorithm 5 (`PriestCompSum`) obtained with directed rounding is deduced in Corollary 3.11.

Corollary 3.11. *With directed rounding, if* $n\mathbf{u} < \frac{1}{2}$, *then, also in the presence of underflow, the result* `res` *of Algorithm 5 (*`PriestCompSum`*) satisfies*

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

Like with Algorithm 3 (`FastCompSum`), we deduce from Corollary 3.11 that the relative error bound on the result of Algorithm 5 (`PriestCompSum`) 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.

4. Accurate dot product

In this section, we present the accuracy obtained with the classic dot product algorithm. We also present an algorithm that enables one to compute a dot product almost as in twice the working precision with rounding to nearest [7]. We recall the error on its result computed with rounding to nearest. Then we analyse the impact of a directed rounding mode on this algorithm. In this section, we assume that no underflow occurs.

4.1. Classic dot product

The classic algorithm for computing a dot product is Algorithm 6 .

Algorithm 6: Classic dot product of $x = \{x_i\}$ and $y = \{y_i\}$, $1 \leq i \leq n$.

function `res` $=$ `Dot`(x, y)

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

The following proposition sums up the properties of this algorithm.

Proposition 4.1. *Let floating point numbers* $x_i, y_i \in \mathbb{F}, 1 \leq i \leq n$, *be given and denote by* `res` $\in \mathbb{F}$ *the result computed by Algorithm 6 (*`Dot`*). With rounding to nearest, if* $n\mathbf{u} < 1$, *we have*

$$|\mathtt{res} - x^T y| \leq \gamma_n(\mathbf{u})|x^T||y|, \tag{4.13}$$

and with directed rounding, if $n\mathbf{u} < \frac{1}{2}$, *we have*

$$|\mathtt{res} - x^T y| \leq \gamma_n(2\mathbf{u})|x^T||y|. \tag{4.14}$$

Proof. The proof can be found in Higham [8, p. 63]. □

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

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

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

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

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

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

4.2. Compensated dot product with rounding to nearest

A compensated dot product algorithm is presented in [7]. This algorithm, intended to be used with rounding to nearest, is based on two error-free transformations: `TwoSum` [13] and `TwoProd` [14] that compute respectively the sum and the product of two floating-point numbers. The `TwoProd` algorithm requires 17 floating-point operations. However another error-free transformation, `TwoProdFMA` presented as Algorithm 7, exists for the product and costs only 2 floating-point operations ([4, p. 152]).

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

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

Algorithm 7: Error-free transformation for the product of two floating-point numbers using an FMA.

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

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

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.

With any rounding mode, the TwoProdFMA algorithm 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 CompDot algorithm, presented as Algorithm 8, is a compensated dot product algorithm based on FastTwoSum

Algorithm 8: Compensated dot product of $x = \{x_i\}$ and $y = \{y_i\}$, $1 \leq i \leq n$.

function res=CompDot(x, y)

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

(Algorithm 2) and TwoProdFMA. As a remark, with rounding to nearest, the result of the CompDot algorithm is identical if other error-free transformations are used for the sum or the product.

The error on the result res of Algorithm 8 obtained with rounding to nearest is analysed in [7]. A bound for the absolute error is recalled in Proposition 4.3.

Proposition 4.3 [7]**.** *Let floating-point numbers $x_i, y_i \in \mathbb{F}$, $1 \leq i \leq n$, be given and denote by $\mathtt{res} \in \mathbb{F}$ the result computed by Algorithm 8 (CompDot) with rounding to nearest. If $n\mathbf{u} < 1$, then,*

$$|\mathtt{res} - x^T y| \leq \mathbf{u}|x^T y| + \gamma_n^2(\mathbf{u})|x^T||y|. \tag{4.15}$$

In Corollary 4.4, Eq. (4.15) is rewritten in terms of the condition number for the dot product.

Corollary 4.4 [7]**.** *With rounding to nearest, if $n\mathbf{u} < 1$, then, the result res of Algorithm 8 (CompDot) satisfies*

$$\frac{|\mathtt{res} - x^T y|}{|x^T y|} \leq \mathbf{u} + \frac{1}{2}\gamma_n^2(\mathbf{u})\mathrm{cond}\left(\mathrm{x^T y}\right).$$

As a consequence, we conclude that the result is almost as accurate as if it was computed in twice the working precision and then rounded to the current working precision. This is the same phenomenon as for the compensated summation algorithm.

4.3. Compensated dot product with directed rounding

We present here the impact of a directed rounding mode on Algorithm 8 (CompDot). For the error analysis we rewrite this algorithm into the following equivalent one.

A bound for the absolute error on the result res of Algorithm 8 is given in Proposition 4.5.

Proposition 4.5. *Let floating-point numbers $x_i, y_i \in \mathbb{F}$, $1 \leq i \leq n$, be given and denote by $\mathtt{res} \in \mathbb{F}$ the result computed by Algorithm 8 (CompDot) with directed rounding. If $(n+1)\mathbf{u} < \frac{1}{2}$, then,*

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

Proof. Thanks to the TwoProdFMA algorithm, we have

$$p_1 + s_1 = x_1 y_1, \tag{4.16}$$

and for $i \geq 2$,

$$h_i + r_i = x_i y_i. \tag{4.17}$$

From Proposition 3.5, it follows that

$$p_i + e_i = p_{i-1} + h_i \quad \text{with} \quad |q_i - e_i| \leq 2\mathbf{u}|e_i|. \tag{4.18}$$

Therefore from Eq. (4.17), we deduce that

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

Then from Eq. (4.16), we derive

$$s_1 + \sum_{i=2}^{n}(e_i + r_i) = (x_1 y_1 - p_1) + \left(\sum_{i=2}^{n} x_i y_i + p_1 - p_n\right) = x^T y - p_n. \tag{4.19}$$

Because the `TwoProdFMA` algorithm is executed with a directed rounding mode, for $i \geq 2$, then

$$|r_i| \leq 2\mathbf{u}|x_i y_i|.$$

Therefore, we have

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

and

$$|s_1| + \sum_{i=2}^{n}|r_i| \leq 2\mathbf{u}|x^T||y|. \tag{4.20}$$

From Lemma 3.6, we deduce

$$\sum_{i=2}^{n}|e_i| \leq \gamma_{n-1}(2\mathbf{u})\left(|p_1| + \sum_{i=2}^{n}|h_i|\right).$$

As a consequence, we have

$$\sum_{i=2}^{n}|e_i| \leq \gamma_{n-1}(2\mathbf{u})\left(\sum_{i=1}^{n}|\mathrm{fl}_*(x_i y_i)|\right),$$

and

$$\sum_{i=2}^{n}|e_i| \leq (1 + 2\mathbf{u})\gamma_{n-1}(2\mathbf{u})|x^T||y|. \tag{4.21}$$

From Eqs. 3.12 and 4.21, we derive

$$\sum_{i=2}^{n}|e_i| \leq \gamma_n(2\mathbf{u})|x^T||y|. \tag{4.22}$$

From Eq. (4.18), we conclude that

$$\sum_{i=2}^{n}|q_i - e_i| \leq 2\mathbf{u}\sum_{i=2}^{n}|e_i|. \tag{4.23}$$

Therefore from Eq. (4.22), we deduce

$$\sum_{i=2}^{n}|q_i - e_i| \leq 2\mathbf{u}\gamma_n(2\mathbf{u})|x^T||y|. \tag{4.24}$$

We have

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

Therefore, Eq. (4.23) yields

$$\sum_{i=2}^{n}|q_i| \leq (1 + 2\mathbf{u})\sum_{i=2}^{n}|e_i|.$$

From Eqs. (3.12) and (4.22), it yields

$$\sum_{i=2}^{n} |q_i| \leq \gamma_{n+1}(2\mathbf{u})|x^T||y|. \tag{4.25}$$

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

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

From Proposition 3.1, it follows that

$$|s_1 + \sum_{i=2}^{n}(q_i + r_i) - s_n| \leq \gamma_{n-1}(2\mathbf{u}) \left(|s_1| + \sum_{i=2}^{n} |\mathrm{fl}_*(q_i + r_i)| \right). \tag{4.26}$$

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

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

Therefore from Eq. (4.26), 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, from Eq. (3.12),

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

From Eqs. (4.20) and (4.25), it follows that

$$|s_1 + \sum_{i=2}^{n}(q_i + r_i) - s_n| \leq \gamma_{n}(2\mathbf{u})(2\mathbf{u} + \gamma_{n+1}(2\mathbf{u}))|x^T||y|. \tag{4.27}$$

We deduce from Eq. (4.19) that

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

As a consequence, it yields

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

and

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

Therefore from Eqs. (4.24) and (4.27), we deduce that

$$|x^T y - p_n - s_n| \leq \gamma_{n}(2\mathbf{u})(4\mathbf{u} + \gamma_{n+1}(2\mathbf{u}))|x^T||y|. \tag{4.28}$$

Let us show that $\gamma_{n+1}(2\mathbf{u}) \geq 4\mathbf{u}$. It is easy to show that

$$\gamma_{n+1}(2\mathbf{u}) - 4\mathbf{u} = \frac{2(n+1)\mathbf{u}}{1 - 2(n+1)\mathbf{u}} - 4\mathbf{u},$$

and

$$\gamma_{n+1}(2\mathbf{u}) - 4\mathbf{u} = \frac{2\mathbf{u}}{1 - 2(n+1)\mathbf{u}}(n - 1 + 4(n+1)\mathbf{u}).$$

Because $(n+1)\mathbf{u} < \frac{1}{2}$, it follows that $\gamma_{n+1}(2\mathbf{u}) - 4\mathbf{u} \geq 0$.

Therefore from Eq. (4.28), we can deduce that

$$|x^T y - p_n - s_n| \leq 2\gamma_{n}(2\mathbf{u})\gamma_{n+1}(2\mathbf{u})|x^T||y|. \tag{4.29}$$

Algorithm 9: Equivalent formulation of Algorithm 8.

function `res=CompDot`(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: $\quad [p_i, q_i] \leftarrow$ `FastTwoSum`(p_{i-1}, h_i)
5: $\quad s_i \leftarrow s_{i-1} + (q_i + r_i)$
6: **end for**
7: `res` $\leftarrow p_n + s_n$

Because Algorithm 9 is executed with a directed rounding mode, it follows that

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

Therefore, we have

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

and

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

Then from Eq. (4.29), it follows that

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

Finally from Eq. (3.12), we conclude that

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

□

From Proposition 4.5, a bound for the relative error on the result of Algorithm 8 (`CompDot`) obtained with directed rounding is deduced in Corollary 4.6.

Corollary 4.6. *With directed rounding, if* $(n+1)\mathbf{u} < \frac{1}{2}$, *then, the result* `res` *of Algorithm 8* (`CompDot`) *satisfies*

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

From Corollary 4.6, the relative error bound on the result of Algorithm 8 (`CompDot`) 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. Like with rounding to nearest, the result obtained with directed rounding is almost as accurate as if it was computed in twice the working precision and then rounded to the current working precision.

5. Accurate Horner scheme

In this section, we present the accuracy obtained with the classic Horner scheme for polynomial evaluation. We recall a compensated Horner scheme algorithm and the error on its result computed with rounding to nearest. Then we analyse the impact of a directed rounding mode on this algorithm. In this section, we assume that no underflow occurs.

5.1. Classic Horner scheme

The classical method for evaluating a polynomial

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

is the Horner scheme which consists of Algorithm 10.

Algorithm 10: Polynomial evaluation with Horner's scheme.

function `res` $=$ `Horner`(p, x)

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

Whatever the rounding mode, a forward error bound on the result of Algorithm 10 is (see [8, 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 $\widetilde{p}(x) = \sum_{i=0}^{n} |a_i|x^i$. It is very interesting to express and interpret this result in terms of the condition number of the polynomial evaluation defined by

$$\mathrm{cond(p, x)} = \frac{\sum_{i=0}^{n} |a_i||x|^i}{|p(x)|} = \frac{\widetilde{p}(|x|)}{|p(x)|}. \tag{5.30}$$

Thus we have

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

If an FMA instruction is available, then the statement $s_i \leftarrow s_{i+1} \cdot x + a_i$ in Algorithm 10 can be rewritten as $s_i \leftarrow \texttt{FMA}(s_{i+1}, x, a_i)$ which slightly improves the error bound. Using an FMA this way, the computed result now satisfies

$$|p(x) - \texttt{res}| \leq \gamma_n(2\mathbf{u})\widetilde{p}(|x|).$$

5.2. A compensated Horner scheme with rounding to nearest

We now want to accurately compute a polynomial at a given point. The Horner scheme algorithm can be modified to compute the rounding error at each elementary operation using error-free transformations. We present as Algorithm 11 a

Algorithm 11: Polynomial evaluation with a compensated Horner scheme.

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

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

compensated algorithm for the Horner scheme. One can find a more detailed description of the compensated Horner scheme algorithm in [18,19]. Algorithm 11 is based on the TwoProdFMA and FastTwoSum algorithms. However, with rounding to nearest, its result is identical if other error-free transformations, such as TwoProd or TwoSum, are used.

If we denote by p_π and p_σ the two following polynomials

$$p_\pi(x) = \sum_{i=0}^{n-1} \pi_i x^i, \qquad p_\sigma(x) = \sum_{i=0}^{n-1} \sigma_i x^i,$$

then one can show, thanks to error-free transformations, that

$$p(x) = s_0 + p_\pi(x) + p_\sigma(x).$$

If one looks closely at the previous algorithm, it is then clear that $s_0 = \texttt{Horner}(p, x)$. As a consequence, we can derive a new error-free transformation for the polynomial evaluation

$$p(x) = \texttt{Horner}(p, x) + p_\pi(x) + p_\sigma(x).$$

The compensated Horner scheme first computes $p_\pi(x) + p_\sigma(x)$ which corresponds to the rounding errors and then adds the value obtained to the result of the classic Horner scheme $\texttt{Horner}(p, x)$. We will show that the result computed by Algorithm 11 admits a significantly better error bound than that computed by the classical Horner scheme. We argue that Algorithm 11 provides a result almost as accurate as if it was computed using twice the working precision. This is summed up in the following theorem.

Theorem 5.1. *Consider a polynomial p of degree n with floating-point coefficients, and a floating-point value x. With rounding to nearest, the forward error in the compensated Horner algorithm is such that*

$$|\texttt{CompHorner}(p, x) - p(x)| \leq \mathbf{u}|p(x)| + \gamma_{2n}^2(\mathbf{u})\widetilde{p}(x). \tag{5.31}$$

It is interesting to interpret the previous theorem in terms of the condition number of the evaluation of p at x. Combining the error bound (5.31) with the condition number (5.30) for polynomial evaluation gives

$$\frac{|\texttt{CompHorner}(p,x)-p(x)|}{|p(x)|} \leq \mathbf{u}+\gamma_{2n}^2(\mathbf{u})\mathrm{cond}(\mathrm{p},\mathrm{x}). \tag{5.32}$$

In other words, the bound for the relative error of the computed result is essentially $\gamma_{2n}^2(\mathbf{u})$ times the condition number of the polynomial evaluation, plus the unavoidable term $\mathbf{u}$ for rounding the result to the working precision. In particular, if $\mathrm{cond}(\mathrm{p},\mathrm{x}) < \gamma_{2n}^{-1}(\mathbf{u})$, then the relative accuracy of the result is bounded by a constant of the order of $\mathbf{u}$. This means that the compensated Horner algorithm computes an evaluation accurate to the last few bits as long as the condition number is smaller than $\gamma_{2n}^{-1}(\mathbf{u}) \approx (2n\mathbf{u})^{-1}$. Besides that, (5.32) tells us that 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.

5.3. A compensated Horner scheme with directed rounding

We now present the impact of a directed rounding mode on Algorithm 11 (`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 $s_{i+1}\cdot x = p_i + \pi_i$ 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

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

By induction, we deduce that

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

with

$$s_0 = \mathrm{fl}_*(p(x)), \quad p_\pi(x) = \sum_{i=0}^{n-1}\pi_i x^i, \quad \text{and} \quad p_\tau(x) = \sum_{i=0}^{n-1}\tau_i x^i. \tag{5.33}$$

We recall that

$$p_\sigma(x) = \sum_{i=0}^{n-1}\sigma_i x^i. \tag{5.34}$$

In the sequel, we will denote $e(x) := p_\pi(x) + p_\sigma(x)$. In this case, we have $p(x) = \mathrm{fl}(\mathrm{p}(\mathrm{x})) + \mathrm{e}(\mathrm{x}) + (\mathrm{p}_\tau - \mathrm{p}_\sigma)(\mathrm{x})$ and $\mathtt{res} = \mathrm{fl}(\mathrm{p}(\mathrm{x}) + \mathrm{e}(\mathrm{x}))$.

Lemma 5.2. *Let $p(x) = \sum_{i=0}^n a_i x^i$ a polynomial with $a_i \in \mathbb{F}$, $0 \leq i \leq n$ and $x \in \mathbb{F}$. Let p_π and p_σ be defined by (5.33) and (5.34). Then, we have*

$$\widetilde{p_\pi}(|x|) + \widetilde{p_\sigma}(|x|) \leq \gamma_{2n+1}(2\mathbf{u})\widetilde{p}(|x|),$$

with $\widetilde{p}(x) = \sum_{i=0}^n |a_i| x^i$.

Proof. Using (2.1), we have, for $i = 1,\ldots,n$,

$$|p_{n-i}| = |\mathrm{fl}_*(s_{n-i+1}\cdot x)| \leq (1+2\mathbf{u})|s_{n-i+1}||x|$$

and

$$|s_{n-i}| = |\mathrm{fl}_*(p_{n-i} + a_{n-i})| \leq (1+2\mathbf{u})(|p_{n-i}| + |a_{n-i}|).$$

Let us show by induction on $i = 1,\ldots,n$ that

$$|p_{n-i}| \leq (1+\gamma_{2i-1}(2\mathbf{u}))\sum_{j=1}^{i}|a_{n-i+j}||x^j| \tag{5.35}$$

and

$$|s_{n-i}| \leq (1+\gamma_{2i}(2\mathbf{u}))\sum_{j=0}^{i}|a_{n-i+j}||x^j|. \tag{5.36}$$

For $i = 1$, as $s_n = a_n$, we have

$$|p_{n-1}| \leq (1+2\mathbf{u})|a_n||x| \leq (1+\gamma_1(2\mathbf{u}))|a_n||x|$$

and so (5.35) is true. In the same way, as

$$|s_{n-1}| \leq (1+2\mathbf{u})((1+\gamma_1(2\mathbf{u}))|a_n||x| + |a_{n-1}|) \leq (1+\gamma_2(2\mathbf{u}))(|a_n||x| + |a_{n-1}|)$$

then (5.36) is also true. Let us assume that (5.35) and (5.36) are true for an integer i, $1 \le i < n$. Then we have

$$|p_{n-(i+1)}| \le (1+2\mathbf{u})|s_{n-i}||x|.$$

By hypothesis, we deduce that

$$\begin{aligned}|p_{n-(i+1)}| &\le (1+2\mathbf{u})(1+\gamma_{2i}(2\mathbf{u}))\sum_{j=0}^{i}|a_{n-i+j}||x^{j+1}| \\ &\le (1+\gamma_{2(i+1)-1}(2\mathbf{u}))\sum_{j=1}^{i+1}|a_{n-(i+1)+j}||x^{j}|.\end{aligned}$$

Hence, it follows that

$$\begin{aligned}|s_{n-(i+1)}| &\le (1+2\mathbf{u})(|p_{n-(i+1)}|+|a_{n-(i+1)}|) \\ &\le (1+2\mathbf{u})(1+\gamma_{2(i+1)-1}(2\mathbf{u}))\left[\sum_{j=1}^{i+1}|a_{n-(i+1)+j}||x^{j}|+|a_{n-(i+1)}|\right] \\ &\le (1+\gamma_{2(i+1)}(2\mathbf{u}))\sum_{j=0}^{i+1}|a_{n-(i+1)+j}||x^{j}|.\end{aligned}$$

Relations (5.35) and (5.36) are then true by induction. As a consequence, for $i = 1, \dots, n$, we have

$$|p_{n-i}||x^{n-i}| \le (1+\gamma_{2i-1}(2\mathbf{u}))\widetilde{p}(x)$$

and

$$|s_{n-i}||x^{n-i}| \le (1+\gamma_{2i}(2\mathbf{u}))\widetilde{p}(x).$$

Following (2.2), we have $|\pi_i| \le 2\mathbf{u}|p_i|$, $|\tau_i| \le 2\mathbf{u}|s_i|$ and $|\sigma_i| \le (1+2\mathbf{u})|\tau_i|$ for $i = 0, \dots, n-1$. Hence,

$$(\widetilde{p_\pi}+\widetilde{p_\sigma})(|x|) = \sum_{i=0}^{n-1}(|\pi_i|+|\sigma_i|)|x^i| \le 2\mathbf{u}(1+2\mathbf{u})\sum_{i=1}^{n}(|p_{n-i}|+|s_{n-i}|)|x^{n-i}|,$$

and so

$$(\widetilde{p_\pi}+\widetilde{p_\sigma})(|x|) \le 2\mathbf{u}(1+2\mathbf{u})\sum_{i=1}^{n}(2+\gamma_{2i-1}(2\mathbf{u})+\gamma_{2i}(2\mathbf{u}))\widetilde{p}(|x|) \le 4n\mathbf{u}(1+2\mathbf{u})(1+\gamma_{2n}(2\mathbf{u}))\widetilde{p}(|x|).$$

As $4n\mathbf{u}(1+\gamma_{2n}(2\mathbf{u})) = \gamma_{2n}(2\mathbf{u})$, we deduce that $(\widetilde{p_\pi}+\widetilde{p_\sigma})(|x|) \le \gamma_{2n+1}(2\mathbf{u})\widetilde{p}(|x|)$. □

Lemma 5.3. *Let $p(x) = \sum_{i=0}^{n} a_i x^i$ be a polynomial with $a_i \in \mathbb{F}$, $0 \le i \le n$, $q(x) = \sum_{i=0}^{n} b_i x^i$ a polynomial with $b_i \in \mathbb{F}$, $0 \le i \le n$ and $x \in \mathbb{F}$. Then the floating-point evaluation of $r(x) = p(x) + q(x)$ via the following algorithm*

1: $r_n \leftarrow \mathrm{fl}*(a_n + b_n)$
2: **for** $i = n-1$ down to 0 **do**
3: $\quad r_i \leftarrow \mathrm{fl}*(r_{i+1}\cdot x + (a_i + b_i))$
4: **end for**
5: `res` $\leftarrow r_0$

satisfies

$$|\mathtt{res} - r(x)| \le \gamma_{2n+1}(2\mathbf{u})\tilde{r}(|x|).$$

Proof. Considering the previous algorithm, we have $r_n = \mathrm{fl}_*(a_n + b_n) = (a_n + b_n)\langle 1\rangle(2\mathbf{u})$ and for $i = n-1$ down to 0,

$$r_i = \mathrm{fl}_*(r_{i+1}\cdot x + (a_i + b_i)) = r_{i+1}x\langle 2\rangle(2\mathbf{u}) + (a_i + b_i)\langle 2\rangle(2\mathbf{u}).$$

As a consequence, we can show by induction that

$$r_0 = (a_n + b_n)x^n\langle 2n+1\rangle(2\mathbf{u}) + \sum_{i=0}^{n-1}(a_i + b_i)x^i\langle 2(i+1)\rangle(2\mathbf{u}).$$

Moreover with quantities $\theta_{2n+1}(2\mathbf{u}), \theta_{2n}(2\mathbf{u}), \dots, \theta_1(2\mathbf{u})$, satisfying $|\theta_i(2\mathbf{u})| \le \gamma_i(2\mathbf{u})$, we have

$$r_0 = (a_n + b_n)x^n(1+\theta_{2n+1})(2\mathbf{u}) + \sum_{i=0}^{n-1}(a_i + b_i)x^i(1+\theta_{2(i+1)}(2\mathbf{u})).$$

As $r_0 = \mathrm{fl}_*(p(x) + q(x))$, we finally get

$$\left| \mathtt{res} - \sum_{i=0}^{n} (a_i + b_i)x^i \right| \leq \gamma_{2n+1}(2\mathbf{u}) \sum_{i=0}^{n} |a_i + b_i||x^i| \leq \gamma_{2n+1}(2\mathbf{u})(\widetilde{p} + \widetilde{q})(|x|),$$

which concludes the proof. □

Theorem 5.4. *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*

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

Proof. By considering Algorithm 11, we have $p(x) = s_0 + e(x) + (p_\tau - p_\sigma)(x)$. We can deduce that

$$\begin{aligned} |\mathtt{res} - p(x)| &= |(1+\varepsilon)(s_0 + \mathrm{fl}_*(e(x))) - p(x)| \\ &= |(1+\varepsilon)(s_0 + \mathrm{fl}_*(e(x)) - p(x) + (p_\tau - p_\sigma)(x)) + \varepsilon p(x) + (1+\varepsilon)(p_\sigma - p_\tau)(x)| \\ &= |(1+\varepsilon)(s_0 + e(x) + (p_\tau - p_\sigma)(x) - p(x)) + (1+\varepsilon)(\mathrm{fl}_*(e(x)) - e(x)) + \\ &\quad \varepsilon p(x) + (1+\varepsilon)(p_\tau - p_\sigma)(x)| \\ &\leq 2\mathbf{u}|p(x)| + (1+2\mathbf{u})|\mathrm{fl}_*(e(x)) - e(x)| + (1+2\mathbf{u})|(p_\tau - p_\sigma)(x)|. \end{aligned}$$

By applying Lemma 5.3, we obtain

$$|\mathrm{fl}_*(e(x)) - e(x)| \leq \gamma_{2n-1}(2\mathbf{u})\widetilde{e}(|x|) \leq \gamma_{2n-1}(2\mathbf{u})(\widetilde{p_\pi}(|x|) + \widetilde{p_\sigma}(|x|)).$$

Moreover from Lemma 5.2, we get

$$\widetilde{p_\pi}(|x|) + \widetilde{p_\sigma}(|x|) \leq \gamma_{2n+1}(2\mathbf{u})\widetilde{p}(|x|).$$

Since $|\tau_i - \sigma_i| \leq 2\mathbf{u}\tau_i$, we have

$$|(p_\tau - p_\sigma)(x)| \leq 2\mathbf{u} \sum_{i=0}^{n-1} |\tau_i||x|^i \leq 2\mathbf{u}\widetilde{p_\tau}(|x|).$$

Moreover, as $|\tau_i| \leq 2\mathbf{u}|s_i|$, we have $\widetilde{p_\tau}(|x|) \leq 2n\mathbf{u}\gamma_{2n}(2\mathbf{u})\widetilde{p}(|x|)$. As a consequence, we deduce

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

As $(1+2\mathbf{u})\gamma_{2n-1}(2\mathbf{u}) \leq \gamma_{2n}(2\mathbf{u})$ and $2n\mathbf{u} \leq \gamma_{2n+1}(2\mathbf{u})$, we obtain

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

which concludes the proof. □

6. Summation as in *K*-fold precision

According to Section 3, Algorithms 3 (`FastCompSum`) and 5 (`PriestCompSum`) compute the sum of n floating-point numbers almost as in twice the working precision, even with directed rounding. In this section, we present the SumK algorithm [7] that computes this sum *as in K*-fold precision. We recall the error bound on its result obtained with rounding to nearest. Then we analyse the impact of directed rounding on this algorithm.

6.1. Summation as in K-fold precision with rounding to nearest

The SumK algorithm, introduced in [7] and presented as Algorithm 12, enables the summation of a vector of floating-

Algorithm 12: Summation of n floating-point numbers $p = \{p_i\}$ in K-fold working precision, $K \geq 3$.

function $\mathtt{res} = \mathtt{SumK}(p)$

1: **for** $k = 1$ to $K - 1$ **do**
2: **for** $i = 2$ to n **do**
3: $[p_i, p_{i-1}] \leftarrow \mathtt{PriestTwoSum}(p_i, p_{i-1})$
4: **end for**
5: **end for**
6: $\mathtt{res} \leftarrow \sum_{i=1}^{n} p_i$

point numbers as in K-fold precision. In [7] the SumK algorithm is based on the `TwoSum` algorithm [13]. However the same result `res` can be obtained using rounding to nearest with any error-free transformation that computes the sum of two floating-point numbers. The SumK algorithm is presented here with Algorithm 4 (`PriestTwoSum`) that, although

more costly, is an error-free transformation with any rounding mode. As a remark if $K = 2$, Algorithm 12 is identical with Algorithm 5 (`PriestCompSum`).

The error on the result `res` of Algorithm 12 obtained with rounding to nearest is analysed in [7]. A bound for the absolute error is recalled in Proposition 6.1 and a bound for the relative error in Corollary 6.2.

Proposition 6.1 [7]. *Let floating-point numbers $p_i \in \mathbb{F}$, $1 \le i \le n$, be given and assume $4n\mathbf{u} \le 1$. Then, also in the presence of underflow, the result* `res` *of Algorithm 12 (*`SumK`*) obtained with rounding to nearest satisfies for $K \ge 3$*

$$|\mathtt{res} - s| \le \left(\mathbf{u} + 3\gamma_{n-1}^2(\mathbf{u})\right)|s| + \gamma_{2n-2}^K(\mathbf{u})S$$

where $s := \Sigma p_i$ and $S := \Sigma|p_i|$.

Corollary 6.2 [7]. *Assume $4n\mathbf{u} \le 1$. The result* `res` *of Algorithm 12 (*`SumK`*) obtained with rounding to nearest, also in the presence of underflow, satisfies*

$$\frac{|\mathtt{res} - s|}{|s|} \le \mathbf{u} + 3\gamma_{n-1}^2(\mathbf{u}) + \gamma_{2n-2}^K(\mathbf{u})\mathrm{cond}\left(\sum \mathrm{p_i}\right).$$

From Corollary 6.2, because $\gamma_n(\mathbf{u}) \approx n\mathbf{u}$, the bound for the relative error on the result obtained with rounding to nearest is essentially the relative rounding error $\mathbf{u}$ plus a term that reflects that the computation is carried out almost as in K-fold precision ($(\alpha\mathbf{u})^K$ times the condition number for a moderate factor α).

6.2. Summation as in K-fold precision with directed rounding

We analyse here the impact of a directed rounding mode on Algorithm 12 (`SumK`). The steps of the error analysis are similar to those presented in [7] for rounding to nearest. We denote the input vector p by $p^{(0)}$, and the vector after finishing loop k by $p^{(k)}$. We also set $S^{(k)} := \sum_{i=1}^n |p_i^{(k)}|$ for $0 \le k \le K-1$.

Lemma 6.3. *With the above notations, the intermediate results of Algorithm 12 (*`SumK`*) satisfy:*

$$s := \sum_{i=1}^n p_i^{(0)} = \sum_{i=1}^n p_i^{(k)} \quad \text{for} \quad 1 \le k \le K-1, \tag{6.37}$$

$$|\mathtt{res} - s| \le 2\mathbf{u}|s| + \gamma_{n-1}^2(2\mathbf{u})S^{(K-2)}, \tag{6.38}$$

$$S^{(k)} \le 3|s| + \gamma_{2n-2}^k(2\mathbf{u})S^{(0)} \quad \text{provided} \quad 8(n-1)\mathbf{u} \le 1 \quad \text{and} \quad 1 \le k \le K-1. \tag{6.39}$$

Proof. Eq. (6.37) follows by successive applications of Eq. (3.10).

Eq. (6.38) can be deduced using $s = \sum_{i=1}^n p_i^{(K-2)}$ and applying Proposition 3.10.

Let us now prove Eq. (6.39). From Lemma 3.9, we obtain

$$\sum_{i=1}^n |p_i^{(1)}| \le |s| + 2\gamma_{n-1}(2\mathbf{u})S^{(0)}. \tag{6.40}$$

By applying successively Eq. (6.40) and using Eq. (6.37) we obtain

$$S^{(2)} \le |s| + 2\gamma_{n-1}(2\mathbf{u})\left(|s| + 2\gamma_{n-1}(2\mathbf{u})S^{(0)}\right),$$

and

$$S^{(k)} \le |s| + \sum_{i=0}^{\infty}(2\gamma_{n-1}(2\mathbf{u}))^i + (2\gamma_{n-1}(2\mathbf{u}))^k S^{(0)} \quad \text{for } 1 \le k \le K-1.$$

We have

$$\sum_{i=0}^{\infty}(2\gamma_{n-1}(2\mathbf{u}))^i = \frac{1 - 2(n-1)\mathbf{u}}{1 - 6(n-1)\mathbf{u}}.$$

If $8(n-1)\mathbf{u} \le 1$, then we get $1 - 2(n-1)\mathbf{u} \le 3(1 - 6(n-1)\mathbf{u})$ and

$$\frac{1 - 2(n-1)\mathbf{u}}{1 - 6(n-1)\mathbf{u}} \le 3.$$

Therefore, we have

$$S^{(k)} \le 3|s| + (2\gamma_{n-1}(2\mathbf{u}))^k S^{(0)}.$$

Because $2\gamma_m(2\mathbf{u}) \le \gamma_{2m}(2\mathbf{u})$, we can conclude that

$$S^{(k)} \le 3|s| + (\gamma_{2n-2}(2\mathbf{u}))^k S^{(0)}.$$

□

A bound for the absolute error on the result of Algorithm 12 (SumK) obtained with directed rounding is given in Proposition 6.4.

Proposition 6.4. *Let floating-point numbers $p_i \in \mathbb{F}$, $1 \le i \le n$, be given and assume $8n\mathbf{u} \le 1$. Then, also in the presence of underflow, the result* res *of Algorithm 12 (SumK) obtained with directed rounding satisfies for $K \ge 3$*

$$|\mathtt{res} - s| \le \left(2\mathbf{u} + 3\gamma_{n-1}^2(2\mathbf{u})\right)|s| + \gamma_{2n-2}^K(2\mathbf{u})S$$

where $s := \Sigma p_i$ and $S := \Sigma |p_i|$.

Proof. Proposition 6.4 can be proved by inserting Eq. (6.39) of Lemma 6.3 into Eq. (6.38). □

From Proposition 6.4, a bound for the relative error on the result of Algorithm 12 (SumK) obtained with directed rounding is deduced in Corollary 6.5.

Corollary 6.5. *Assume $8n\mathbf{u} \le 1$. The result* res *of Algorithm 12 (SumK) obtained with directed rounding, also in the presence of underflow, satisfies*

$$\frac{|\mathtt{res} - s|}{|s|} \le 2\mathbf{u} + 3\gamma_{n-1}^2(2\mathbf{u}) + \gamma_{2n-2}^K(2\mathbf{u})\mathrm{cond}\left(\sum \mathrm{p_i}\right). \tag{6.41}$$

From Corollary 6.5, because $\gamma_n(2\mathbf{u}) \approx 2n\mathbf{u}$, the bound for the relative error on the result res obtained with directed rounding is essentially the relative rounding error $2\mathbf{u}$ plus $(\alpha\mathbf{u})^K$ times the condition number for a moderate factor α. Like with rounding to nearest, the last term on the right hand side of Eq. (6.41) reflects that the computation is carried out almost as in K-fold precision.

7. Dot product as in *K*-fold precision

According to Section 4, Algorithm 8 (CompDot) computes a dot product almost as in twice the working precision, even with directed rounding. In this section, we present the DotK algorithm [7] that computes a dot product almost as in K-fold precision. We recall the error bound on its result obtained with rounding to nearest. Then we analyse the impact of directed rounding on this algorithm. Like in Section 4, we assume in this section that no underflow occurs.

7.1. Dot product as in K-fold precision with rounding to nearest

The DotK algorithm, introduced in [7] and presented as Algorithm 13, enables one to compute a dot product almost as in K-fold precision. In [7] the DotK algorithm is based on TwoProd [14] and TwoSum [13] that are error-free transformations with rounding to nearest. The DotK algorithm is presented here with TwoProdFMA and PriestTwoSum that are error-free transformations with any rounding mode. As a remark if $K = 2$, Algorithm 13 is identical with Algorithm 8 (CompDot).

Algorithm 13: Dot product algorithm in K-fold working precision, $K \ge 3$.

function res = DotK(x, y, K)
1: $[p, r_1] \leftarrow$ TwoProdFMA(x_1, y_1)
2: **for** $i = 2$ to n **do**
3: $\quad [h, r_i] \leftarrow$ TwoProdFMA(x_i, y_i)
4: $\quad [p, r_{n+i-1}] \leftarrow$ PriestTwoSum(p, h)
5: **end for**
6: $r_{2n} \leftarrow p$
7: res $\leftarrow$ SumK($r, K-1$)

The error on the result res of Algorithm 13 obtained with rounding to nearest is analysed in [7]. A bound for the absolute error is recalled in Proposition 7.1 and a bound for the relative error in Corollary 7.2.

Proposition 7.1 [7]**.** *Let floating-point numbers $x_i, y_i \in \mathbb{F}$, $1 \le i \le n$, be given and assume $8n\mathbf{u} \le 1$. Denote by* res $\in \mathbb{F}$ *the result computed by Algorithm 13 (DotK) with rounding to nearest.*
Then

$$|\mathtt{res} - x^T y| \le \left(\mathbf{u} + 2\gamma_{4n-2}^2(\mathbf{u})\right)|x^T y| + \gamma_{4n-2}^K(\mathbf{u})|x^T||y|.$$

Corollary 7.2 [7]**.** *Assume $8n\mathbf{u} \le 1$. The result* res *of Algorithm 13 (DotK) obtained with rounding to nearest satisfies*

$$\left|\frac{\mathtt{res} - x^T y}{x^T y}\right| \le \mathbf{u} + 2\gamma_{4n-2}^2(\mathbf{u}) + \frac{1}{2}\gamma_{4n-2}^K(\mathbf{u})\mathrm{cond}\left(\mathrm{x^T y}\right).$$

From Corollary 7.2, the bound for the relative error on the result is essentially the relative rounding error $\mathbf{u}$ plus a term that reflects that the computation is carried out almost as in K-fold precision ($\alpha(K)\mathbf{u}^K$ times the condition number for a moderate factor $\alpha(K)$).

7.2. Dot product as in K-fold precision with directed rounding

We present here the impact of a directed rounding mode on Algorithm 13 (DotK). For the analysis we rewrite Algorithm 13 into the following equivalent one (*Algorithm 14*).

Algorithm 14: Equivalent formulation of Algorithm 13.

function $\texttt{res} = \texttt{DotK}(x, y, K)$

1: $[p_1, r_1] \leftarrow \texttt{TwoProdFMA}(x_1, y_1)$
2: **for** $i = 2$ to n **do**
3: $\quad [h_i, r_i] \leftarrow \texttt{TwoProdFMA}(x_i, y_i)$
4: $\quad [p_i, r_{n+i-1}] \leftarrow \texttt{PriestTwoSum}(p_{i-1}, h_i)$
5: **end for**
6: $r_{2n} \leftarrow p_n$
7: $\texttt{res} \leftarrow \texttt{SumK}(r, K-1)$

A bound for the absolute error on the result of Algorithm DotK obtained with directed rounding is given in Proposition 7.3.

Proposition 7.3. *Let floating-point numbers $x_i, y_i \in \mathbb{F}$, $1 \le i \le n$, be given and denote by $\texttt{res} \in \mathbb{F}$ the result computed by Algorithm 13 (DotK) with directed rounding. If $16n\mathbf{u} \le 1$, then*

$$|\texttt{res} - x^T y| \le \left(2\mathbf{u} + 2\gamma^2_{4n-2}(2\mathbf{u})\right)|x^T y| + \gamma^K_{4n-2}(2\mathbf{u})|x^T||y|$$

Proof. Because TwoProdFMA and PriestTwoSum are error-free transformations even with directed rounding, we have

$$s := \sum_{i=1}^{2n} r_i = x^T y. \tag{7.42}$$

Indeed, it is clear that

$$r_1 = x_1 y_1 - p_1,$$

and for $i \ge 2$,

$$\begin{aligned} r_i + r_{n+i-1} &= (x_i y_i - h_i) + (p_{i-1} + h_i - p_i), \\ &= x_i y_i + p_{i-1} - p_i. \end{aligned}$$

Therefore, because $r_{2n} = p_n$, we have

$$\begin{aligned} \sum_{i=1}^{2n-1} r_i &= (x_1 y_1 - p_1) + \left(\sum_{i=2}^{n} x_i y_i + p_1 - r_{2n}\right), \\ &= x^T y - r_{2n}. \end{aligned} \tag{7.43}$$

Therefore we can deduce Eq. (7.42).

Applying Proposition 6.4 requires to estimate $S := \sum_{i=1}^{2n} |r_i|$. Because Algorithm 13 is executed with directed rounding, we have

$$|r_1| \le 2\mathbf{u}|x_1 y_1|, \tag{7.44}$$

and

$$\sum_{i=2}^{n} |r_i| \le 2\mathbf{u} \sum_{i=2}^{n} |x_i y_i|. \tag{7.45}$$

From Lemma 3.6 applied to Algorithm PriestTwoSum, we deduce

$$\sum_{i=2}^{n} |r_{n+i-1}| \le \gamma_{n-1}(2\mathbf{u}) \left(|p_1| + \sum_{i=2}^{n} |h_i|\right),$$

$$= \gamma_{n-1}(2\mathbf{u}) \sum_{i=1}^{n} |fl_*(x_i y_i)|,$$
$$\leq (1+2\mathbf{u})\gamma_{n-1}(2\mathbf{u})|x^T||y|. \tag{7.46}$$

From Eqs. (7.44), (7.45) and (7.46), we obtain

$$\sum_{i=1}^{2n-1} |r_i| \leq 2\mathbf{u}|x^T||y| + (1+2\mathbf{u})\gamma_{n-1}(2\mathbf{u})|x^T||y|,$$
$$\leq \frac{2n\mathbf{u}}{1-2(n-1)\mathbf{u}}|x^T||y|. \tag{7.47}$$

From Eq. (7.43), we deduce that

$$|r_{2n}| = |x^T y - \sum_{i=1}^{2n-1} r_i|,$$
$$\leq |x^T y| + \sum_{i=1}^{2n-1} |r_i|.$$

Therefore, we have

$$\sum_{i=1}^{2n} |r_i| \leq |x^T y| + 2\sum_{i=1}^{2n-1} |r_i|. \tag{7.48}$$

From Eq. (7.47), we obtain

$$2\sum_{i=1}^{2n-1} |r_i| \leq \frac{(2n)(2\mathbf{u})}{1-(n-1)(2\mathbf{u})}|x^T||y|,$$
$$\leq \frac{(2n)(2\mathbf{u})}{1-2n(2\mathbf{u})}|x^T||y|,$$
$$\leq \gamma_{2n}(2\mathbf{u})|x^T||y|. \tag{7.49}$$

From Eqs. (7.48) and (7.49), we have

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

From Proposition 6.4, using $2\gamma_m(2\mathbf{u}) \leq \gamma_{2m}(2\mathbf{u})$ and noting that the vector r is of length $2n$ yields

$$|\mathtt{res} - x^T y| \leq \left(2\mathbf{u} + 3\gamma_{2n-1}^2(2\mathbf{u})\right)|x^T y| + \gamma_{4n-2}^{K-1}(2\mathbf{u})\left(|x^T y| + \gamma_{2n}(2\mathbf{u})|x^T||y|\right),$$
$$\leq \left(2\mathbf{u} + 3\gamma_{2n-1}^2(2\mathbf{u}) + \gamma_{4n-2}^{K-1}(2\mathbf{u})\right)|x^T y| + \gamma_{2n}(2\mathbf{u})\gamma_{4n-2}^{K-1}(2\mathbf{u})|x^T||y|,$$
$$\leq \left(2\mathbf{u} + \frac{3}{4}\gamma_{4n-2}^2(2\mathbf{u}) + \gamma_{4n-2}^{K-1}(2\mathbf{u})\right)|x^T y| + \gamma_{4n-2}^{K}(2\mathbf{u})|x^T||y|.$$

If $8(2n-1)\mathbf{u} \leq 1$, then we can conclude that $\gamma_{4n-2}(2\mathbf{u}) \leq 1$ and

$$|\mathtt{res} - x^T y| \leq \left(2\mathbf{u} + 2\gamma_{4n-2}^2(2\mathbf{u})\right)|x^T y| + \gamma_{4n-2}^{K}(2\mathbf{u})|x^T||y|.$$

□

From Proposition 7.3, a bound for the relative error on the result of Algorithm DotK obtained with directed rounding is deduced in Corollary 7.4.

Corollary 7.4. *Assume* $16n\mathbf{u} \leq 1$. *The result* res *of Algorithm 13* (DotK) *obtained with directed rounding satisfies*

$$\left|\frac{\mathtt{res} - x^T y}{x^T y}\right| \leq 2\mathbf{u} + 2\gamma_{4n-2}^2(2\mathbf{u}) + \frac{1}{2}\gamma_{4n-2}^{K}(2\mathbf{u})\mathrm{cond}(\mathrm{x^T y}).$$

From Corollary 7.4, the bound for the relative error on the result obtained with directed rounding is essentially the relative rounding error $2\mathbf{u}$ plus a term that reflects that the computation is carried out almost as in K-fold precision ($\alpha(K)\mathbf{u}^K$ times the condition number for a moderate factor $\alpha(K)$).

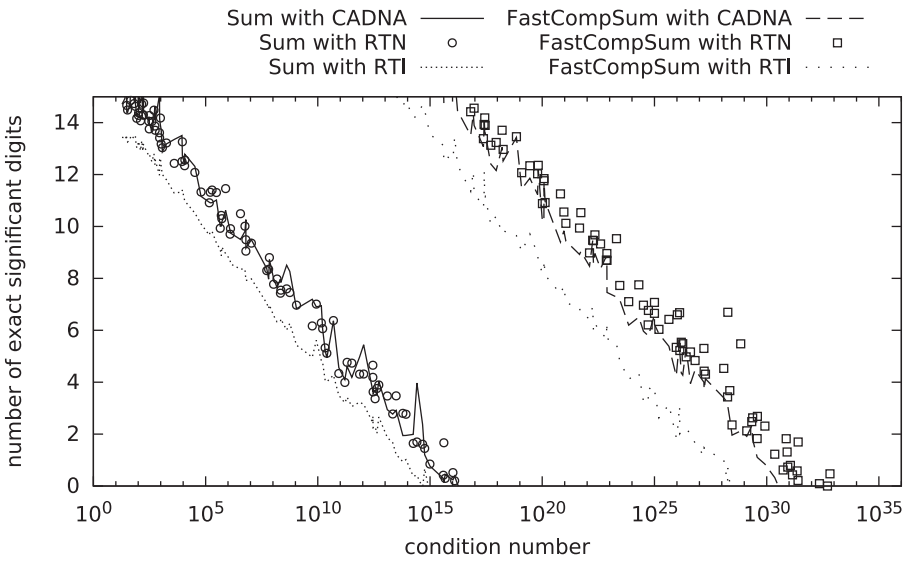

Fig. 1. Accuracy (evaluated from the exact results) of the results computed with CADNA, with rounding to nearest (RTN), and with rounding to plus infinity (RTI), using the Sum and the FastCompSum algorithms with 200 randomly generated floating-point numbers.

8. Numerical results

In the previous sections, we have analysed the error generated by compensated algorithms executed with directed rounding. We have shown the accuracy improvement obtained, not only with rounding to nearest, but also with directed rounding. DSA enables one to estimate rounding errors by using a random rounding mode: either rounding to plus or to minus infinity with the same probability. Therefore the new errors bounds proposed in this article for directed rounding are valid with the random rounding mode of DSA. The numerical experiments presented in this section confirm that compensated algorithms executed with DSA lead to an accuracy improvement. Furthermore in this section we show that DSA enables one to estimate the number of correct digits in results of compensated algorithms.

In the numerical experiments presented here, compensated algorithms previously described are executed with the CADNA library. CADNA is an implementation of DSA devoted to programs written in C/C++ or Fortran. Thanks to three executions of the user program with the random rounding mode, CADNA estimates, with the probability 95%, the number of exact significant digits of any computed result. The CADNA library allows one to use new numerical types: the stochastic types. Each stochastic variable contains three values of the corresponding floating-point type. Arithmetic operators, comparison operators, all the mathematical functions have been overloaded to return a stochastic type when called with stochastic arguments. Therefore the use of CADNA in a program requires only a few modifications: essentially changes in the declarations of variables and in input/output statements. Results presented in this section are computed in double precision, *i.e.* using the *binary64* format of the IEEE 754 standard [5]: each stochastic variable contains three *binary64* floating-point values.

Fig. 1 presents for the `Sum` and the `FastCompSum` algorithms the number d of exact digits obtained from the relative difference between the computed results (hereafter denoted as R) and those computed symbolically (denoted as R_{exact}).

$$\text{If } R_{exact} \neq 0,\ d = -\log_{10}\left|\frac{R - R_{exact}}{R_{exact}}\right|, \qquad \text{otherwise } \ d = -\log_{10}|R|. \tag{8.50}$$

The results R considered for Fig. 1 are computed with CADNA, with rounding to nearest, and with rounding to plus infinity. One can observe that, for the `Sum` algorithm, the accuracy of results computed with CADNA is very close to that of results computed with rounding to nearest. For the `FastCompSum` algorithm, the accuracy of results obtained with rounding to nearest can be better than that of results computed with CADNA. This can be explained by the fact that the `FastCompSum` algorithm has been designed to be used with rounding to nearest. As expected, results computed with CADNA or with rounding to nearest have a better numerical quality than those obtained with rounding to plus infinity. This difference in terms of accuracy is slightly higher with the `FastCompSum` algorithm than with the `Sum` algorithm. The same remarks would have been valid for rounding to minus infinity with the numerical data used in this experiment. Comments on the relation between the condition number and the accuracy of the results are given in the sequel for several classic and compensated algorithms (including the `Sum` and the `FastCompSum` algorithms).

Figs. 2–6 present the accuracy estimated by CADNA of the results obtained using classic algorithms and associated compensated versions. In Figs. 2–4 is also reported the accuracy evaluated from the exact results, *i.e.* the number d of exact

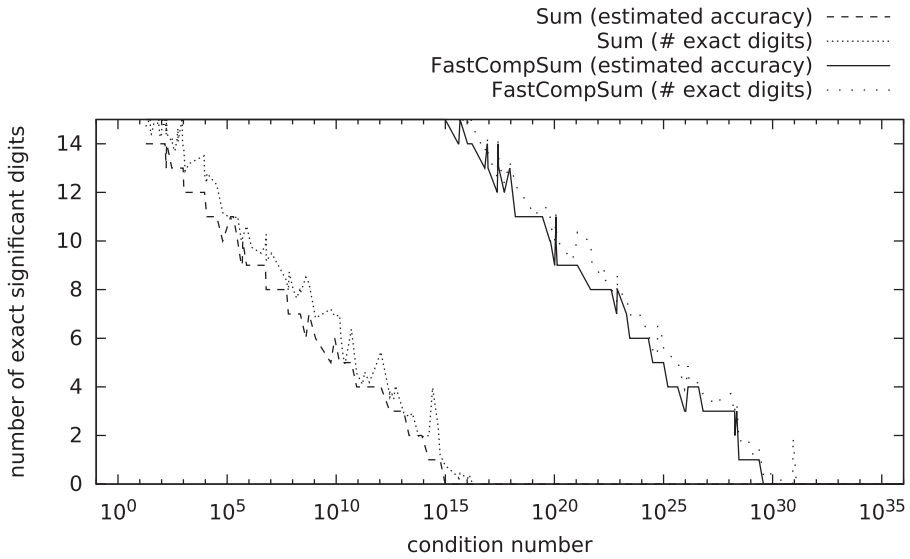

Fig. 2. Accuracy (estimated by CADNA and computed from the exact results) using the Sum and the FastCompSum algorithms with 200 randomly generated floating-point numbers.

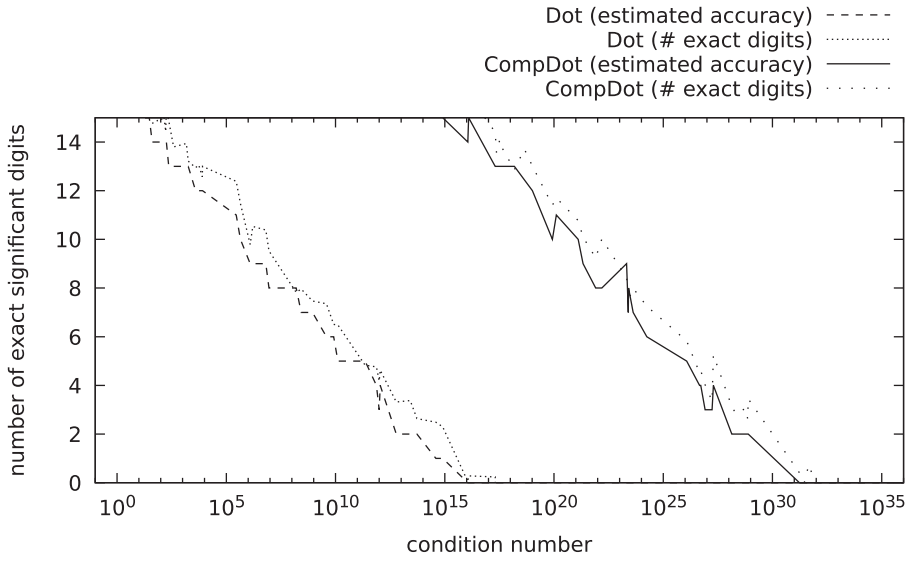

Fig. 3. Accuracy (estimated by CADNA and computed from the exact results) using the Dot and the CompDot algorithms to compute the dot product of arrays of 100 randomly generated elements.

digits obtained using Eq. (8.50) from the relative difference between the results provided by CADNA and those computed symbolically. One can observe in Figs. 2–4 that CADNA correctly estimates the number of exact digits in the results. The accuracy estimation by CADNA is rather pessimistic. However it may happen, for instance in `CompHorner` results (see Fig. 4) that it is slightly optimistic. From Figs. 2–6, if the condition number increases, the accuracy decreases, and with classic algorithms, results have no more correct digits for condition numbers greater than 10^{16}.

From Figs. 2–4, as long as the condition number is less than 10^{15}, the compensated algorithms produce results with the maximal accuracy (15 exact significant digits in double precision). For condition numbers greater than 10^{15}, the accuracy decreases and there are no more correct digits for condition numbers greater than 10^{32}. The results provided by CADNA are consistent with the properties of compensated algorithms given in Section 3–5 for directed rounding: with the current precision, the `FastCompSum`, `CompDot`, and `CompHorner` algorithms compute results similar to those that would have been obtained with twice the working precision.

Figs. 5 and 6 present the accuracy estimated by CADNA of the results computed using the `SumK` and `DotK` algorithms described in Sections 6 and 7. As a remark, although the results obtained with `SumK` and `DotK` for $K = 2$ are reported in Figs 5 and 6, for performance reasons algorithms `FastCompSum` and `CompDot` are preferable for a computation almost

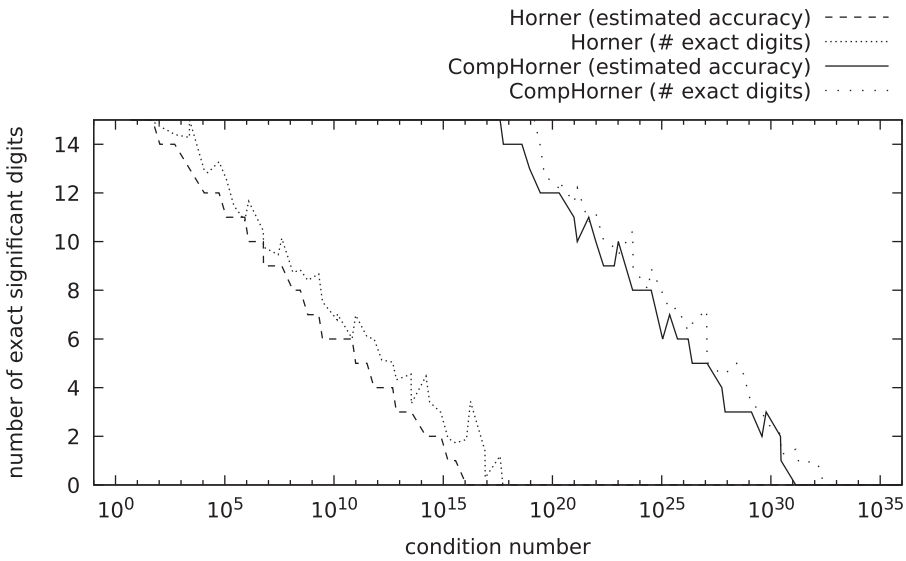

Fig. 4. Accuracy (estimated by CADNA and computed from the exact results) using the `Horner` and the `CompHorner` algorithms to compute $(x-1)^n$ for x close to 1 and for various values of n.

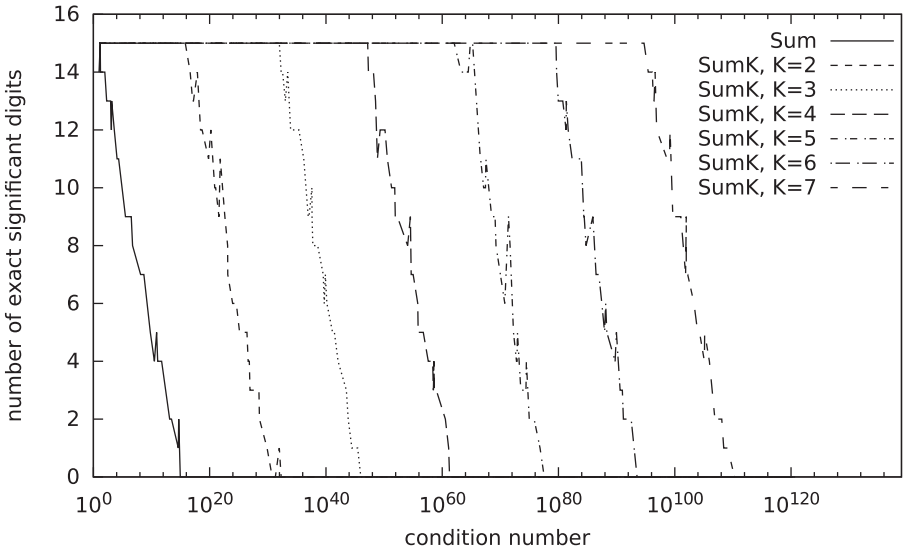

Fig. 5. Accuracy estimated by CADNA using the Sum and the SumK algorithms with 200 randomly generated floating-point numbers.

as with twice the working precision. It can be observed in Figs. 5 and 6 that, if the condition number is less than about $10^{16(K-1)}$, algorithms `SumK` and `DotK` produce results with the maximum possible accuracy in double precision. Then, the accuracy decreases if the condition number increases from about $10^{16(K-1)}$ to 10^{16K}. For condition numbers greater than about 10^{16K}, results computed by `SumK` and `DotK` have no more correct digits. Figs. 5 and 6 are consistent with the properties of algorithms `SumK` and `DotK` given in Sections 6 and 7 for directed rounding: results are computed almost as in K-fold precision.

During the execution, CADNA can detect numerical instabilities, which are usually due to the presence of numerical noise. With CADNA it is essential to perform a so-called *self-validation*, *i.e.* the control of multiplications and divisions during the execution. Indeed if each operand of a multiplication or a divisor has no more correct digits, the estimation of accuracy by DSA may be invalid [1]. The algorithms presented in this article require no division. No multiplication is performed in summation algorithms. Concerning dot product algorithms, the operands of multiplications are elements of the initial arrays. For the evaluation of a polynomial $p(x)$, the value x is always one of the operands of the multiplications performed. Because these initial data are supposed to be significant (not numerical noise), no unstable multiplication can be generated. All algorithms presented in this article may generate cancellations, i.e. sudden losses of accuracy due to subtractions of very close values. The number of cancellations detected depends on the condition number, the size of the data arrays for the

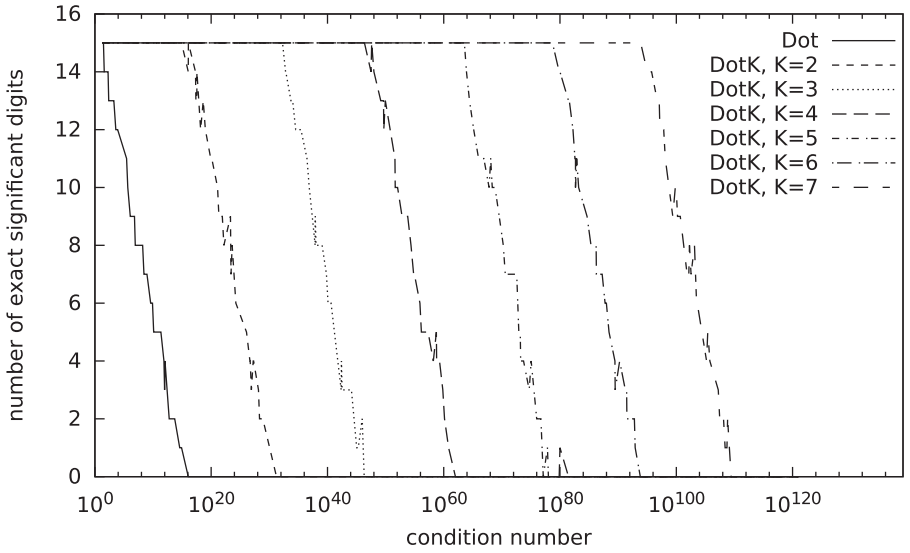

Fig. 6. Accuracy estimated by CADNA using the Dot and the DotK algorithms to compute the dot product of arrays of 100 randomly generated elements.

Table 1
Execution times with and without CADNA for the sum of 10^8 floating-point numbers.

Algorithm	Execution	Execution time (s)	Ratio
`Sum`	without CADNA	8.45E−02	1
	CADNA, self-validation	5.49E−01	6.5
	CADNA, all instabilities	1.62E+00	19.1
`FastCompSum`	without CADNA	1.61E−01	1
	CADNA, self-validation	1.76E+00	10.9
	CADNA, all instabilities	4.54E+00	28.1
`PriestCompSum`	without CADNA	3.79E−01	1
	CADNA, self-validation	3.65E+00	9.6
	CADNA, all instabilities	5.87E+00	15.5
`SumK`, $K = 2$	without CADNA	7.61E−01	1
	CADNA, self-validation	5.12E+00	6.7
	CADNA, all instabilities	7.54E+00	9.9
`SumK`, $K = 3$	without CADNA	1.13E+00	1
	CADNA, self-validation	8.44E+00	7.5
	CADNA, all instabilities	1.12E+01	9.9
`SumK`, $K = 4$	without CADNA	1.51E+00	1
	CADNA, self-validation	1.19E+01	7.9
	CADNA, all instabilities	1.49E+01	9.9
`SumK`, $K = 5$	without CADNA	1.87E+00	1
	CADNA, self-validation	1.52E+01	8.1
	CADNA, all instabilities	1.86E+01	9.8
`SumK`, $K = 6$	without CADNA	2.27E+00	1
	CADNA, self-validation	1.86E+01	8.2
	CADNA, all instabilities	2.24E+01	9.8
`SumK`, $K = 7$	without CADNA	2.64E+00	1
	CADNA, self-validation	2.20E+01	8.3
	CADNA, all instabilities	2.61E+01	9.9

sum and the dot product, and the degree of the polynomial for Horner scheme. Because cancellations are not related to the self-validation of DSA, they cannot invalid the estimation of accuracy by CADNA.

Tables 1–3 present execution times measured in double precision with and without CADNA on an Intel Core i5-4690 CPU (Haswell) at 3.5 GHz using g++ version 5.3.1. As a remark, this architecture supports the `FMA` operation. The codes have been run using CADNA with three kinds of instability detection:

- no detection of instabilities;
- self-validation;

Table 2
Execution times with and without CADNA for the dot product of arrays of $2.5\,10^7$ elements.

Algorithm	Execution	Execution time (s)	Ratio
`Dot`	without CADNA	2.52E–02	1
	CADNA, self-validation	2.14E–01	8.5
	CADNA, all instabilities	5.40E–01	21.4
`CompDot`	without CADNA	5.63E–02	1
	CADNA, self-validation	7.66E–01	13.6
	CADNA, all instabilities	1.68E+00	29.9
`DotK`, $K = 2$	without CADNA	2.86E–01	1
	CADNA, self-validation	1.46E+00	5.1
	CADNA, all instabilities	2.30E+00	8.0
`DotK`, $K = 3$	without CADNA	4.68E–01	1
	CADNA, self-validation	3.31E+00	7.1
	CADNA, all instabilities	4.78E+00	10.2
`DotK`, $K = 4$	without CADNA	6.53E–01	1
	CADNA, self-validation	4.94E+00	7.6
	CADNA, all instabilities	6.84E+00	10.5
`DotK`, $K = 5$	without CADNA	8.36E–01	1
	CADNA, self-validation	6.57E+00	7.8
	CADNA, all instabilities	8.94E+00	10.7
`DotK`, $K = 6$	without CADNA	1.02E+00	1
	CADNA, self-validation	8.19E+00	8.0
	CADNA, all instabilities	1.07E+01	10.5
`DotK`, $K = 7$	without CADNA	1.21E+00	1
	CADNA, self-validation	9.83E+00	8.2
	CADNA, all instabilities	1.26E+01	10.5

Table 3
Execution times with and without CADNA for the evaluation of polynomials of degree $5\,10^7$.

Algorithm	Execution	Execution time (s)	Ratio
`Horner`	without CADNA	4.20E–02	1
	CADNA, self-validation	4.51E–01	10.6
	CADNA, all instabilities	1.38E+00	32.4
`CompHorner`	without CADNA	9.64E–02	1
	CADNA, self-validation	1.61E+00	16.7
	CADNA, all instabilities	3.71E+00	38.7

- the detection of all kinds of instabilities.

With the algorithms considered in this article, the execution times measured with self-validation are very close to those obtained if instability detection is deactivated. With summation algorithms, these times are necessarily the same. Therefore the execution times reported in Tables 1–3 have been measured with self-validation or with the detection of all kinds of instabilities.

From Tables 1–3 the cost of compensated algorithms that compute results almost as with twice the working precision (`FastCompSum`, `CompDot`, `CompHorner`) over the classic algorithms is about 2 without CADNA and about 3 with CADNA. The heavier cost of compensated algorithms with CADNA is mainly explained by the increase of data movements that are more costly with stochastic variables.

The executions times of `PriestCompSum` and `SumK` for $K = 2$ are mentioned in Table 1. However, for performance reasons, `FastCompSum` that is based on `FastTwoSum` is preferable for summations almost as with twice the working precision. Indeed `PriestCompSum` is based on `PriestTwoSum` that requires more floating-point operations and an extra branching statement compared to `FastTwoSum`. `SumK` for $K = 2$ is more costly than `PriestCompSum` because it requires to fetch the errors from an array. Indeed at the end of the computation `SumK` sums up the errors stored in an array, while `PriestCompSum` adds each error as soon as it is available.

Similarly the execution times of `DotK` for $K = 2$ are reported in Table 2, although `CompDot` is preferable to compute dot products almost as with twice the working precision. Indeed `PriestTwoSum` that is called in `DotK` is more costly than `FastTwoSum` called in `CompDot`. Furthermore `DotK` for $K = 2$ sums up the errors fetched from an array at the end of the computation, while `CompDot` adds the errors as soon as they are computed.

One can observe in Table 1 that the cost of `SumK` over the classic summation regularly increases with K, whatever the instability detection level with CADNA and also without CADNA. A similar remark can be formulated from Table 2 about

the cost of DotK over the classic dot product. In all the algorithms mentioned in Tables 1–3, the cost of CADNA in terms of execution time varies from 5 to 17 if self-validation is activated. This overhead is higher if any instability is detected mainly because of the heavy cost of cancellation detection.

9. Conclusion

We have shown that it is possible to validate some compensated algorithms using stochastic arithmetic. We studied compensated summation, dot product and polynomial evaluation. For that, we described the behavior of error-free transformations for addition and multiplication when a random rounding mode is used. We were also able to validate *K*-fold compensated algorithms for summation and dot-product by using a special error-free transformation from Priest [15].

As a future work, we plan to show that it may be possible to validate other compensated algorithms with stochastic arithmetic. We intend to study compensated floating-point product and exponentiation [20], compensated Newton's scheme [21,22], and the compensated evaluation of elementary symmetric functions [23]. This approach can also be easily generalized to the evaluation of polynomials in other bases and of bivariate polynomials (see [24–28] for example). Moreover, a more challenging problem will be to validate the compensated algorithm for solving triangular systems [29].

Acknowledgment

This work was partly supported by the project FastRelax ANR-14-CE25-0018-01. The authors also wish to thank EDF (Electricité De France) for its financial support, and the reviewers for their helpful comments.

References

[1] J.-M. Chesneaux, L'arithmétique stochastique et le logiciel CADNA, Habilitation à diriger des recherches. Université Pierre et Marie Curie, Paris, France, 1995.
[2] F. Jézéquel, J.-M. Chesneaux, CADNA: a library for estimating round-off error propagation, Comput. Phys. Commun. 178 (12) (2008) 933–955.
[3] J. Vignes, Discrete stochastic arithmetic for validating results of numerical software, Numer. Algorit. 37 (1–4) (2004) 377–390.
[4] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, S. Torres, Handbook of Floating-Point Arithmetic, Birkhäuser, Boston, 2010.
[5] IEEE Computer Society, IEEE Standard for Floating-Point Arithmetic, IEEE Standard 754–2008, 2008, doi:10.1109/IEEESTD.2008.4610935.
[6] S. Graillat, F. Jézéquel, R. Picot, Numerical validation of compensated summation algorithms with stochastic arithmetic, Electron. Notes Theor. Comput. Sci. 317 (2015) 55–69, doi:10.1016/j.entcs.2015.10.007.
[7] T. Ogita, S.M. Rump, S. Oishi, Accurate sum and dot product, SIAM J. Sci. Comput. 26 (6) (2005) 1955–1988.
[8] N.J. Higham, Accuracy and Stability of Numerical Algorithms, second, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2002.
[9] G.W. Stewart, Introduction to Matrix Computations, Academic Press, New York-London, 1973.
[10] C.-P. Jeannerod, S.M. Rump, Improved error bounds for inner products in floating-point arithmetic, SIAM J. Matrix Anal. Appl. 34 (2) (2013) 338–344, doi:10.1137/120894488.
[11] S.M. Rump, Error estimation of floating-point summation and dot product, BIT. Numer. Math. 52 (1) (2012) 201–220, doi:10.1007/s10543-011-0342-4.
[12] S.M. Rump, F. Bünger, C.-P. Jeannerod, Improved error bounds for floating-point products and Horner's scheme, BIT. Numer. Math. 56 (1) (2016) 293–307, doi:10.1007/s10543-015-0555-z.
[13] D.E. Knuth, The Art of Computer Programming, Volume 2: Seminumerical Algorithms, third, Addison-Wesley, Boston, MA, USA, 1997.
[14] T.J. Dekker, A floating-point technique for extending the available precision, Numer. Math. 18 (3) (1971) 224–242, doi:10.1007/BF01397083.
[15] D.M. Priest, On Properties of Floating Point Arithmetics: Numerical Stability and the Cost of Accurate Computations, Ph.D. thesis. Mathematics Department, University of California, Berkeley, CA, USA, 1992.
[16] A. Neumaier, Rundungsfehleranalyse einiger Verfahren zur Summation endlicher Summen, ZAMM (Zeitschr. Angew. Math. Mech.) 54 (1974) 39–51.
[17] M. Pichat, Correction d'une somme en arithmĘtique à virgule flottante, Numer. Math. 19 (1972) 400–406.
[18] S. Graillat, P. Langlois, N. Louvet, Algorithms for accurate, validated and fast polynomial evaluation, Jpn. J. Ind. Appl. Math. 2–3 (26) (2009) 191–214. Special issue on State of the Art in Self-Validating Numerical Computations.
[19] S. Graillat, N. Louvet, Ph. Langlois, Compensated Horner scheme, Research Report, 04, Équipe de recherche DALI, Laboratoire LP2A, Université de Perpignan Via Domitia, France, 52 avenue Paul Alduy, 66860 Perpignan cedex, France, 2005.
[20] S. Graillat, Accurate floating point product and exponentiation, IEEE Trans. Comput. 58 (7) (2009) 994–1000.
[21] S. Graillat, Accurate simple zeros of polynomials in floating point arithmetic, Comput. Math. Appl. 56 (4) (2008) 1114–1120.
[22] H. Jiang, S. Graillat, C. Hu, S. Lia, X. Liao, L. Cheng, F. Su, Accurate evaluation of the *k*th derivative of a polynomial, J. Comput. Appl. Math. 191 (2013a) 28–47.
[23] H. Jiang, S. Graillat, R. Barrio, Accurate and fast evaluation of elementary symmetric functions, in: Proceedings of the Twenty-First IEEE Symposium on Computer Arithmetic, Austin, TX, USA, April 7–10, 2013b, pp. 183–190.
[24] P. Du, H. Jiang, L. Cheng, Accurate evaluation of polynomials in legendre basis, J. Appl. Math. 2014 (2014) 742538:1–742538:13.
[25] P. Du, H. Jiang, H. Li, L. Cheng, C. Yang, Accurate evaluation of bivariate polynomials, in: Proceedings of the Seventeenth International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT 2016, Guangzhou, China, December 16–18, 2016, pp. 51–56.
[26] H. Jiang, R. Barrio, H. Li, X. Liao, L. Cheng, F. Su, Accurate evaluation of a polynomial in Chebyshev form, Appl. Math. Comput. 217 (23) (2011) 9702–9716.
[27] H. Jiang, S. Li, L. Cheng, F. Su, Accurate evaluation of a polynomial and its derivative in Bernstein form, Comput. Math. Appl. 60 (3) (2010) 744–755.
[28] P. Du, R. Barrio, H. Jiang, L. Cheng, Accurate quotient-difference algorithm: error analysis, improvements and applications, Appl. Math. Comput. 309 (2017) 245–271.
[29] N. Louvet, Algorithmes compensés en arithmétique flottante : précision, validation, performances, Ph.D. thesis. Université de Perpignan Via Domitia, 2007.