Numerical Validation of Compensated Algorithms with
Stochastic Arithmetic
Stef Graillat, Fabienne Jézéquel, Romain Picot

To cite this version:

HAL Id: hal-01367769
https://hal.archives-ouvertes.fr/hal-01367769
Submitted on 16 Sep 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Numerical validation of compensated algorithms with stochastic arithmetic

S. Graillat*1, F. Jézéquel†1,2, and R. Picot‡1,3

1Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, F-75005, Paris, France
2Université Panthéon-Assas, 12 place du Panthéon, F-75231 Paris CEDEX 05, France
3EDF R&D, 7 boulevard Gaspard Monge, F-91120, Palaiseau, France

September 16, 2016

Abstract

Compensated algorithms consist in computing the rounding error 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 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.

Keywords: CADNA, compensated algorithms, Discrete Stochastic Arithmetic, error-free transformations, floating-point arithmetic, numerical validation, rounding errors.

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 lot 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.

*stef.graillat@upmc.fr
†fabienne.jezequel@lip6.fr
‡romain.picot@lip6.fr
A well-known solution to assert the numerical quality is to use the numerical library called CADNA [13] that implements Discrete Stochastic Arithmetic (DSA) [25] and makes it possible to provide a confidence interval of the computed result.

Nevertheless if the accuracy of the computed result is not sufficient, it is necessary to increase the precision of the computation. One possible technique is the use of compensated algorithms (see [2]). These algorithms are based on error-free transformations (EFTs) that make it possible to compute exactly the rounding errors of some elementary operations like addition and multiplication. We now assume a floating-point arithmetic adhering to the IEEE754-2008 Standard [12]. 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 stochastic arithmetic to validate some numerical codes that heavily rely on the use of error-free transformations.

In this article, we show that we can use stochastic arithmetic to validate compensated summation, dot product and Horner scheme algorithms. Concerning compensated summation, part of this work has been done in [8]. For completeness, we recall some results obtained in Sect. 4.

In Sect. 2, we give some definitions and notations used in the sequel. In Sect. 3, we present the principles of DSA. We show in Sect. 4 that we can still use stochastic arithmetic with compensated summation. Section 5 is devoted to the validation of compensated dot product, while Sect. 6 concerns the validation of the compensated Horner scheme. Section 7 presents the validation of summation in $K$-fold precision, the validation of dot product in $K$-fold precision being done in Sect. 8. Finally, Sect. 9 is devoted to numerical experiments.

## 2 Definitions and notations

In this paper, we assume to work with a binary floating-point arithmetic adhering to IEEE 754 floating-point standard [12] and we suppose that no overflow occurs. The set of floating-point numbers is denoted by $\mathbb{F}$, the relative rounding error by $u$. For IEEE 754 double precision, we have $u = 2^{-53}$ and for single precision $u = 2^{-24}$.

We denote by $\text{fl}^*(\cdot)$ 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 [11]

$$\exists \epsilon_1 \in \mathbb{R}, \epsilon_2 \in \mathbb{R} \text{ such that } \text{fl}^*(a \circ b) = (a \circ b)(1 + \epsilon_1) = (a \circ b)/(1 + \epsilon_2) \text{ for } \circ = \{+, -\} \text{ and } |\epsilon_\nu| \leq 2u. \quad (2.1)$$

As a consequence,

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

We use standard notations for error estimations. The quantities $\gamma_n$ are defined as usual [11] by

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

where implicitly assumed that $nu < 1$. 

2
To keep track of the \((1 + \varepsilon)\) factors in our error analysis, we use the relative error counters introduced by Stewart [22]. For a positive integer \(n\), \(\langle n \rangle\) denotes the following product

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

The relative error counters verify \(\langle j \rangle(u)\langle k \rangle(u) = \langle j \rangle(u)/\langle k \rangle(u) = \langle j + k \rangle(u)\). When \(\langle n \rangle\) denotes any error counter, then there exists a quantity \(\theta_n\) such that

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

Remark 1. We give the following relations about \(\gamma_n\), that will be frequently used in the sequel of the paper. For any positive integer \(n\),

\[
nu \leq \gamma_n(u), \quad \gamma_n(u) \leq \gamma_{n+1}(u), \quad (1 + u)\gamma_n(u) \leq \gamma_{n+1}(u), \quad 2nu(1 + \gamma_{2n-2}(u)) \leq \gamma_{2n}(u).
\]

3 Principles of Discrete Stochastic Arithmetic (DSA)

This section briefly recalls the principles of Discrete Stochastic Arithmetic. Further details are given for instance in [25]. Based on the CESTAC method [24], Discrete Stochastic Arithmetic enables one to estimate in a computed result which digits are affected by rounding errors. It requires a random rounding mode that consists in rounding any result upwards or downwards with the same probability.

To use the CESTAC method in a code that computes a result \(R\), one executes \(N\) times this code with the random rounding mode. Therefore \(N\) different results \(R_i\) are obtained. The value of the computed result \(\overline{R}\) is chosen to be the mean value of \(\{R_i\}\) and, if no overflow occurs, the number of exact significant digits in \(\overline{R}\) can be estimated as

\[
C_{\overline{R}} = \log_{10} \left( \frac{\sqrt{N} |\overline{R}|}{\sigma \tau_{1-\beta}} \right)
\]

where \(\sigma\) is the standard deviation of \(\{R_i\}\) and \(\tau_{1-\beta}\) is the value of Student’s distribution for \(N - 1\) degrees of freedom and a confidence level \(1 - \beta\). In practice \(\beta = 0.05\) and \(N = 3\). Indeed, it has been shown [1, 3] that \(N = 3\) is in some reasonable sense the optimal value. The estimation with \(N = 3\) is more reliable than with \(N = 2\) and increasing the size of the sample does not improve the quality of the estimation.

The CESTAC method relies on a first order model of rounding errors which becomes invalid if both operands in a multiplication or the divisor in a division are not significant [1]. Therefore the CESTAC method requires to control all multiplications and divisions during the execution. This so-called self-validation has led to the concept of computational zero [23], defined below, and also to the synchronous implementation of the method: each arithmetic operation is performed \(N\) times before the next one is executed.

Definition 3.1. A result \(R = \{R_i\}\) computed using the CESTAC method is a computational zero, denoted by \(\mathbf{0}\), if \(\forall i, R_i = 0\) or \(C_{R_i} \leq 0\).

A computational zero is either zero or a result that has no more correct digits because of rounding errors. From the concept of computational zero, discrete stochastic relations [4] have been defined as follows.
Definition 3.2. Let \( X = \{X_i\} \) and \( Y = \{Y_i\} \) be two results computed with the CESTAC method,

1. \( X = Y \) if and only if \( X - Y = \oplus.0 \),
2. \( X > Y \) if and only if \( \overline{X} > \overline{Y} \) and \( X - Y \neq \oplus.0 \),
3. \( X \geq Y \) if and only if \( \overline{X} \geq \overline{Y} \) or \( X - Y = \oplus.0 \).

Discrete Stochastic Arithmetic (DSA) is the combination of the CESTAC method, the concept of computational zero, and the discrete stochastic relations [25].

The CADNA\(^1\) (Control of Accuracy and Debugging for Numerical Applications) library [13] is an implementation of DSA devoted to programs written in C/C++ and Fortran. CADNA provides, with the probability \( 1 - \beta = 95\% \), the number of exact significant digits of any computed result. The CADNA library allows to use new numerical types: the stochastic types. Each stochastic variable contains \( N = 3 \) values of the corresponding floating-point type, one for each sample \( R_i \). When a stochastic variable is printed, only its exact significant digits appear. 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. During the execution, CADNA can detect numerical instabilities, which are usually due to the presence of numerical noise. When numerical instabilities are detected, dedicated CADNA counters are incremented. At the end of the run, the value of these counters together with appropriate warning messages are printed on standard output.

In the next sections, we aim at analysing the effects of the random rounding mode required by DSA on compensated algorithms \textit{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.

4 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.

4.1 Classic summation

The classic algorithm for computing summation is the recursive Algorithm 1.

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

Algorithm 1: Summation of \( n \) floating-point numbers \( p = \{p_i\} \)
```

The error generated by Algorithm 1 is recalled in Proposition 4.1.

\(^1\)URL address: \texttt{http://cadna.lip6.fr}
**Proposition 4.1** ([11]). Let us suppose Algorithm 1 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 rounding to nearest, if $nu < 1$, then

$$|\text{res} - s| \leq \gamma_{n-1}(u)S.$$  \hfill (4.3)

With directed rounding, if $nu < \frac{1}{2}$, then

$$|\text{res} - s| \leq \gamma_{n-1}(2u)S.$$  \hfill (4.4)

In Corollary 4.2 Equations 4.3 and 4.4 are rewritten in terms of the condition number on $\sum p_i$:

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

**Corollary 4.2.** With rounding to nearest, if $nu < 1$, the result $\text{res}$ of Algorithm 1 satisfies

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

With directed rounding, if $nu < \frac{1}{2}$, the result $\text{res}$ of Algorithm 1 satisfies

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

Because $\gamma_{n-1}(u) \approx (n-1)u$, the bound for the relative error is essentially $nu$ times the condition number. This accuracy is sometimes not sufficient in practice. Indeed, when the condition number is large (greater than $1/u$) 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.

### 4.2 Compensated summation with rounding to nearest

Error-free transformations exist for the sum of two floating-point numbers with rounding to nearest: TwoSum [16] which requires 6 floating-point operations and FastTwoSum [5], 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 [21, 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) [19]. 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).

The error on the result $\text{res}$ of Algorithm 3 obtained with rounding to nearest is analysed in [20]. A bound for the absolute error is recalled in Proposition 4.3 and a bound for the relative error in Corollary 4.4.
function \([c, d] = \text{FastTwoSum}(a, b)\)
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 2:** Error-free transformation for the sum of two floating-point numbers with rounding to nearest

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

**Algorithm 3:** Compensated summation of \(n\) floating-point numbers \(p = \{p_i\}\) using FastTwoSum

**Proposition 4.3** ([20]). Let us suppose Algorithm 3 (\textit{FastCompSum}) is applied, with rounding to nearest, 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 \(nu < 1\), then, also in the presence of underflow,

\[
|\text{res} - s| \leq u|s| + \gamma_{n-1}(u)S.
\]

**Corollary 4.4** ([20]). With rounding to nearest, if \(nu < 1\), then, also in the presence of underflow, the result \(\text{res}\) of Algorithm 3 (\textit{FastCompSum}) satisfies

\[
\frac{|\text{res} - s|}{|s|} \leq u + \gamma_{n-1}(u) \text{ cond} \left(\sum p_i\right).
\]

From Corollary 4.4, because \(\gamma_{n-1}(u) \approx (n - 1)u\), the bound for the relative error on the result is essentially \((nu)^2\) times the condition number plus the rounding \(u\) due to the working precision. The second term on the right hand side reflects that the computation is carried out as in twice the working precision \((u^2)\). The first term represents the rounding back in the working precision.

**4.3 Compensated summation with directed rounding**

We recall here the impact of a directed rounding mode on Algorithm 3 (\textit{FastCompSum}). With directed rounding, Algorithm 2 (\textit{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 4.5.

**Proposition 4.5** ([8]). Let \(c\) and \(d\) be the floating-point addition of \(a\) and \(b\) and the correction both computed by Algorithm 2 (\textit{FastTwoSum}) using directed rounding. Let \(e\) be the error on \(c\): \(a + b = c + e\). Then

\[
|e - d| \leq 2u|e|.
\]
Equation 4.5 in Lemma 4.6, established in [8], is recalled for later use.

**Lemma 4.6 ([8]).** Let us suppose Algorithm 3 (FastCompSum) is applied, with directed rounding, to floating-point numbers \( p_i \in \mathbb{F}, 1 \leq i \leq n \). For \( i = 2, \ldots, n \), let \( e_i \) be the error on the floating-point addition of \( \pi_{i-1} \) and \( p_i \): \( \pi_{i-1} + e_i = \pi_{i-1} + p_i \). If \( nu < \frac{1}{2} \), then

\[
\sum_{i=2}^{n} |e_i| \leq \gamma_{n-1}(2u) \sum_{i=1}^{n} |p_i|.
\]  

(4.5)

A bound for the absolute error on the result of Algorithm 3 (FastCompSum) obtained with directed rounding is recalled in Proposition 4.7.

**Proposition 4.7 ([8]).** Let us suppose Algorithm 3 (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 \( nu < \frac{1}{2} \), then, also in the presence of underflow,

\[
|\text{res} - s| \leq 2u|s| + 2(1 + 2u)\gamma_n^2(2u)S.
\]

From Proposition 4.7, a bound for the relative error on the result of Algorithm 3 (FastCompSum) obtained with directed rounding is deduced in Corollary 4.8.

**Corollary 4.8.** With directed rounding, if \( nu < \frac{1}{2} \), then, also in the presence of underflow, the result \( \text{res} \) of Algorithm 3 (FastCompSum) satisfies

\[
\frac{|\text{res} - s|}{|s|} \leq 2u + 2(1 + 2u)\gamma_n^2(2u) \text{cond} \left( \sum p_i \right).
\]

From Corollary 4.8, because \( \gamma_n(2u) \approx 2nu \), the relative error bound is essentially \((nu)^2\) times the condition number plus the inevitable rounding \( 2u \) 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 Sect. 7. The PriestTwoSum algorithm [21, p.14-15] is recalled as Algorithm 4.

```plaintext
function [c, d] = PriestTwoSum(a, b)
1: if |b| > |a| then
2:    exchange a and b
3: end if
4: c ← a + b
5: e ← c - a
6: g ← c - e
7: h ← g - a
8: f ← b - h
9: d ← f - e
10: if d + e ≠ f then
11:   c ← a
12:   d ← b
13: end if
```

**Algorithm 4:** Error-free transformation for the sum of two floating-point numbers with any rounding mode
A compensated summation algorithm based on PriestTwoSum is given as Algorithm 5.

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

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

Lemma 4.9 is given for later use in Sect. 7.

**Lemma 4.9.** Let us suppose Algorithm 5 (PriestCompSum) 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 $nu < \frac{1}{2}$, then

$$\sum_{i=2}^{n} |q_i| + |\pi_n| \leq |s| + 2\gamma_{n-1}(2u)S.$$  \hfill (4.6)

**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

$$\sum_{i=2}^{n} |q_i| + |\pi_n| \leq |s| + 2\sum_{i=2}^{n} |q_i|.$$  \hfill (4.7)

From Lemma 4.6, we deduce that

$$\sum_{i=2}^{n} |q_i| \leq \gamma_{n-1}(2u) \sum_{i=1}^{n} |p_i| = \gamma_{n-1}(2u)S.$$  \hfill (4.8)

Finally Equation 4.6 is obtained from Equations 4.7 and 4.8. \qed

A bound for the absolute error on the result of Algorithm 5 (PriestCompSum) obtained with directed rounding is given in Proposition 4.10.

**Proposition 4.10.** 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 $s := \sum p_i$ and $S := \sum |p_i|$. If $nu < \frac{1}{2}$, then, also in the presence of underflow,

$$|\text{res} - s| \leq 2u|s| + \gamma_{n-1}^2(2u)S.$$  \hfill (4.9)

The proof is similar to the one given in [20] for compensated summation with rounding to nearest.
Proof. Because $\sigma_n = \text{fl}^*(\sum_{i=2}^{n} q_i)$, we have

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

Therefore, from Equation 4.8, we deduce that

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

As Algorithm 5 is executed with directed rounding, it yields

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

$$|\text{res} - s| = |\text{fl}^*(\pi_n + \sigma_n) - s|,$$

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

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

Since

$$s = \sum_{i=1}^{n} p_i = \pi_n + \sum_{i=2}^{n} q_i,$$  \hspace{1cm} (4.10)

then

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

$$|\text{res} - s| \leq (1 + 2u)\gamma_{n-2}(2u)\gamma_{n-1}(2u)S + 2u|s|. \hspace{1cm} (4.11)$$

We have

$$(1 + 2u)\gamma_{n-1}(2u) < \gamma_n(2u). \hspace{1cm} (4.12)$$

Indeed, it is clear that

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

and then

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

Finally Equation 4.9 can be deduced from Equations 4.11 and 4.12. \hfill $\square$

From Proposition 4.10, a bound for the relative error on the result of Algorithm 5 (PriestCompSum) obtained with directed rounding is deduced in Corollary 4.11.
Corollary 4.11. With directed rounding, if \( nu < \frac{1}{2} \), then, also in the presence of underflow, the result \( \text{res} \) of Algorithm 5 (\text{PriestCompSum}) satisfies

\[
\frac{|\text{res} - s|}{|s|} \leq 2u + \gamma_n^{-1}(2u) \text{cond} \left( \sum p_i \right).
\]

Like with Algorithm 3 (\text{FastCompSum}), we deduce from Corollary 4.11 that the relative error bound on the result of Algorithm 5 (\text{PriestCompSum}) computed with directed rounding is essentially \( (nu)^2 \) times the condition number plus the rounding \( 2u \) due to the working precision.

5 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 as in twice the working precision with rounding to nearest [20]. 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.

5.1 Classic dot product

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

```plaintext
function \( \text{res} = \text{Dot}(x, y) \)
1: \( s_1 \leftarrow x_1y_1 \)
2: for \( i = 2 : n \) do
3: \( s_i \leftarrow x_i \cdot y_i + s_{i-1} \)
4: end for
5: \( \text{res} \leftarrow s_n \)

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

The following proposition sums up the properties of this algorithm.

**Proposition 5.1.** Let floating point numbers \( x_i, y_i \in \mathbb{F}, 1 \leq i \leq n \), be given and denote by \( \text{res} \in \mathbb{F} \) the result computed by Algorithm 6 (\text{Dot}). With rounding to nearest, if \( nu < 1 \), we have

\[
|\text{res} - x^T y| \leq \gamma_n(u) |x^T| |y|, \quad (5.13)
\]

and with directed rounding, if \( nu < \frac{1}{2} \), we have

\[
|\text{res} - x^T y| \leq \gamma_n(2u) |x^T| |y|. \quad (5.14)
\]

**Proof.** The proof can be found in Higham [11, p.63].

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

\[
\text{cond}(x^T y) = \frac{|x^T| |y|}{|x^T y|}.
\]
Corollary 5.2. With rounding to nearest, if \( nu < 1 \), the result \( res \) of Algorithm 6 satisfies

\[
\frac{|res - x^T y|}{|x^T y|} \leq \frac{1}{2} \gamma_n(u) \text{cond}(x^T y).
\]

With directed rounding, if \( nu < \frac{1}{2} \), the result \( res \) of Algorithm 6 satisfies

\[
\frac{|res - x^T y|}{|x^T y|} \leq \frac{1}{2} \gamma_n(2u) \text{cond}(x^T y).
\]

5.2 Compensated dot product with rounding to nearest

A compensated dot product algorithm is presented in [20]. This algorithm, intended to be used with rounding to nearest, is based on two error-free transformations: TwoSum [16] and TwoProd [5] 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 ([18, p. 152]).

function \([x, y] = \text{TwoProdFMA}(a, b)\)  
1: \(x \leftarrow a \times b\)  
2: \(y \leftarrow \text{FMA}(a, b, -x)\)  

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

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 \( a, b, c \in \mathbb{F} \), \( \text{FMA}(a, b, c) \) is an approximation of \( a \times b + c \in \mathbb{R} \) that satisfies:

\[
\text{FMA}(a, b, c) = (a \times b + c)(1 + \varepsilon_1) = (a \times b + c)/(1 + \varepsilon_2)
\]

where \(|\varepsilon_\nu| \leq u\) with rounding to nearest and \(|\varepsilon_\nu| \leq 2u\) 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 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.
function res=CompDot(x, y)
1: \([p, s] \leftarrow \text{TwoProdFMA}(x_1, y_1)\)
2: for \(i = 2\) to \(n\) do
3: \([h, r] \leftarrow \text{TwoProdFMA}(x_i, y_i)\)
4: \([p, q] \leftarrow \text{FastTwoSum}(p, h)\)
5: \(s \leftarrow s + (q + r)\)
6: end for
7: res \(\leftarrow p + s\)

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

The error on the result \(\text{res}\) of Algorithm 8 obtained with rounding to nearest is analysed in [20]. A bound for the absolute error is recalled in Proposition 5.3.

**Proposition 5.3 ([20]).** Let floating-point numbers \(x_i, y_i \in \mathbb{F}, 1 \leq i \leq n\), be given and denote by \(\text{res} \in \mathbb{F}\) the result computed by Algorithm 8 (CompDot) with rounding to nearest. If \(nu < 1\), then,

\[
|\text{res} - x^T y| \leq u|x^T y| + \gamma_n^2(u)|x^T||y|. \quad (5.15)
\]

In Corollary 5.4, Equation 5.15 is rewritten in terms of the condition number for the dot product.

**Corollary 5.4 ([20]).** With rounding to nearest, if \(nu < 1\), then, the result \(\text{res}\) of Algorithm 8 (CompDot) satisfies

\[
\frac{|\text{res} - x^T y|}{|x^T y|} \leq u + \frac{1}{2} \gamma_n^2(u) \text{cond}(x^T y).
\]

As a consequence, we conclude that the result is as accurate as if 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.

### 5.3 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.

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

**Algorithm 9:** Equivalent formulation of Algorithm 8

A bound for the absolute error on the result \(\text{res}\) of Algorithm 8 is given in Proposition 5.5.

**Proposition 5.5.** Let floating-point numbers \(x_i, y_i \in \mathbb{F}, 1 \leq i \leq n\), be given and denote by \(\text{res} \in \mathbb{F}\) the result computed by Algorithm 8 (CompDot) with directed rounding. If \((n + 1)u < \frac{1}{2}\), then,

\[
|\text{res} - x^T y| \leq 2u|x^T y| + 2\gamma_{n+1}^2(2u)|x^T||y|.
\]

12
Proof. Thanks to the TwoProdFMA algorithm, we have

\[ p_1 + s_1 = x_1 y_1, \]  

(5.16)

and for \( i \geq 2 \),

\[ h_i + r_i = x_i y_i. \]  

(5.17)

From Proposition 4.5, it follows that

\[ p_i + e_i = p_{i-1} + h_i \quad \text{with} \quad |q_i - e_i| \leq 2u|e_i|. \]  

(5.18)

Therefore from Equation 5.17, 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 Equation 5.16, we derive

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

(5.19)

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

\[ |r_i| \leq 2u|x_i y_i|. \]

Therefore, we have

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

and

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

(5.20)

From Lemma 4.6, we deduce

\[ \sum_{i=2}^{n} |e_i| \leq \gamma_{n-1}(2u) \left( |p_1| + \sum_{i=2}^{n} |h_i| \right), \]

As a consequence, we have

\[ \sum_{i=2}^{n} |e_i| \leq \gamma_{n-1}(2u) \left( \sum_{i=1}^{n} |\text{fl}(x_i y_i)| \right), \]

and

\[ \sum_{i=2}^{n} |e_i| \leq (1 + 2u)\gamma_{n-1}(2u|x^T||y|. \]  

(5.21)

From Equations 4.12 and 5.21, we derive

\[ \sum_{i=2}^{n} |e_i| \leq \gamma_n(2u)|x^T||y|. \]  

(5.22)
From Equation 5.18, we conclude that
\[ \sum_{i=2}^{n} |q_i - e_i| \leq 2u \sum_{i=2}^{n} |e_i|. \]  \hfill (5.23)

Therefore from Equation 5.22, we deduce
\[ \sum_{i=2}^{n} |q_i - e_i| \leq 2u \gamma_\alpha (2u) |x^T||y|. \]  \hfill (5.24)

We have
\[ \sum_{i=2}^{n} |q_i| \leq \sum_{i=2}^{n} |e_i| + \sum_{i=2}^{n} |q_i - e_i|. \]

Therefore, Equation 5.23 yields
\[ \sum_{i=2}^{n} |q_i| \leq (1 + 2u) \sum_{i=2}^{n} |e_i|. \]

From Equations 4.12 and 5.22, it yields
\[ \sum_{i=2}^{n} |q_i| \leq \gamma_{n+1} (2u) |x^T||y|. \]  \hfill (5.25)

For later use, we evaluate an upper bound on the following expression
\[ |s_1 + \sum_{i=2}^{n} (q_i + r_i) - s_n| = |s_1 + \sum_{i=2}^{n} (q_i + r_i) - \text{fl}^*(s_1 + \sum_{i=2}^{n} (q_i + r_i))|. \]

From Proposition 4.1, it follows that
\[ |s_1 + \sum_{i=2}^{n} (q_i + r_i) - s_n| \leq \gamma_{n-1} (2u) \left( |s_1| + \sum_{i=2}^{n} |\text{fl}^*(q_i + r_i)| \right). \]  \hfill (5.26)

Furthermore, because a directed rounding mode is used, we have
\[ \sum_{i=2}^{n} |\text{fl}^*(q_i + r_i)| \leq (1 + 2u) \sum_{i=2}^{n} |q_i + r_i|. \]

Therefore from Equation 5.26, we deduce that
\[ |s_1 + \sum_{i=2}^{n} (q_i + r_i) - s_n| \leq (1 + 2u) \gamma_{n-1} (2u) \left( |s_1| + \sum_{i=2}^{n} |q_i + r_i| \right), \]

and, from Equation 4.12,
\[ |s_1 + \sum_{i=2}^{n} (q_i + r_i) - s_n| \leq \gamma_n (2u) \left( |s_1| + \sum_{i=2}^{n} |q_i + r_i| \right). \]
From Equations 5.20 and 5.25, it follows that
\[
|s_1 + \sum_{i=2}^{n} (q_i + r_i) - s_n| \leq \gamma_n(2u) (2u + \gamma_{n+1}(2u)) |x^T||y|.
\] (5.27)

We deduce from Equation 5.19 that
\[
| (x^T y - p_n) - s_n| = \left| s_1 + \sum_{i=2}^{n} (e_i + r_i) - s_n \right| .
\]

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 Equations 5.24 and 5.27, we deduce that
\[
|x^T y - p_n - s_n| \leq \gamma_n(2u) (4u + \gamma_{n+1}(2u)) |x^T||y|.
\] (5.28)

Let us show that \( \gamma_{n+1}(2u) \geq 4u \). It is easy to show that
\[
\gamma_{n+1}(2u) - 4u = \frac{2(n + 1)u}{1 - 2(n + 1)u} - 4u ,
\]
and
\[
\gamma_{n+1}(2u) - 4u = \frac{2u}{1 - 2(n + 1)u} (n - 1 + 4(n + 1)u) .
\]

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

Therefore from Equation 5.28, we can deduce that
\[
|x^T y - p_n - s_n| \leq 2\gamma_n(2u)\gamma_{n+1}(2u) |x^T||y| .
\] (5.29)

Because Algorithm 9 is executed with a directed rounding mode, it follows that
\[
|res - x^T y| = |(1 + \varepsilon)(p_n + s_n) - x^T y| \quad \text{with} \quad |\varepsilon| \leq 2u .
\]

Therefore, we have
\[
|res - x^T y| = |\varepsilon x^T y + (1 + \varepsilon)(p_n + s_n) - x^T y| ,
\]
and
\[
|res - x^T y| \leq 2u|x^T y| + (1 + 2u)|p_n + s_n - x^T y| .
\]

Then from Equation 5.29, it follows that
\[
|res - x^T y| \leq 2u|x^T y| + 2(1 + 2u)\gamma_n(2u)\gamma_{n+1}(2u)|x^T||y| .
\]

Finally from Equation 4.12, we conclude that
\[
|res - x^T y| \leq 2u|x^T y| + 2\gamma_{n+1}^2(2u)|x^T||y| .
\]

\( \square \)
From Proposition 5.5, a bound for the relative error on the result of Algorithm 8 (CompDot) obtained with directed rounding is deduced in Corollary 5.6.

**Corollary 5.6.** With directed rounding, if \((n + 1)u < \frac{1}{2}\), then, the result \(\text{res}\) of Algorithm 8 (CompDot) satisfies

\[
\frac{|\text{res} - x^T y|}{|x^T y|} \leq 2u + \gamma_{n+1}^2 (2u) \text{cond}(x^T y).
\]

From Corollary 5.6, the relative error bound on the result of Algorithm 8 (CompDot) computed with directed rounding is essentially \((nu)^2\) times the condition number plus the rounding \(2u\) due to the working precision. Like with rounding to nearest, the result obtained with directed rounding is as accurate as if computed in twice the working precision and then rounded to the current working precision.

### 6 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.

#### 6.1 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.

```
function res = Horner(p, x)
    1: s_n ← a_n
    2: for i = n − 1 downto 0 do
    3:     s_i ← s_{i+1} · x + a_i
    4: end for
    5: res ← s_0

Algorithm 10: Polynomial evaluation with Horner’s scheme
```

Whatever the rounding mode, a forward error bound on the result of Algorithm 10 is (see [11, p. 95]):

\[
|p(x) - \text{res}| \leq \gamma_{2n}(2u) \sum_{i=0}^{n} |a_i| |x|^i = \gamma_{2n}(2u) \tilde{p}(|x|)
\]

where \(\tilde{p}(x) = \sum_{i=0}^{n} |a_i| x^i\). It is very interesting to express and interpret this result in terms of the condition number of the polynomial evaluation defined by

\[
\text{cond}(p, x) = \frac{\sum_{i=0}^{n} |a_i| |x|^i}{|p(x)|} = \frac{\tilde{p}(|x|)}{|p(x)|}
\]

(6.30)
Thus we have

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

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 \( s_i \leftarrow \text{FMA}(s_{i+1}, x, a_i) \) which slightly improves the error bound. Using an FMA this way, the computed result now satisfies

\[ |p(x) - \text{res}| \leq \gamma_n(2u)|\tilde{p}||x|. \]

6.2 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 compensated algorithm for the Horner scheme. One can find a more detailed description of the compensated Horner scheme algorithm in [9, 10]. 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.

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

Algorithm 11: Polynomial evaluation with a compensated Horner scheme

If we denote by \( p_\pi \) and \( p_\sigma \) the two following polynomials

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

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 \( s_0 = \text{Horner}(p, x) \). As a consequence, we can derive a new error-free transformation for the polynomial evaluation

\[ p(x) = \text{Horner}(p, x) + p_\pi(x) + p_\sigma(x). \]

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 \( \text{Horner}(p, x) \). We will show that the result computed by Algorithm 11 admits a significantly better error bound than that computed with the classical Horner scheme. We argue that Algorithm 11 provides a result as if it was computed using twice the working precision. This is summed up in the following theorem.
Lemma 6.2. Let $e_{\tilde{\imath}}$ be defined by

\[ (6.33) \]

We recall that

\[ \gamma \]

It follows that

\[ \gamma \]

We now present the impact of a directed rounding mode on Algorithm 11 (CompHorner).

6.3 A compensated Horner scheme with directed rounding

We now present the impact of a directed rounding mode on Algorithm 11 (CompHorner).

Let $\tau_i$ be the rounding error in the floating-point addition of $p_i$ and $a_i$ ($\tau_i$ 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 $|\tau_i - \sigma_i| \leq 2u\tau_i$. 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

\[ s_0 = \text{fl}_s(p(x)), \quad p_\pi(x) = \sum_{i=0}^{n-1} \pi_i x^i, \quad \text{and} \quad p_\tau(x) = \sum_{i=0}^{n-1} \tau_i x^i. \quad (6.33) \]

We recall that

\[ p_\sigma(x) = \sum_{i=0}^{n-1} \sigma_i x^i. \quad (6.34) \]

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

Lemma 6.2. 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 $p_\pi$ and $p_\sigma$ be defined by (6.33) and (6.34). Then, we have

\[ \tilde{p}_\pi(|x|) + \tilde{p}_\sigma(|x|) \leq \gamma_{2n+1}(2u)\tilde{p}(|x|), \]

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

18
Proof. Using (2.1), we have, for $i = 1, \ldots, n$,

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

and

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

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

$$|p_{n-i}| \leq (1 + \gamma_{2i-1}(2u))\sum_{j=1}^{i} |a_{n-i+j}||x^j| \tag{6.35}$$

and

$$|s_{n-i}| \leq (1 + \gamma_{2i}(2u))\sum_{j=0}^{i} |a_{n-i+j}||x^j|. \tag{6.36}$$

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

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

and so (6.35) is true. In the same way, as

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

then (6.36) is also true. Let us assume that (6.35) and (6.36) are true for an integer $i$, $1 \leq i < n$. Then we have

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

By hypothesis, we deduce that

$$|p_{n-(i+1)}| \leq (1 + 2u)(1 + \gamma_{2i}(2u))\sum_{j=0}^{i} |a_{n-i+j}||x^{j+1}|$$

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

Hence, it follows that

$$|s_{n-(i+1)}| \leq (1 + 2u)(|p_{n-(i+1)}| + |a_{n-(i+1)}|)$$

$$\leq (1 + 2u)(1 + \gamma_{2(i+1)-1}(2u))\left[\sum_{j=1}^{i+1} |a_{n-(i+1)+j}||x^j| + |a_{n-(i+1)}|\right]$$

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

Relations (6.35) and (6.36) are then true by induction. As a consequence, for $i = 1, \ldots, n$, we have

$$|p_{n-i}||x^{n-i}| \leq (1 + \gamma_{2i-1}(2u))\tilde{p}(x)$$
and

\[ |s_{n-i}|x^{n-i}| \leq (1 + \gamma_{2i}(2u))\tilde{p}(x). \]

Following (2.2), we have \(|\pi_i| \leq 2u|p_i|, |\tau_i| \leq 2u|s_i|\) and \(|\sigma_i| \leq (1 + 2u)|\tau_i|\) for \(i = 0, \ldots, n - 1\).

Hence,

\[ (\tilde{p}^\pi + \tilde{p}^\sigma)(|x|) = \sum_{i=0}^{n-1} (|\pi_i| + |\sigma_i|)|x|^i \leq 2u(1 + 2u) \sum_{i=1}^{n} (|p_{n-i}| + |s_{n-i}|)|x^{n-i}|, \]

and so

\[ (\tilde{p}^\pi + \tilde{p}^\sigma)(|x|) \leq 2u(1 + 2u) \sum_{i=1}^{n} (2 + \gamma_{2i-1}(2u) + \gamma_{2i}(2u)) \tilde{p}(|x|) \leq 4nu(1 + 2u)(1 + \gamma_{2n}(2u)) \tilde{p}(|x|). \]

As \(4nu(1 + \gamma_{2n}(2u)) = \gamma_{2n}(2u)\), we deduce that \((\tilde{p}^\pi + \tilde{p}^\sigma)(|x|) \leq \gamma_{2n+1}(2u)\tilde{p}(|x|)\).

\[ \square \]

**Lemma 6.3.** Let \(p(x) = \sum_{i=0}^{n} a_ix^i\) be a polynomial with \(a_i \in \mathbb{F}, 0 \leq i \leq n\), \(q(x) = \sum_{i=0}^{n} b_ix^i\) a polynomial with \(b_i \in \mathbb{F}, 0 \leq i \leq n\) and \(x \in \mathbb{F}\). Then the floating-point evaluation of \(r(x) = p(x) + q(x)\) via the following algorithm

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

\[ |\text{res} - r(x)| \leq \gamma_{2n+1}(2u)\tilde{r}(|x|). \]

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

\[ r_i = \text{fl}^*(r_{i+1} \cdot x + (a_i + b_i)) = r_{i+1}x(2)(2u) + (a_i + b_i)(2)(2u). \]

As a consequence, we can show by induction that

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

Moreover with quantities \(\theta_{2n+1}(2u), \theta_2(2u), \ldots, \theta_1(2u)\), satisfying \(|\theta_i(2u)| \leq \gamma_i(2u)\), we have

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

As \(r_0 = \text{fl}^*(p(x) + q(x))\), we finally get

\[ |\text{res} - \sum_{i=0}^{n} (a_i + b_i)x^i| \leq \gamma_{2n+1}(2u)\sum_{i=0}^{n} |a_i + b_i||x^i| \leq \gamma_{2n+1}(2u)(\tilde{p} + \tilde{q})(|x|), \]

which concludes the proof. \[ \square \]
Theorem 6.4. 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

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

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

$$|\text{res} - p(x)| = |(1 + \varepsilon)(s_0 + \text{fl}^*(e(x))) - p(x)|$$

$$= |(1 + \varepsilon)(s_0 + \text{fl}^*(e(x))) - p(x) + (p_r - p_\sigma)(x)| + \varepsilon p(x) + (1 + \varepsilon)(p_\sigma - p_r)(x)|$$

$$= |(1 + \varepsilon)(s_0 + e(x) + (p_r - p_\sigma)(x) - p(x)) + (1 + \varepsilon)(\text{fl}^*(e(x)) - e(x)) +$$

$$\varepsilon p(x) + (1 + \varepsilon)(p_r - p_\sigma)(x)|$$

$$\leq 2u|p(x)| + (1 + 2u)|\text{fl}^*(e(x)) - e(x)| + (1 + 2u)|(p_r - p_\sigma)(x)|.$$ 

By applying Lemma 6.3, we obtain

$$|\text{fl}^*(e(x)) - e(x)| \leq \gamma_{n-1}(2u)\tilde{e}(|x|) \leq \gamma_{2n-1}(2u)(\tilde{p}_\tau(|x|) + \tilde{p}_\sigma(|x|)).$$

Moreover from Lemma 6.2, we get

$$\tilde{p}_\tau(|x|) + \tilde{p}_\sigma(|x|) \leq \gamma_{2n+1}(2u)\tilde{p}(|x|).$$

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

$$|(p_r - p_\sigma)(x)| \leq 2u \sum_{i=0}^{n-1} |\tau_i||x|^i \leq 2u\tilde{p}_r(|x|).$$

Moreover, as $|\tau_i| \leq 2u|s_i|$, we have $\tilde{p}_r(|x|) \leq 2nu\gamma_{2n}(2u)\tilde{p}(|x|)$. As a consequence, we deduce

$$|\text{res} - p(x)| \leq 2u|p(x)| + (1 + 2u)\gamma_{2n-1}(2u)\gamma_{2n+1}(2u)\tilde{p}(|x|) + 2nu(1 + 2u)\gamma_{2n}(2u)\tilde{p}(|x|).$$

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

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

which concludes the proof. \qed

7 Summation as in $K$-fold precision

According to Sect. 4, Algorithms 3 (FastCompSum) and 5 (PriestCompSum) compute the sum of $n$ floating-point numbers as in twice the working precision, even with directed rounding. In this section, we present the $\text{SumK}$ algorithm [20] 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.
7.1 Summation as in $K$-fold precision with rounding to nearest

The \texttt{SumK} algorithm, introduced in [20] and presented as Algorithm 12, enables the summation of a vector of floating-point numbers as in $K$-fold precision. In [20] the \texttt{SumK} algorithm is based on the \texttt{TwoSum} algorithm [16]. However the same result can be obtained using rounding to nearest with any error-free transformation that computes the sum of two floating-point numbers. The \texttt{SumK} algorithm is presented here with Algorithm 4 (\texttt{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 (\texttt{PriestCompSum}).

\begin{algorithm}
\begin{algorithmic}[1]
\Function {res} {\texttt{SumK}(p,K)}
\For {$k = 1$ to $K-1$}
\For {$i = 2$ to $n$}
\State $[p_i,p_{i-1}] \leftarrow \texttt{PriestTwoSum}(p_i,p_{i-1})$
\EndFor
\EndFor
\State $\texttt{res} \leftarrow \sum_{i=1}^{n} p_i$
\EndFunction
\end{algorithmic}
\end{algorithm}

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

The error on the result \texttt{res} of Algorithm 12 obtained with rounding to nearest is analysed in [20]. A bound for the absolute error is recalled in Proposition 7.1 and a bound for the relative error in Corollary 7.2.

\textbf{Proposition 7.1 ([20])}. Let floating-point numbers $p_i \in \mathbb{F}$, $1 \leq i \leq n$, be given and assume $4nu \leq 1$. Then, also in the presence of underflow, the result \texttt{res} of Algorithm 12 (\texttt{SumK}) obtained with rounding to nearest satisfies for $K \geq 3$

$$|\texttt{res} - s| \leq (u + 3\gamma_n^2(u))|s| + \gamma_K S$$

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

\textbf{Corollary 7.2 ([20])}. Assume $4nu \leq 1$. The result \texttt{res} of Algorithm 12 (\texttt{SumK}) obtained with rounding to nearest, also in the presence of underflow, satisfies

$$\frac{|\texttt{res} - s|}{|s|} \leq u + 3\gamma_n^2(u) + \gamma_K \text{cond}(\sum p_i)$$

From Corollary 7.2, because $\gamma_n(u) \approx nu$, the bound for the relative error on the result obtained with rounding to nearest is essentially the relative rounding error $u$ plus a term that reflects that the computation is carried out as in $K$-fold precision ($(\alpha u)^K$ times the condition number for a moderate factor $\alpha$).

7.2 Summation as in $K$-fold precision with directed rounding

We analyse here the impact of a directed rounding mode on Algorithm 12 (\texttt{SumK}). The steps of the error analysis are similar to those presented in [20] 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 $0 \leq k \leq K-1$. 

22
Lemma 7.3. With the above notations, the intermediate results of Algorithm 12 (SumK) satisfy:

\[ s := \sum_{i=1}^{n} p_i^{(0)} = \sum_{i=1}^{n} p_i^{(k)} \text{ for } 1 \leq k \leq K - 1, \tag{7.37} \]

\[ |\text{res} - s| \leq 2u|s| + \frac{2}{(n-1)u} S^{(K-2)}, \tag{7.38} \]

\[ S^{(k)} \leq 3|s| + \gamma_{2n-2}^k (2u) S^{(0)} \quad \text{provided } 8(n-1)u \leq 1 \text{ and } 1 \leq k \leq K - 1. \tag{7.39} \]

Proof. Equation 7.37 follows by successive applications of Equation 4.10. Equation 7.38 can be deduced using \( s = \sum_{i=1}^{n} p_i^{(K-2)} \) and applying Proposition 4.10. Let us now prove Equation 7.39. From Lemma 4.9, we obtain

\[ \sum_{i=1}^{n} |p_i^{(1)}| \leq |s| + 2\gamma_{n-1}(2u) S^{(0)}. \tag{7.40} \]

By applying successively Equation 7.40 and using Equation 7.37 we obtain

\[ S^{(2)} \leq |s| + 2\gamma_{n-1}(2u) \left( |s| + 2\gamma_{n-1}(2u) S^{(0)} \right), \]

and

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

We have

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

If \( 8(n-1)u \leq 1 \), then we get

\[ 1 - 2(n-1)u \leq 3(1 - 6(n-1)u) \quad \text{and} \quad \frac{1 - 2(n-1)u}{1 - 6(n-1)u} \leq 3. \]

Therefore, we have

\[ S^{(k)} \leq 3|s| + (2\gamma_{n-1}(2u))^k S^{(0)}. \]

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

\[ S^{(k)} \leq 3|s| + (\gamma_{2n-2}(2u))^k S^{(0)}. \]

\[ \square \]

A bound for the absolute error on the result of Algorithm 12 (SumK) obtained with directed rounding is given in Proposition 7.4.

**Proposition 7.4.** Let floating-point numbers \( p_i \in \mathbb{F}, 1 \leq i \leq n \), be given and assume \( 8nu \leq 1 \). Then, also in the presence of underflow, the result \( \text{res} \) of Algorithm 12 (SumK) obtained with directed rounding satisfies for \( K \geq 3 \)

\[ |\text{res} - s| \leq (2u + 3\gamma_{n-1}^2(2u)) |s| + \gamma_{2n-2}^K (2u) S \]

where \( s := \sum p_i \) and \( S := \sum |p_i| \).

23
Proof. Proposition 7.4 can be proved by inserting Equation 7.39 of Lemma 7.3 into Equation 7.38.

From Proposition 7.4, a bound for the relative error on the result of Algorithm 12 (SumK) obtained with directed rounding is deduced in Corollary 7.5.

**Corollary 7.5.** Assume $8nu \leq 1$. The result $res$ of Algorithm 12 (SumK) obtained with directed rounding, also in the presence of underflow, satisfies

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

(7.41)

From Corollary 7.5, because $\gamma_n(2u) \approx 2nu$, the bound for the relative error on the result $res$ obtained with directed rounding is essentially the relative rounding error $2u$ plus $(\alpha u)^K$ times the condition number for a moderate factor $\alpha$. Like with rounding to nearest, the last term on the right hand side of Equation 7.41 reflects that the computation is carried out as in $K$-fold precision.

8 Dot product as in $K$-fold precision

According to Sect. 5, Algorithm 8 (CompDot) computes a dot product as in twice the working precision, even with directed rounding. In this section, we present the DotK algorithm [20] that computes a dot product 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 Sect. 5, we assume in this section that no underflow occurs.

8.1 Dot product as in $K$-fold precision with rounding to nearest

The DotK algorithm, introduced in [20] and presented as Algorithm 13, enables one to compute a dot product as in $K$-fold precision. In [20] the DotK algorithm is based on TwoProd [5] and TwoSum [16] 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).

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

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

The error on the result $res$ of Algorithm 13 obtained with rounding to nearest is analysed in [20]. A bound for the absolute error is recalled in Proposition 8.1 and a bound for the relative error in Corollary 8.2.
Proposition 8.1 ([20]). Let floating-point numbers $x_i, y_i \in \mathbb{F}, 1 \leq i \leq n$, be given and assume $8nu \leq 1$. Denote by $\text{res} \in \mathbb{F}$ the result computed by Algorithm 13 (DotK) with rounding to nearest. Then

$$|\text{res} - x^T y| \leq (u + 2 \gamma^2_{4n-2}(u)) |x^T y| + \frac{K}{2} \gamma_{4n-2}(u) \text{cond}(x^T y).$$

Corollary 8.2 ([20]). Assume $8nu \leq 1$. The result $\text{res}$ of Algorithm 13 (DotK) obtained with rounding to nearest satisfies

$$|\text{res} - x^T y| \leq u + 2 \gamma^2_{4n-2}(u) + \frac{1}{2} \gamma_{4n-2}(u) \text{cond}(x^T y).$$

From Corollary 8.2, the bound for the relative error on the result is essentially the relative rounding error $u$ plus a term that reflects that the computation is carried out as in $K$-fold precision ($\alpha(K)u^K$ times the condition number for a moderate factor $\alpha(K)$).

8.2 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.

```
function res = DotK(x, y, K)
1: [p1, r1] ← TwoProdFMA(x1, y1)
2: for $i = 2$ to $n$ do
3:     [hi, ri] ← TwoProdFMA(xi, yi)
4:     [pi, rin+i-1] ← PriestTwoSum(pi-1, hi)
5: end for
6: $r_{2n} ← p_n$
7: res ← SumK(r, K - 1)

Algorithm 14: Equivalent formulation of Algorithm 13
```

A bound for the absolute error on the result of Algorithm DotK obtained with directed rounding is given in Proposition 8.3.

Proposition 8.3. Let floating-point numbers $x_i, y_i \in \mathbb{F}, 1 \leq i \leq n$, be given and denote by $\text{res} \in \mathbb{F}$ the result computed by Algorithm 13 (DotK) with directed rounding. If $16nu \leq 1$, then

$$|\text{res} - x^T y| \leq (2u + 2 \gamma^2_{4n-2}(2u)) |x^T y| + \gamma_{4n-2}(2u) |x^T||y|$$

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

$$s := \sum_{i=1}^{2n} r_i = x^T y.$$  \hspace{1cm} (8.42)

Indeed, it is clear that

$$r_1 = x_1y_1 - p_1,$$

and for $i \geq 2$,

$$r_i + rin+i-1 = (x_iy_i - h_i) + (pi-1 + h_i - pi),$$

$$= x_iy_i + pi-1 - pi.$$
Therefore, because $r_{2n} = p_n$, we have
\[
\sum_{i=1}^{2n-1} r_i = (x_1y_1 - p_1) + \left( \sum_{i=2}^{n} x_iy_i + p_1 - r_{2n} \right),
\]
\[= x^T y - r_{2n}. \tag{8.43}\]

Therefore we can deduce Equation 8.42.

Applying Proposition 7.4 requires to estimate $S := \sum_{i=1}^{2n} |r_i|$. Because Algorithm 13 is executed with directed rounding, we have
\[|r_1| \leq 2u|x_1y_1|, \tag{8.44}\]
and
\[\sum_{i=2}^{n} |r_i| \leq 2u \sum_{i=2}^{n} |x_iy_i|. \tag{8.45}\]

From Lemma 4.6 applied to Algorithm PriestTwoSum, we deduce
\[\sum_{i=2}^{n} |r_{n+i-1}| \leq \gamma_{n-1}(2u) \left( |p_1| + \sum_{i=2}^{n} |h_i| \right),
\[= \gamma_{n-1}(2u) \sum_{i=1}^{n} |f_i(x_iy_i)|,
\[\leq (1 + 2u)\gamma_{n-1}(2u)|x^T||y|. \tag{8.46}\]

From Equations 8.44, 8.45 and 8.46, we obtain
\[\sum_{i=1}^{2n-1} |r_i| \leq 2u|x^T||y| + (1 + 2u)\gamma_{n-1}(2u)|x^T||y|,
\[\leq \frac{2nu}{1 - 2(n-1)u}|x^T||y|. \tag{8.47}\]

From Equation 8.43, 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
\[\sum_{i=1}^{2n} |r_i| \leq |x^T y| + 2 \sum_{i=1}^{2n-1} |r_i|. \tag{8.48}\]

From Equation 8.47, we obtain
\[2 \sum_{i=1}^{2n-1} |r_i| \leq \frac{(2n)(2u)}{1 - (n-1)(2u)}|x^T||y|,
\[\leq \frac{(2n)(2u)}{1 - 2n(2u)}|x^T||y|,
\[\leq \gamma_{2n}(2u)|x^T||y|. \tag{8.49}\]
From Equations 8.48 and 8.49, we have

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

From Proposition 7.4, using \(2\gamma_m(2u) \leq \gamma_{2n}(2u)\) and noting that the vector \(r\) is of length \(2n\) yields

\[ |\text{res} - x^T y| \leq (2u + 3\gamma_{2n-1}(2u))|x^T y| + \frac{\gamma_{4n-2}(2u)}{4} |x^T y| + \gamma_{2n}(2u)\gamma_{4n-2}(2u)|x^T||y|, \]

\[ \leq \left(2u + 3\gamma_{2n-1}(2u) + \gamma_{4n-2}(2u)\right)|x^T y| + \gamma_{2n}(2u)\gamma_{4n-2}(2u)|x^T||y|, \]

\[ \leq \left(2u + \frac{3}{4}\gamma_{2n-1}(2u) + \gamma_{4n-2}(2u)\right)|x^T y| + \gamma_{4n-2}(2u)|x^T||y|. \]

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

\[ |\text{res} - x^T y| \leq \left(2u + 2\gamma_{4n-2}(2u)\right)|x^T y| + \gamma_{4n-2}(2u)|x^T||y|. \]

From Proposition 8.3, a bound for the relative error on the result of Algorithm \texttt{DotK} obtained with directed rounding is deduced in Corollary 8.4.

Corollary 8.4. Assume \(16nu \leq 1\). The result \(\text{res}\) of Algorithm 13 (\texttt{DotK}) obtained with directed rounding satisfies

\[ \left|\frac{\text{res} - x^T y}{x^T y}\right| \leq 2u + 2\gamma_{4n-2}(2u) + \frac{1}{2}\gamma_{4n-2}(2u)\text{cond}(x^T y). \]

From Corollary 8.4, the bound for the relative error on the result obtained with directed rounding is essentially the relative rounding error \(2u\) plus a term that reflects that the computation is carried out as in \(K\)-fold precision \((\alpha(K)u^K\) times the condition number for a moderate factor \(\alpha(K)\)).

9 Numerical results

In the numerical experiments presented here, compensated algorithms previously described are executed with the CADNA library. Results are computed in double precision, i.e. using the binary64 format of the IEEE standard [12]: each stochastic variable contains three binary64 floating-point values. Figures 1 to 5 present the number of exact significant digits estimated by CADNA in the results obtained using classic algorithms and associated compensated versions. One can observe in Fig. 1 to 5 that, if the condition number increases, the number of exact significant digits decreases and with classic algorithms, results have no more correct digits for condition numbers greater than \(10^{16}\).

From Fig. 1 to 3, as long as the condition number is less than \(10^{16}\), the compensated algorithms produce results with the maximal accuracy (15 exact significant digits in double precision). For condition numbers greater than \(10^{16}\), 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 Sect. 4 to 6 for directed rounding: with the current
precision, the FastCompSum, CompDot, and CompHorner algorithms compute results that could have been obtained with twice the working precision.

Figure 1: Accuracy estimated by CADNA using the Sum and the FastCompSum algorithms with 200 randomly generated floating-point numbers.

Figure 2: Accuracy estimated by CADNA using the Dot and the CompDot algorithms to compute the dot product of arrays of 100 randomly generated elements.
Figure 3: Accuracy estimated by CADNA using the Horner and the CompHorner algorithms to compute \((x - 1)^n\) for \(x\) close to 1 and for various values of \(n\).

Figures 4 and 5 present the accuracy estimated by CADNA of the results computed using the SumK and DotK algorithms described in Sect. 7 and 8. As a remark, although the results obtained with SumK and DotK for \(K = 2\) are reported in Fig. 4 and 5, for performance reasons algorithms FastCompSum and CompDot should be preferably used for a computation as with twice the working precision. It can be observed in Fig. 4 and 5 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. Figures 4 and 5 are consistent with the properties of algorithms SumK and DotK given in Sect. 7 and 8 for directed rounding: results are computed as in \(K\)-fold precision.
Figure 4: Accuracy estimated by CADNA using the Sum and the SumK algorithms with 200 randomly generated floating-point numbers.

Figure 5: Accuracy estimated by CADNA using the Dot and the DotK algorithms to compute the dot product of arrays of 100 randomly generated elements.

CADNA can detect numerical instabilities that occur during the execution. As already mentioned in Sect. 3, the use of DSA requires 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. 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, \textit{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 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 to 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 \textit{FMA} operation. The codes have been run using CADNA with three kinds of instability detection:

- no detection of instabilities,
- self-validation,
- 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 to 3 have been measured with self-validation or with the detection of all kinds of instabilities.

From Tables 1 to 3, the cost of compensated algorithms that compute results as with twice the working precision (\textit{FastCompSum}, \textit{CompDot}, \textit{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 \textit{PriestCompSum} and \textit{SumK} for \( K = 2 \) are mentioned in Table 1. However, for performance reasons, \textit{FastCompSum} that is based on \textit{FastTwoSum} should be preferably used to compute summations as with twice the working precision. Indeed \textit{PriestCompSum} is based on \textit{PriestTwoSum} that requires more floating-point operations and an extra branching statement compared to \textit{FastTwoSum}. \textit{SumK} for \( K = 2 \) is more costly than \textit{PriestCompSum} because it requires to fetch the errors from an array. Indeed at the end of the computation \textit{SumK} sums up the errors stored in an array, while \textit{PriestCompSum} adds each error as soon as it is available. Similarly the execution times of \textit{DotK} for \( K = 2 \) are reported in Table 2, although \textit{CompDot} should be preferable used to compute dot products as with twice the working precision. Indeed \textit{PriestTwoSum} that is called in \textit{DotK} is more costly than \textit{FastTwoSum} called in \textit{CompDot}. Furthermore \textit{DotK} for \( K = 2 \) sums up the errors fetched from an array at the end of the computation, while \textit{CompDot} adds the errors as soon as they are computed. One can observe in Table 1 that the cost of \textit{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 \textit{DotK} over the classic dot product. In all the algorithms mentioned in Tables 1 to 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.

<table>
<thead>
<tr>
<th>algorithm</th>
<th>execution time (s)</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sum</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>without CADNA</td>
<td>8.45E–02</td>
<td>1</td>
</tr>
<tr>
<td>CADNA, self-validation</td>
<td>5.49E–01</td>
<td>6.5</td>
</tr>
<tr>
<td>CADNA, all instabilities</td>
<td>1.62E+00</td>
<td>19.1</td>
</tr>
<tr>
<td>FastCompSum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>without CADNA</td>
<td>1.61E–01</td>
<td>1</td>
</tr>
<tr>
<td>CADNA, self-validation</td>
<td>1.76E+00</td>
<td>10.9</td>
</tr>
<tr>
<td>CADNA, all instabilities</td>
<td>4.54E+00</td>
<td>28.1</td>
</tr>
<tr>
<td>PriestCompSum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>without CADNA</td>
<td>3.79E–01</td>
<td>1</td>
</tr>
<tr>
<td>CADNA, self-validation</td>
<td>3.65E+00</td>
<td>9.6</td>
</tr>
<tr>
<td>CADNA, all instabilities</td>
<td>5.87E+00</td>
<td>15.5</td>
</tr>
<tr>
<td><strong>SumK, K = 2</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>without CADNA</td>
<td>7.61E–01</td>
<td>1</td>
</tr>
<tr>
<td>CADNA, self-validation</td>
<td>5.12E+00</td>
<td>6.7</td>
</tr>
<tr>
<td>CADNA, all instabilities</td>
<td>7.54E+00</td>
<td>9.9</td>
</tr>
<tr>
<td><strong>SumK, K = 3</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>without CADNA</td>
<td>1.13E+00</td>
<td>1</td>
</tr>
<tr>
<td>CADNA, self-validation</td>
<td>8.44E+00</td>
<td>7.5</td>
</tr>
<tr>
<td>CADNA, all instabilities</td>
<td>1.12E+01</td>
<td>9.9</td>
</tr>
<tr>
<td><strong>SumK, K = 4</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>without CADNA</td>
<td>1.51E+00</td>
<td>1</td>
</tr>
<tr>
<td>CADNA, self-validation</td>
<td>1.19E+01</td>
<td>7.9</td>
</tr>
<tr>
<td>CADNA, all instabilities</td>
<td>1.49E+01</td>
<td>9.9</td>
</tr>
<tr>
<td><strong>SumK, K = 5</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>without CADNA</td>
<td>1.87E+00</td>
<td>1</td>
</tr>
<tr>
<td>CADNA, self-validation</td>
<td>1.52E+01</td>
<td>8.1</td>
</tr>
<tr>
<td>CADNA, all instabilities</td>
<td>1.86E+01</td>
<td>9.8</td>
</tr>
<tr>
<td><strong>SumK, K = 6</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>without CADNA</td>
<td>2.27E+00</td>
<td>1</td>
</tr>
<tr>
<td>CADNA, self-validation</td>
<td>1.86E+01</td>
<td>8.2</td>
</tr>
<tr>
<td>CADNA, all instabilities</td>
<td>2.24E+01</td>
<td>9.8</td>
</tr>
<tr>
<td><strong>SumK, K = 7</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>without CADNA</td>
<td>2.64E+00</td>
<td>1</td>
</tr>
<tr>
<td>CADNA, self-validation</td>
<td>2.20E+01</td>
<td>8.3</td>
</tr>
<tr>
<td>CADNA, all instabilities</td>
<td>2.61E+01</td>
<td>9.9</td>
</tr>
</tbody>
</table>

Table 1: Execution times with and without CADNA for the sum of 10^8 floating-point numbers.
<table>
<thead>
<tr>
<th>algorithm</th>
<th>execution</th>
<th>execution time (s)</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dot</td>
<td>without CADNA</td>
<td>2.52E–02</td>
<td>1</td>
</tr>
<tr>
<td>CADNA, self-validation</td>
<td>2.14E–01</td>
<td>8.5</td>
<td></td>
</tr>
<tr>
<td>CADNA, all instabilities</td>
<td>5.40E–01</td>
<td>21.4</td>
<td></td>
</tr>
<tr>
<td>CompDot</td>
<td>without CADNA</td>
<td>5.63E–02</td>
<td>1</td>
</tr>
<tr>
<td>CADNA, self-validation</td>
<td>7.66E–01</td>
<td>13.6</td>
<td></td>
</tr>
<tr>
<td>CADNA, all instabilities</td>
<td>1.68E+00</td>
<td>29.9</td>
<td></td>
</tr>
<tr>
<td>DotK, $K = 2$</td>
<td>without CADNA</td>
<td>2.86E–01</td>
<td>1</td>
</tr>
<tr>
<td>CADNA, self-validation</td>
<td>1.46E+00</td>
<td>5.1</td>
<td></td>
</tr>
<tr>
<td>CADNA, all instabilities</td>
<td>2.30E+00</td>
<td>8.0</td>
<td></td>
</tr>
<tr>
<td>DotK, $K = 3$</td>
<td>without CADNA</td>
<td>4.68E–01</td>
<td>1</td>
</tr>
<tr>
<td>CADNA, self-validation</td>
<td>3.31E+00</td>
<td>7.1</td>
<td></td>
</tr>
<tr>
<td>CADNA, all instabilities</td>
<td>4.78E+00</td>
<td>10.2</td>
<td></td>
</tr>
<tr>
<td>DotK, $K = 4$</td>
<td>without CADNA</td>
<td>6.53E–01</td>
<td>1</td>
</tr>
<tr>
<td>CADNA, self-validation</td>
<td>4.94E+00</td>
<td>7.6</td>
<td></td>
</tr>
<tr>
<td>CADNA, all instabilities</td>
<td>6.84E+00</td>
<td>10.5</td>
<td></td>
</tr>
<tr>
<td>DotK, $K = 5$</td>
<td>without CADNA</td>
<td>8.36E–01</td>
<td>1</td>
</tr>
<tr>
<td>CADNA, self-validation</td>
<td>6.57E+00</td>
<td>7.8</td>
<td></td>
</tr>
<tr>
<td>CADNA, all instabilities</td>
<td>8.94E+00</td>
<td>10.7</td>
<td></td>
</tr>
<tr>
<td>DotK, $K = 6$</td>
<td>without CADNA</td>
<td>1.02E+00</td>
<td>1</td>
</tr>
<tr>
<td>CADNA, self-validation</td>
<td>8.19E+00</td>
<td>8.0</td>
<td></td>
</tr>
<tr>
<td>CADNA, all instabilities</td>
<td>1.07E+01</td>
<td>10.5</td>
<td></td>
</tr>
<tr>
<td>DotK, $K = 7$</td>
<td>without CADNA</td>
<td>1.21E+00</td>
<td>1</td>
</tr>
<tr>
<td>CADNA, self-validation</td>
<td>9.83E+00</td>
<td>8.2</td>
<td></td>
</tr>
<tr>
<td>CADNA, all instabilities</td>
<td>1.26E+01</td>
<td>10.5</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>algorithm</th>
<th>execution</th>
<th>execution time (s)</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horner</td>
<td>without CADNA</td>
<td>4.20E–02</td>
<td>1</td>
</tr>
<tr>
<td>CADNA, self-validation</td>
<td>4.51E–01</td>
<td>10.6</td>
<td></td>
</tr>
<tr>
<td>CADNA, all instabilities</td>
<td>1.38E+00</td>
<td>32.4</td>
<td></td>
</tr>
<tr>
<td>CompHorner</td>
<td>without CADNA</td>
<td>9.64E–02</td>
<td>1</td>
</tr>
<tr>
<td>CADNA, self-validation</td>
<td>1.61E+00</td>
<td>16.7</td>
<td></td>
</tr>
<tr>
<td>CADNA, all instabilities</td>
<td>3.71E+00</td>
<td>38.7</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Execution times with and without CADNA for the evaluation of polynomials of degree $5 \times 10^7$. 
10 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 [21].

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 [7], compensated Newton’s scheme [6, 15], and the compensated evaluation of elementary symmetric functions [14]. Moreover, a more challenging problem will be to validate the compensated algorithm for solving triangular systems [17].

Acknowledgement

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.

References


