Japan J. Indust. Appl. Math., 26 (2009), 191–214 Area ⟨2⟩

Algorithms for Accurate, Validated and Fast Polynomial Evaluation*

Stef Graillat†, Philippe Langlois‡ and Nicolas Louvet§

† *PEQUAN, LIP6, Université Pierre et Marie Curie, CNRS, Paris, France*
E-mail: stef.graillat@lip6.fr
‡ *DALI, ELIAUS, Université de Perpignan Via Domitia, France*
E-mail: langlois@univ-perp.fr
§ *Arénaire, LIP, INRIA, Université de Lyon, CNRS, France*
E-mail: nicolas.louvet@ens-lyon.fr

Received April 20, 2008
Revised December 5, 2008

We survey a class of algorithms to evaluate polynomials with floating point coefficients and for computation performed with IEEE-754 floating point arithmetic. The principle is to apply, once or recursively, an error-free transformation of the polynomial evaluation with the Horner algorithm and to accurately sum the final decomposition. These compensated algorithms are as accurate as the Horner algorithm performed in K times the working precision, for K an arbitrary positive integer. We prove this accuracy property with an a priori error analysis. We also provide validated dynamic bounds and apply these results to compute a faithfully rounded evaluation. These compensated algorithms are fast. We illustrate their practical efficiency with numerical experiments on significant environments. Comparing to existing alternatives these K-times compensated algorithms are competitive for K up to 4, i.e., up to 212 mantissa bits.

Key words: polynomial evaluation, compensated algorithm, floating-point arithmetic, IEEE-754

1. Introduction

One of the main computing process with polynomials is evaluation. Higham devotes an entire chapter to polynomials and more especially to polynomial evaluation [7, Chap. 5]. The small backward error the Horner algorithm introduces in floating point arithmetic justifies its practical interest. Nevertheless the Horner algorithm returns results arbitrarily less accurate than the working precision $\mathbf{u}$ when evaluating $p(x)$ is ill-conditioned. The relative accuracy of the computed evaluation with the Horner algorithm (`Horner`) satisfies the following well known a priori bound:

$$\frac{|\texttt{Horner}(p,x) - p(x)|}{|p(x)|} \leq \mathrm{cond}(p,x) \times \mathcal{O}(\mathbf{u}). \tag{1.1}$$

In the right-hand side of this inequality, $\mathbf{u}$ is the computing precision and the condition number $\mathrm{cond}(p,x) \geq 1$ only depends on the entry x and on the coefficients of p—its expression will be given further. Evaluation is ill-conditioned for example

*This work has been prepared while the authors were members of DALI at ELIAUS laboratory of Université de Perpignan and partly funded by the ANR Project EVA-Flo (ANR-BLAN06-2-135670 2006).

in the neighborhood of multiple roots where most of the digits, or even the order of magnitude of the computed value of $p(x)$ can be false [3].

Numerous multiprecision libraries are available when the computing precision $\mathbf{u}$ is not sufficient to guarantee the expected accuracy. Fixed-length expansions such as double-double or quad-double libraries [1] are effective solutions to simulate twice or four times the IEEE-754 double precision. These fixed-length expansions are currently embedded in major developments such as the new extended and mixed precision BLAS [15]. On the other hand, techniques similar to the compensated summation (see [7, pp. 83–88]) can be used, such as in [18] where efficient algorithms for summation and dot product are introduced only using the IEEE-754 double precision.

The *compensated algorithms* we here consider are similar to those of Ogita, Rump and Oishi [18]. The compensated Horner algorithm introduced in [4] is a fast alternative to the Horner algorithm implemented with double-double arithmetic. By fast we mean that this compensated evaluation runs at least twice as fast as the double-double counterpart with the same output accuracy. As the Horner algorithm with double-double arithmetic, the accuracy of the compensated Horner algorithm (`CompHorner`) now satisfies

$$\frac{|\texttt{CompHorner}(p,x) - p(x)|}{|p(x)|} \leq \mathbf{u} + \mathrm{cond}(p,x) \times \mathcal{O}(\mathbf{u}^2). \tag{1.2}$$

Comparing to Relation (1.1), this relation means that the computed value is now as accurate as the result of the Horner algorithm performed in twice the working precision with a final rounding back to this working precision $\mathbf{u}$—the same behavior is proved in [18] for compensated summation and dot product. This motivates the two following issues we focus in this paper.

The bound (1.2) tells us that the compensated Horner algorithm may yield a full precision accuracy for not too ill-conditioned polynomials, that is for p and x such that the second term $\mathrm{cond}(p,x) \times \mathcal{O}(\mathbf{u}^2)$ is small compared to the working precision $\mathbf{u}$. We describe how to guarantee a faithfully rounded evaluation, i.e., how to compute one of the two consecutive floating point values that enclose $p(x)$.

As in Relation (1.1) the accuracy of the compensated result still depends on the condition number and may be arbitrarily bad for ill-conditioned polynomial evaluations. We describe how to iterate the compensating process and improve the accuracy of the computed result by a factor $\mathbf{u}$ at every iteration step. So we derive `CompHornerK`, a K-fold compensated Horner algorithm that satisfies the following a priori bound for any arbitrary integer K:

$$\frac{|\texttt{CompHornerK}(p,x) - p(x)|}{|p(x)|} \leq \mathbf{u} + \mathrm{cond}(p,x) \times \mathcal{O}(\mathbf{u}^K). \tag{1.3}$$

Comparing to Relation (1.1), Relation (1.3) means that the computed value with `CompHornerK` is now as accurate as the result of the Horner algorithm performed in K times the working precision with a final rounding back to this working precision.

We invite the reader to jump to Fig. 5 at the end of this article to visualise this interesting behavior.

The time penalty to improve the accuracy with these compensated algorithms justifies their practical interest. `CompHorner` is at least twice faster than its challenger in double-double arithmetic. Despite its exponential complexity with respect to K, `CompHornerK` is an efficient alternative to other software solutions such as the quad-double library or MPFR [6, 16] for $K \leq 4$. We report significant practical performances where an optimized version of `CompHornerK` with $K = 4$ runs about 40 % faster than the corresponding routine with quad-double arithmetic. In [13] we exhibit that the instruction level parallelism of the compensated algorithms justifies such good measured computing times when running on up-to-date superscalar processors.

Many problems in Computer Assisted Design reduce to find the roots of a polynomial equation, which is accuracy sensitive when dealing with multiple roots. Accurate polynomial evaluation algorithms are investigated in this area [8]. Our compensated algorithms may be used with success in such cases (while neither underflow nor overflow appears) since no restriction applies to the magnitude of $|x|$, nor to the coefficients neither to the degree of the polynomial.

We start illustrating this motivation with a simple example. Let us consider the evaluation in the neighborhood of its multiple roots of $p(x) = (0.75 - x)^5(1 - x)^{11}$, written in its expanded form. Double precision IEEE-754 arithmetic is used for these experiments and the coefficients of p in the monomial basis are double precision numbers. We evaluate $p(x)$ in the neighborhood of its multiple roots 0.75 and 1 with the three algorithms `Horner`, `CompHorner` and `CompHornerK` with $K = 3$. Fig. 1 presents these evaluations for 400 equally spaced points in intervals $[0.68, 1.15]$, $[0.74995, 0.75005]$ and $[0.9935, 1.0065]$. It is clear that accurate polynomial evaluation is necessary to recover the expected smooth drawing at a reasonable focus.

The paper is organized as follows. In Section 2 we recall the classic notations and the basic error free transformations (EFT) for floating point arithmetic. In Section 3, a first compensated algorithm for polynomial evaluation, `CompHorner`, is derived from the Horner algorithm and its associated EFT. In Section 4 we highlight the accuracy of the compensated Horner algorithm exhibiting how to guarantee a faithfully rounded polynomial evaluation; numerical experiments also illustrate the sharpness of the proposed dynamic bound of the evaluation accuracy. In Section 5 we explain how to derive new EFT for the Horner algorithm. In Section 6 we apply this recursive EFT and introduce a K-fold compensated algorithm, `CompHornerK`; we prove its accuracy satisfies Relation (1.3) and illustrate it with some numerical experiments. In the last Section 7 we demonstrate the practical efficiency in terms of running time comparing our algorithms and up-to-date challengers on several significant computing environments.

Due to page limitation most of the proofs have not been detailed and some even not included. Some technical developments have not been presented here, e.g., how to take care of underflow and subnormal floating point numbers or how to benefit from some specific arithmetic operators or how to explain that the measured

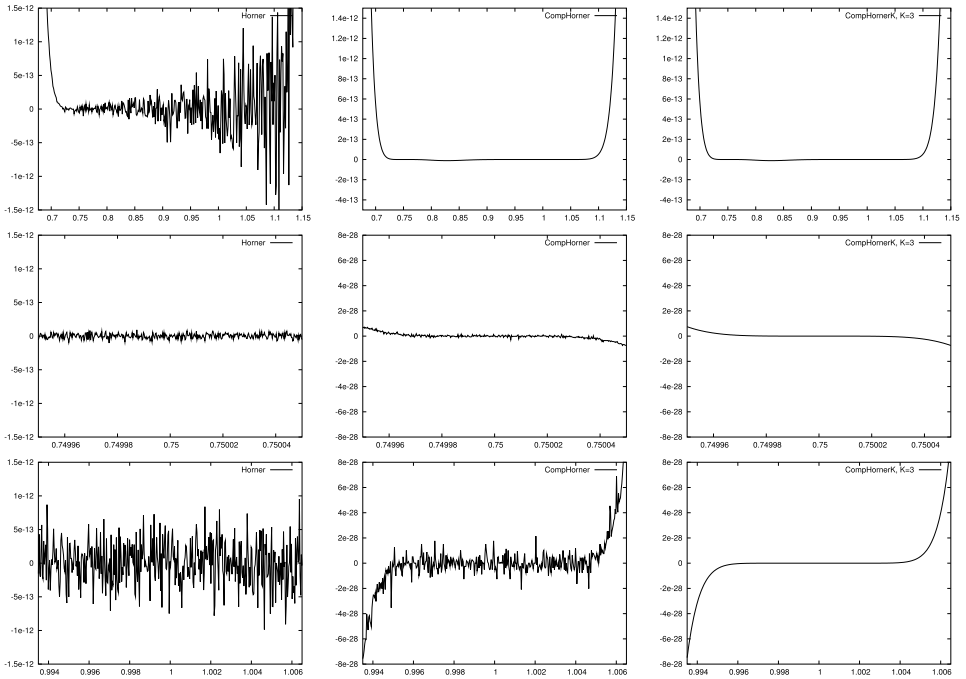


Fig. 1. Evaluation of $p(x) = (0.75 - x)^5(1 - x)^{11}$ in the neighborhood of its multiple roots, using `Horner` (left), `CompHorner` (center) and `CompHornerK` with $K = 3$ (right).

running times are significantly better than the classic flop count. We invite the interested readers to consider the references [11, 4, 5, 12, 13, 14], from the authors, to complete this survey on accurate, validated and fast polynomial evaluation thanks to compensated algorithms.

2. Floating point arithmetic and basic error-free transformations

Throughout this paper, we assume a floating point arithmetic adhering to the IEEE-754 floating point standard [9]. We constraint all the computations to be performed in one working precision, with the "round to the nearest" rounding mode. We also assume that no overflow nor underflow occurs during the computations.

2.1. Standard notations for floating point arithmetic

Next notations are standard—see [7, Chap. 2] for example. $\mathbb{F}$ is the set of all normalized floating point numbers and $\mathbf{u}$ denotes the unit round-off, that is half the spacing between 1 and the next representable floating point value. For IEEE-754 double precision with rounding to the nearest, we have $\mathbf{u} = 2^{-53} \approx 1.11 \cdot 10^{-16}$.

The symbols $\oplus$, $\ominus$, $\otimes$ and $\oslash$ represent respectively the floating point addition, subtraction, multiplication and division. For more complicated arithmetic expressions, $\mathrm{fl}(\cdot)$ denotes the result of a floating point computation where every operation inside the parenthesis is performed in the working precision. So we have for example, $a \oplus b = \mathrm{fl}(a + b)$.

When no underflow nor overflow occurs, the following standard model describes the accuracy of every considered floating point computation: for $a, b \in \mathbb{F}$ and for $\circ \in \{+, -, \times, /\}$, we have

$$\mathrm{fl}(a \circ b) = (a \circ b)(1 + \varepsilon_1) = (a \circ b)/(1 + \varepsilon_2), \quad \text{with } |\varepsilon_1|, |\varepsilon_2| \leq \mathbf{u}. \tag{2.1}$$

To keep track of the $(1 + \varepsilon)$ factors in the error analysis, we use the classic $(1 + \theta_k)$ and γ_k notations [7, Chap. 3]. For any positive integer k, θ_k denotes a quantity bounded according to $|\theta_k| \leq \gamma_k := k\mathbf{u}/(1 - k\mathbf{u})$. When using these notations, we always implicitly assume $k\mathbf{u} < 1$. In further a priori error analysis, we essentially use the following relations:

$$(1 + \theta_k)(1 + \theta_j) \leq (1 + \theta_{k+j}), \quad k\mathbf{u} \leq \gamma_k, \quad \gamma_k \leq \gamma_{k+1}.$$

Moreover, if $k(k+1)\mathbf{u} \leq 1$ then $(1+\gamma_k)\gamma_k \leq \gamma_{k+1}$. To derive validated and dynamic bounds we also use the next relations [18]:

$$\hat{\gamma}_k := (k\mathbf{u}) \oslash (1 \ominus k\mathbf{u}), \quad \gamma_k \leq (1 + \mathbf{u})\hat{\gamma}_k, \quad (1 + \mathbf{u})^n |x| \leq \mathrm{fl}\left(\frac{|x|}{1 - (n + 1)\mathbf{u}}\right). \tag{2.2}$$

We define the floating point predecessor and successor of a real number r as follows:

$$\mathrm{pred}(r) = \max\{f \in \mathbb{F} \mid f < r\} \quad \text{and} \quad \mathrm{succ}(r) = \min\{f \in \mathbb{F} \mid r < f\}.$$

A floating point number f is defined to be a faithful rounding of a real number r if

$$\mathrm{pred}(f) < r < \mathrm{succ}(f).$$

We present further how to compute a faithfully rounded evaluation of a polynomial. Faithful rounding means that the computed result equals the exact result if the latter is a floating point number and otherwise is one of the two consecutive floating point numbers enclosing the exact result $p(x)$.

2.2. EFT for the elementary operations

In this section, we review well known results concerning the error free transformations (EFT) of the elementary floating point operations $\oplus$, $\ominus$ and $\otimes$. These EFT are the core of the implementation of fixed length expansions, as double-double or quad-double [1]; we will also use it within compensated algorithms.

Let $\circ$ be an operator in $\{+, -, \times\}$, a and b be two floating point numbers, and $\hat{x} = \mathrm{fl}(a \circ b)$. Then there exist a floating point value y such that

$$a \circ b = \hat{x} + y. \tag{2.3}$$

The difference y between the exact result and the computed result is the elementary rounding error in the computation of $\hat{x}$. Let us emphasize that Relation (2.3)

between four floating point values relies on real operators and exact equality, i.e., not on approximate floating point counterparts. Ogita, Rump and Oishi call such an equality an error free transformation (EFT) in [18]. The practical interest of the EFT comes from next Algorithms 1 and 3 that compute the exact error term y for $\oplus$ and $\otimes$.

ALGORITHM 1. *EFT of the sum of two floating point numbers.*

function $[x, y] = \texttt{TwoSum}(a, b)$
$x = a \oplus b$
$z = x \ominus a$
$y = (a \ominus (x \ominus z)) \oplus (b \ominus z)$

ALGORITHM 2. *Splitting of a floating point number into two parts.*

function $[x, y] = \texttt{Split}(a)$
$z = a \otimes (2^r + 1)$
$x = z \ominus (z \ominus a)$
$y = a \ominus x$

ALGORITHM 3. *EFT of the product of two floating point numbers.*

function $[x, y] = \texttt{TwoProd}(a, b)$
$x = a \otimes b$
$[a_h, a_l] = \texttt{Split}(a)$
$[b_h, b_l] = \texttt{Split}(b)$
$y = a_l \otimes b_l \ominus (((x \ominus a_h \otimes b_h) \ominus a_l \otimes b_h) \ominus a_h \otimes b_l)$

For the EFT of the addition we use Algorithm 1, the well known `TwoSum` algorithm by Knuth [10] that requires 6 flop (floating point operations). For the product, we first need to split the input arguments into two parts. It is done using Algorithm 2 of Dekker [2]. If q is the number of bits of the mantissa, let $r = \lceil q/2 \rceil$. Algorithm 2 splits a floating point number a into two parts x and y, both having at most $r - 1$ nonzero bits, such that $a = x + y$. For example, with the IEEE-754 double precision, $q = 53$, $r = 27$, therefore the output numbers have at most 26 bits. The trick is that one bit sign is used for the splitting. Next, Algorithm 3 of Veltkamp (see [2]) can be used for the EFT of the product. This algorithm is commonly called `TwoProd` and requires 17 flop.

The next theorem exhibits interesting properties of `TwoSum` and `TwoProd`.

THEOREM 4 ([18]). *Let a, b in $\mathbb{F}$ and $x, y \in \mathbb{F}$ such that $[x, y] = \texttt{TwoSum}(a, b)$ (*Algorithm 1*). Then the floating point addition verifies*

$$a + b = x + y, \quad x = a \oplus b, \quad |y| \leq \mathbf{u}|x|, \quad |y| \leq \mathbf{u}|a + b|.$$

Let $a, b \in \mathbb{F}$ and $x, y \in \mathbb{F}$ such that $[x, y] =$ `TwoProd`(a, b) *(*Algorithm 3*). Then the floating point product verifies*

$$a \times b = x + y, \quad x = a \otimes b, \quad |y| \leq \mathbf{u}|x|, \quad |y| \leq \mathbf{u}|a \times b|.$$

We notice that algorithms `TwoSum` and `TwoProd` only require well optimizable floating point operations. They do not use branches, nor access to the mantissa that can be time-consuming. `TwoProd` can be rewritten straightforwardly for processors that provide a fused-multiply-and-add operator (`FMA`), such as Intel Itanium or IBM PowerPC [17, 18]. For a, b and c in $\mathbb{F}$, `FMA`(a, b, c) is exactly $a \times b + c$ rounded to the nearest floating point value. Thus computing $y = a \times b - a \otimes b =$ `FMA`$(a, b, -a \otimes b)$ now only costs 2 flop. We discuss how to manage such optimization in [5].

3. From Horner algorithm to compensated Horner algorithm

We recall the classic Horner algorithm to introduce a first EFT for the polynomial evaluation. We apply this `EFTHorner` to derive our compensated Horner algorithm. We end this section proving the accuracy behavior of this compensated algorithm previously announced by Relation (1.2).

3.1. Accuracy with the Horner algorithm

The classic condition number of the evaluation of $p(x) = \sum_{i=0}^{n} a_i x^i$ at a given entry x is [3]

$$\mathrm{cond}(p, x) = \frac{\tilde{p}(x)}{|p(x)|} \quad \text{with } \tilde{p}(x) = \sum_{i=0}^{n} |a_i|\,|x|^i. \tag{3.1}$$

For any floating point value x we denote by `Horner`(p, x) the result of the floating point evaluation of the polynomial p at x using the Horner algorithm.

Algorithm 5. *Horner algorithm.*

---

function $r_0 =$ `Horner`(p, x)
  $r_n = a_n$
  for $i = n - 1 : -1 : 0$
    $r_i = r_{i+1} \otimes x \oplus a_i$
  end

We can now write Relation (1.1) with more details. The accuracy of the result of Algorithm 5 is linked to the condition number of the polynomial evaluation thanks to the following forward error bound:

$$\frac{|\texttt{Horner}(p, x) - p(x)|}{|p(x)|} \leq \gamma_{2n}\,\mathrm{cond}(p, x). \tag{3.2}$$

Clearly, the condition number $\mathrm{cond}(p, x)$ can be arbitrarily large: we cannot guarantee that the computed result `Horner`(p, x) has any correct digit when $\mathrm{cond}(p, x) > \gamma_{2n}^{-1}$.

3.2. An EFT for the Horner algorithm

We now propose an EFT for the polynomial evaluation with the Horner algorithm. We prove that the error generated by the Horner algorithm is exactly the sum of two polynomials with floating point coefficients. This allows us to write an algorithm to approximate this generated error.

ALGORITHM 6. *EFT for the Horner algorithm.*

---

function $[r_0, p_\pi, p_\sigma] = \texttt{EFTHorner}(p, x)$
  $r_n = a_n$
  for $i = n-1 : -1 : 0$
    $[p_i, \pi_i] = \texttt{TwoProd}(r_{i+1}, x)$
    $[r_i, \sigma_i] = \texttt{TwoSum}(p_i, a_i)$
    Let π_i *be the coefficient of degree* i *in* p_π.
    Let σ_i *be the coefficient of degree* i *in* p_σ.
  end

THEOREM 7 ([4, 12]). *Let* $p(x) = \sum_{i=0}^{n} a_i x^i$ *be a polynomial of degree* n *with floating point coefficients, and let* x *be a floating point value. Then* `EFTHorner` *(*Algorithm 6*) computes both*

i) *the floating point evaluation* $\texttt{Horner}(p, x)$ *and*

ii) *two polynomials* p_π *and* p_σ *of degree* $n-1$ *with floating point coefficients, such that*

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

The Horner algorithm satisfies

$$p(x) = \texttt{Horner}(p, x) + (p_\pi + p_\sigma)(x); \tag{3.3}$$

and we have

$$\widetilde{(p_\pi + p_\sigma)}(x) \leq (\widetilde{p_\pi} + \widetilde{p_\sigma})(x) \leq \gamma_{2n}\tilde{p}(x). \tag{3.4}$$

Relation (3.3) means that algorithm `EFTHorner` is an EFT for the polynomial evaluation with the Horner algorithm.

Proof of Theorem 7. Since `TwoProd` and `TwoSum` are EFT from Theorem 4, it is easy to verify that at the end of the loop we have

$$r_0 = \sum_{i=0}^{n} a_i x^i - \sum_{i=0}^{n-1} \pi_i x^i - \sum_{i=0}^{n-1} \sigma_i x^i,$$

which proves Relation (3.3). Using the standard model of floating-point arithmetic it can be proved by induction that we have

$$|p_{n-i}| \leq (1 + \gamma_{2i-1}) \sum_{j=1}^{i} |a_{n-i+j}|\,|x^j|, \quad \text{and} \quad |r_{n-i}| \leq (1 + \gamma_{2i}) \sum_{j=0}^{i} |a_{n-i+j}|\,|x^j|,$$

for $i = 1, \ldots, n$. Since $[p_i, \pi_i] = \texttt{TwoProd}(r_{i+1}, x)$ and $[r_i, \sigma_i] = \texttt{TwoSum}(p_i, a_i)$, according to Theorem 4 we have $|\pi_i| \leq \mathbf{u}|p_i|$ and $|\sigma_i| \leq \mathbf{u}|r_i|$ for $i = 0, \ldots, n-1$. As a consequence,

$$\begin{aligned}(\widetilde{p_\pi} + \widetilde{p_\sigma})(x) &\leq \mathbf{u} \sum_{i=1}^{n} (|p_{n-i}| + |r_{n-i}|)|x^{n-i}| \\ &\leq \mathbf{u} \sum_{i=1}^{n} (2 + 2\gamma_{2i})\tilde{p}(x) \leq 2n\mathbf{u}(1 + \gamma_{2n})\tilde{p}(x).\end{aligned}$$

Since $2n\mathbf{u}(1 + \gamma_{2n}) = \gamma_{2n}$, we obtain $(\widetilde{p_\pi} + \widetilde{p_\sigma})(x) \leq \gamma_{2n}\tilde{p}(x)$. □

It is now easy to define the compensated Horner algorithm.

3.3. Compensated Horner algorithm

From Theorem 7 the forward error in the floating point evaluation of p at x with the Horner algorithm is

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

where the two polynomials p_π and p_σ are exactly identified by `EFTHorner` (Algorithm 6)—this latter also computes $\texttt{Horner}(p, x)$. Therefore, the key of the algorithm proposed in this section is to compute an approximate $\hat{c}$ of the forward error c in the working precision, and then a corrected result

$$\bar{r} = \texttt{Horner}(p, x) \oplus \hat{c}.$$

We say that $\hat{c}$ is a correcting term for $\texttt{Horner}(p, x)$. The corrected result $\bar{r}$ is expected to be more accurate than the first result $\texttt{Horner}(p, x)$ as proved in the rest of the section. The previous remarks leads to next algorithm `CompHorner` (Algorithm 8).

ALGORITHM 8. *Compensated Horner algorithm.*

---

function $\bar{r} = \texttt{CompHorner}(p, x)$
  $[\hat{r}, p_\pi, p_\sigma] = \texttt{EFTHorner}(p, x)$
  $\hat{c} = \texttt{Horner}(p_\pi \oplus p_\sigma, x)$
  $\bar{r} = \hat{r} \oplus \hat{c}$

3.4. Accuracy of the compensated Horner algorithm

We prove hereafter that the result of a polynomial evaluation computed with the compensated Horner algorithm (`CompHorner`) is as accurate as if computed by the classic Horner algorithm (`Horner`) using twice the working precision and then rounded to the working precision.

THEOREM 9 ([4, 12]). *Consider a polynomial p of degree n with floating point coefficients, and x a floating point value. The forward error in the compensated Horner algorithm is such that*

$$|\texttt{CompHorner}(p,x) - p(x)| \leq \mathbf{u}|p(x)| + \gamma_{2n}^2 \tilde{p}(x). \tag{3.5}$$

Proof. The forward error in Algorithm 8 is

$$|\bar{r} - p(x)| = |(\hat{r} \oplus \hat{c}) - p(x)| = |(1+\varepsilon)(\hat{r} + \hat{c}) - p(x)| \quad \text{with} \quad |\varepsilon| \leq \mathbf{u}.$$

Let $c = (p_\pi + p_\sigma)(x)$. From Theorem 7 we have $\hat{r} = \texttt{Horner}(p,x) = p(x) - c$, thus

$$|\bar{r} - p(x)| = |(1+\varepsilon)(p(x) - c + \hat{c}) - p(x)| \leq \mathbf{u}|p(x)| + (1+\mathbf{u})|\hat{c} - c|.$$

Since $\hat{c} = \texttt{Horner}(p_\pi \oplus p_\sigma, x)$ with p_π and p_σ two polynomials of degree $n-1$, we verify that $|\hat{c} - c| \leq \gamma_{2n-1}(\widetilde{p_\pi + p_\sigma})(x)$. Then using (3.4) we have $|\hat{c} - c| \leq \gamma_{2n-1}\gamma_{2n}\tilde{p}(x)$. Since $(1+\mathbf{u})\gamma_{2n-1} \leq \gamma_{2n}$, we finally write the expected error bound (3.5). □

In [4] we prove that the same behavior is still satisfied when underflow occurs. For later use, we notice that $|\hat{c} - c| \leq \gamma_{2n-1}\gamma_{2n}\tilde{p}(x)$ implies

$$|\hat{c} - c| \leq \gamma_{2n}^2 \tilde{p}(x). \tag{3.6}$$

It is interesting to interpret the previous theorem in terms of the condition number of the polynomial evaluation of p at x. Combining the error bound (3.5) with the condition number (3.1) for polynomial evaluation gives

$$\frac{|\texttt{CompHorner}(p,x) - p(x)|}{|p(x)|} \leq \mathbf{u} + \gamma_{2n}^2 \operatorname{cond}(p,x). \tag{3.7}$$

In other words, the bound for the relative error of the computed result is essentially γ_{2n}^2 times the condition number of the polynomial evaluation, plus the inevitable term $\mathbf{u}$ for rounding the result to the working precision. In particular, if $\operatorname{cond}(p,x) < \gamma_{2n}^{-1}$, then the relative accuracy of the result is bounded by a constant of the order $\mathbf{u}$. 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 $\gamma_{2n}^{-1} \approx (2n\mathbf{u})^{-1}$. Besides that, Relation (3.7) tells us that the computed result is as accurate as if computed by the classic Horner algorithm with twice the working precision, and then rounded to the working precision.

4. Faithfully rounded polynomial evaluation

Since the compensated Horner algorithm yields accurate results for small condition numbers, we now discuss when and how this evaluation returns a faithfully rounded floating point value of the exact result. Thanks to an a priori error analysis, we provide a sufficient criterion on the condition number $\operatorname{cond}(p,x)$ to ensure

that the corrected result $\bar{r}$ computed with `CompHorner` is a faithful rounding of the exact result $p(x)$. We also derive a validated running error analysis for `CompHorner`, which leads to a sharper enclosure of the computed result $\bar{r}$. In particular, our numerical experiments show that the computed evaluation can be proved to be faithfully rounded at the running time, as long as the condition number is smaller than the inverse of the working precision.

4.1. A priori condition for faithful rounding

Next lemma provides a sufficient condition on the accuracy of the correcting term $\hat{c}$ to ensure faithful rounding.

LEMMA 10. *Let p be a polynomial of degree n with floating point coefficients, and x be a floating point value. We consider the approximate $\bar{r}$ of $p(x)$ computed with* `CompHorner`(p, x). *Let c denote $c = (p_\pi + p_\sigma)(x)$. If $|\hat{c} - c| < \frac{\mathbf{u}}{2}|\bar{r}|$, then $\bar{r}$ is a faithful rounding of $p(x)$.*

Proof (see [12] *for details).* The proof relies on [19, Lemma 2.4] and Relation (3.3). □

From Relation (3.6), we know that the absolute error $|\hat{c} - c|$ is bounded by $\gamma_{2n}^2\tilde{p}(x)$. This provides us a more useful criterion about the condition number $\mathrm{cond}(p, x)$ to ensure that `CompHorner` computes a faithfully rounded result.

PROPOSITION 11. *Let p be a polynomial of degree n with floating point coefficients, and x a floating point value. If*

$$\mathrm{cond}(p, x) < \frac{1 - \mathbf{u}}{2 + \mathbf{u}}\mathbf{u}\gamma_{2n}^{-2} \approx \frac{1}{8n^2}\mathbf{u}^{-1}, \tag{4.1}$$

then `CompHorner`(p, x) *computes a faithful rounding of the exact $p(x)$.*

Proof (see [12] *for details).* Using Relation (3.5) and Inequality (3.6) we prove $(1 - \mathbf{u})|p(x)| - \gamma_{2n}^2\tilde{p}(x) \leq |\bar{r}|$. Relation (3.6) and a small computation give

$$|\hat{c} - c| \leq \gamma_{2n}^2\tilde{p}(x) < \frac{\mathbf{u}}{2}\left[(1 - \mathbf{u})|p(x)| - \gamma_{2n}^2\tilde{p}(x)\right] \leq \frac{\mathbf{u}}{2}|\bar{r}|.$$

From Lemma 10 we deduce that $\bar{r}$ is a faithful rounding of $p(x)$. □

Numerical values for the upper bound (4.1) to ensure faithful rounding with the compensated Horner algorithm are presented in Table 1 for degrees varying from 10 to 300 and IEEE-754 double precision.

Table 1. A priori bound on the condition number for a faithfully rounded polynomial of degree n.

n	10	100	200	300
$\frac{1-\mathbf{u}}{2+\mathbf{u}}\mathbf{u}\gamma_{2n}^{-2} \approx \frac{1}{8n^2}\mathbf{u}^{-1}$	$1.13 \cdot 10^{13}$	$1.13 \cdot 10^{11}$	$2.82 \cdot 10^{10}$	$1.13 \cdot 10^{10}$

4.2. Dynamic error bounds

The previous results are perfectly suited for theoretical purposes, for instance when we can a priori bound the condition number of the evaluation. However, neither the error bound in Theorem 9, nor the criterion proposed in Proposition 11 can be easily checked using only floating point arithmetic. Here we provide dynamical counterparts of Theorem 9 and Proposition 11 that can be evaluated using floating point arithmetic in the "round to the nearest" rounding mode.

LEMMA 12. *Consider a polynomial p of degree n with floating point coefficients, and x a floating point value. We use the notations of* Algorithm 8*, and we denote $(p_\pi + p_\sigma)(x)$ by c. Then*

$$|c - \hat{c}| \leq \mathrm{fl}\left(\frac{\hat{\gamma}_{2n-1}\,\mathtt{Horner}(|p_\pi| \oplus |p_\sigma|, |x|)}{1 - 2(n+1)\mathbf{u}}\right) := \hat{\alpha}. \tag{4.2}$$

Proof. Let us denote $\mathtt{Horner}(|p_\pi| \oplus |p_\sigma|, |x|)$ by $\hat{b}$. Since $c = (p_\pi + p_\sigma)(x)$ and $\hat{c} = \mathtt{Horner}(p_\pi \oplus p_\sigma, x)$ where p_π and p_σ are two polynomials of degree $n-1$. Bounding the error in $\mathtt{Horner}(p_\pi \oplus p_\sigma, x)$ as done for $\mathtt{Horner}(p, x)$, we write

$$|c - \hat{c}| \leq \gamma_{2n-1}(\widetilde{p_\pi + p_\sigma})(x) \leq (1+\mathbf{u})^{2n-1}\gamma_{2n-1}\hat{b}.$$

From the first inequality in Relation (2.2) and the standard model (2.1) it follows that

$$|c - \hat{c}| \leq (1+\mathbf{u})^{2n}\hat{\gamma}_{2n-1}\hat{b} \leq (1+\mathbf{u})^{2n+1}\,\mathrm{fl}(\hat{\gamma}_{2n-1}\hat{b}).$$

Finally we use the second inequality in Relation (2.2) to obtain the error bound. □

Lemma 12 allows us to compute an error bound for the computed correcting term $\hat{c}$. From Theorem 7 we know that $p(x) = \hat{r} + c$, and since $\bar{r} = \hat{r} \oplus \hat{c}$ we write

$$|\bar{r} - p(x)| \leq |(\hat{r} \oplus \hat{c}) - (\hat{r} + \hat{c})| + |(\hat{c} - c)|.$$

The first term $|(\hat{r} \oplus \hat{c}) - (\hat{r} + \hat{c})|$ is the absolute rounding error that occurs when computing $\hat{r} \oplus \hat{c}$. Using only (2.1), it could be bounded by $\mathbf{u}|\bar{r}|$. But here we use algorithm `TwoSum` to compute the actual rounding error exactly which slightly improves the error bound.

PROPOSITION 13. *Consider a polynomial p of degree n with floating point coefficients, and x a floating point value. We use the notations of* Algorithm 8*, and we assume that e is the floating point value such that $\bar{r} + e = \hat{r} + \hat{c}$, i.e., $[\bar{r}, e] = \mathtt{TwoSum}(\hat{r}, \hat{c})$. Moreover, let $\hat{\alpha}$ be the error bound defined by* (4.2)*. Then, the absolute error on the computed result $\bar{r} = \mathtt{CompHorner}(p, x)$ is bounded according to*

$$|\bar{r} - p(x)| \leq \mathrm{fl}\left(\frac{\hat{\alpha} + |e|}{1 - 2\mathbf{u}}\right) := \hat{\beta}. \tag{4.3}$$

Moreover, if $\hat{\alpha} < \frac{\mathbf{u}}{2}|\bar{r}|$, then $\bar{r}$ is a faithful rounding of $p(x)$.

Proof. By hypothesis $\bar{r} = \hat{r} + \hat{c} - e$, and from Theorem 7 we have $p(x) = \hat{r} + c$, thus

$$|\bar{r} - p(x)| = |\hat{c} - c - e| \leq |\hat{c} - c| + |e| \leq \hat{\alpha} + |e|.$$

From the standard model (2.1) and the second inequality in (2.2) it follows that

$$|\bar{r} - p(x)| \leq (1 + \mathbf{u})\,\mathrm{fl}(\hat{\alpha} + |e|) \leq \mathrm{fl}\left(\frac{\hat{\alpha} + |e|}{1 - 2\mathbf{u}}\right).$$

The second part of the proposition follows directly from Lemma 10. □

From Proposition 13 we conclude with next algorithm `CompHornerIsFaithful`. It computes the compensated result $\bar{r}$ together with the validated error bound $\hat{\beta}$. Moreover, the boolean value `isfaithful` is set to true if and only if the result is proved to be faithfully rounded. As usual, we assume that no overflow nor underflow occurs during the computations.

ALGORITHM 14. *Compensated Horner algorithm with check of the faithful rounding.*

---

function $[\bar{r}, \hat{\beta}, \texttt{isfaithful}] = \texttt{CompHornerIsFaithful}(p, x)$
$[\hat{r}, p_\pi, p_\sigma] = \texttt{EFTHorner}(p, x)$
$\hat{c} = \texttt{Horner}(p_\pi \oplus p_\sigma, x)$
$\hat{b} = \texttt{Horner}(|p_\pi| \oplus |p_\sigma|, |x|)$
$[\bar{r}, e] = \texttt{TwoSum}(\hat{r}, \hat{c})$
$\hat{\alpha} = (\hat{\gamma}_{2n-1} \otimes \hat{b}) \oslash (1 \ominus 2(n+1) \otimes \mathbf{u})$
$\hat{\beta} = (\hat{\alpha} \oplus |e|) \oslash (1 - 2 \otimes \mathbf{u})$
$\texttt{isfaithful} = \left(\hat{\alpha} < \frac{\mathbf{u}}{2}|\bar{r}|\right)$

4.3. Testing the faithful rounding and the sharpness of the dynamic bound

We focus on both the a priori and the dynamic bounds. We recall that two cases may occur when the dynamic test for faithful rounding in `CompHornerIsFaithful` is performed.

1. If the dynamic test is satisfied, this proves that the compensated result is a faithful rounding of the exact $p(x)$. Corresponding plots are reported with a square (□) in Fig. 2.
2. If the dynamic test fails then the compensated result may or may not be faithfully rounded. We distinguish two sub-cases where we compare the compensated results to reference ones obtained from high-precision computation.
   a) If the compensated result is actually faithfully rounded, we report a filled circle (•);
   b) when the compensated result does not faithfully round $p(x)$ we plot a cross (×).

We generate polynomials of degree 50 whose condition numbers vary from 10^2 to 10^{35} (see [12] for details about the generation algorithm). The results of the tests with `CompHornerIsFaithful` (Algorithm 14) are reported with Fig. 2—on this plot the horizontal axis represents the condition number (3.1).

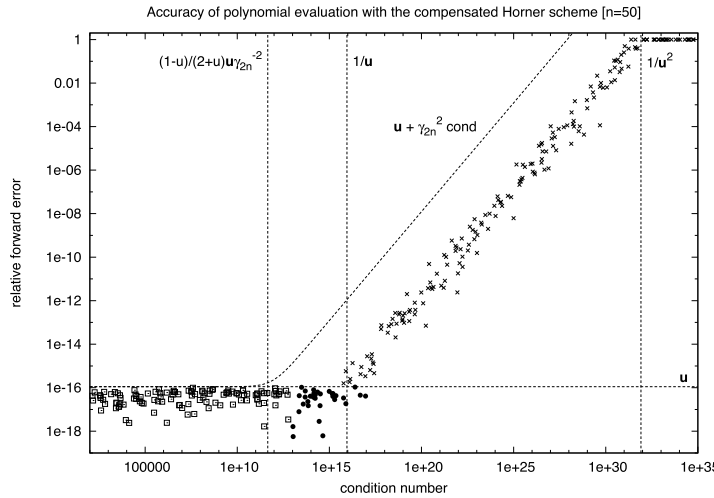


Fig. 2. Accuracy of `CompHornerIsFaithful` w.r.t. the condition number. The leftmost vertical line is the a priori condition (4.1) and the broken line is the a priori bound (3.7).

We observe that the compensated algorithm exhibits the expected behavior. The relative error in the compensated result is smaller than the working precision $\mathbf{u}$—the horizontal line—as long as the condition number is smaller than $1/\mathbf{u}$—the rightmost vertical line. Then, for condition numbers between $1/\mathbf{u}$ and $1/\mathbf{u}^2$, this relative error degrades to no accuracy at all. As usual, the a priori error bound (3.7) appears to be pessimistic by many orders of magnitude—compare the observed behavior with the comments we provide just after Relation (3.7)

The a priori sufficient condition (4.1) for faithful rounding with respect to the condition number is also represented on Fig. 2 by the leftmost vertical line. As expected, every polynomial evaluation with a condition number smaller than this a priori bound (4.1) is faithfully evaluated with `CompHornerIsFaithful`. We also see that the dynamic test for faithful rounding (Proposition 13) succeeds for condition numbers larger than the a priori bound (4.1)—let us recall that all the compensated evaluations proved to be faithfully rounded thanks to the dynamic test are reported with a square. Finally we notice that the compensated Horner algorithm produces accurate evaluations for condition numbers up to about $1/\mathbf{u}$—evaluations reported with a square or a filled circle.

Now we illustrate the significance of the dynamic error bound (4.3), compared to the a priori absolute error bound (3.5) and to the actual forward error. We evaluate the expanded form of $p(x) = (1-x)^5$ for 400 points around $x = 1$. For

every value of the entry x, we compute $\texttt{CompHorner}(p, x)$, the associated dynamic error bound (4.3) and the actual forward error. The results are reported on Fig. 3.

As already noticed, the closer the argument is to the root 1 (i.e., the more the condition number increases), the more pessimistic becomes the a priori error bound. Our dynamic error bound is more significant than the a priori error bound as it takes into account the rounding errors that occur during the computation.

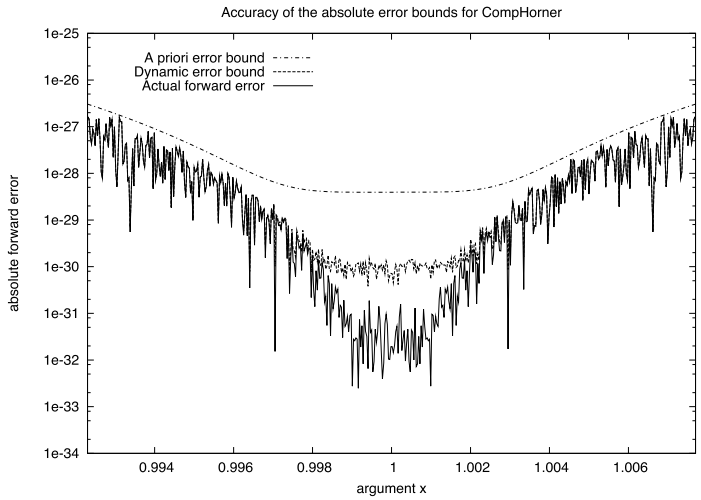


Fig. 3. Significance of the absolute error bounds.

5. A recursive EFT for polynomial evaluation

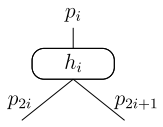
In the sequel of the paper, p_1 is a polynomial of degree n with floating point coefficients, and x is a floating point value. Given an integer $K \geq 2$, we now define a new EFT for polynomial evaluation. The starting point is Relation (3.3) that proves the forward error within the Horner algorithm is the sum of two polynomials with floating point coefficients. So the principle of this EFT is to apply algorithm `EFTHorner` (Algorithm 6) recursively $K - 1$ times.

5.1. Recursive application of `EFTHorner`

Further developments will be easier to read introducing a graphical representation of one application of the `EFTHorner` transformation (Algorithm 6). Given p_i a polynomial of degree d with floating point coefficients and x a floating point number, we consider the floating point value h_i and the polynomials p_{2i} and p_{2i+1} of degree at most $d-1$ such that $[h_i, p_{2i}, p_{2i+1}] = \texttt{EFTHorner}(p_i, x)$. From Theorem 7, we have $h_i = \texttt{Horner}(p_i, x)$ and

$$p_i(x) = h_i + (p_{2i} + p_{2i+1})(x).$$

We represent this EFT of $p_i(x)$ with the following cell where edges are polynomials (one entry and two outputs) and the node is a floating point value.



Now we describe the principle of the `EFTHornerK` algorithm as the binary tree of depth K represented with Fig. 4. For levels 1 to $K-1$, we recursively apply `EFTHorner`. At the last level K this gives 2^{K-1} polynomials here represented as rectangles.

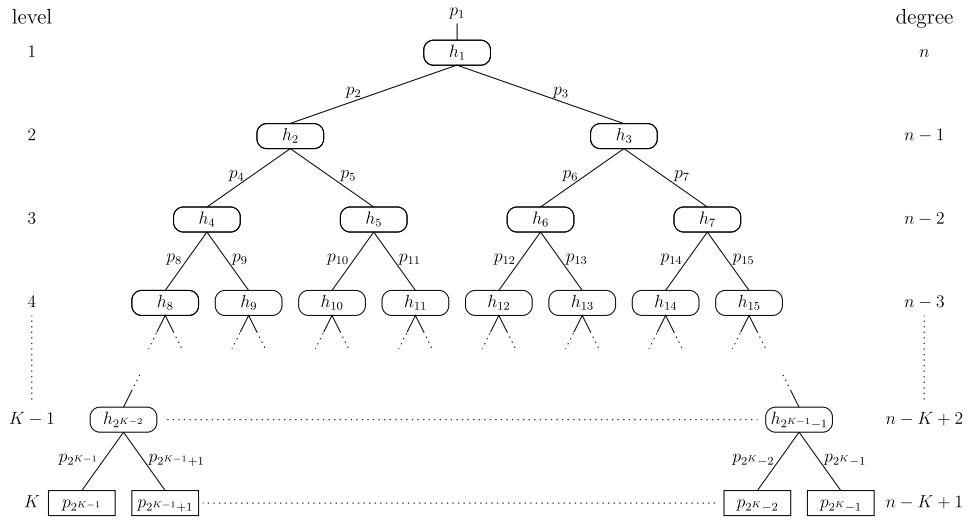


Fig. 4. Representation of `EFTHornerK` as a binary tree.

When `EFTHorner` is applied to a polynomial of degree d then the two generated polynomials are of degree $d-1$. Since p_1 is of degree n and `EFTHorner` is applied to $K-1$ levels, the polynomials computed on the level K are of degree at most $n-K+1$. In particular, if $n-K+1=0$ then the polynomials computed at the leaves of the binary tree are constants and so it is useless to apply again `EFTHorner`. Therefore, to simplify the discussion we will always assume $2 \leq K \leq n+1$ in the sequel.

To easily identify the nodes in this binary tree, we define the following sets of index.

- $\mathcal{N}_T^K = \{1, \ldots, 2^K - 1\}$ is the set of all the nodes of the tree, and $\operatorname{card}\big(\mathcal{N}_T^K\big) = 2^K - 1$;
- $\mathcal{N}_I^K = \{1, \ldots, 2^{K-1} - 1\}$ is the set of the internal nodes, and $\operatorname{card}\big(\mathcal{N}_I^K\big) = 2^{K-1} - 1$;
- $\mathcal{N}_L^K = \{2^{K-1}, \ldots, 2^K - 1\}$ is the set of the leaves, and $\operatorname{card}\big(\mathcal{N}_L^K\big) = 2^{K-1}$.

In particular we have $\mathcal{N}_T = \mathcal{N}_I \cup \mathcal{N}_L$ and $\mathcal{N}_I \cap \mathcal{N}_L = \emptyset$. We avoid the exponent K in the set notations except when necessary. The recursive application of `EFTHorner` to $K-1$ levels is then defined by

$$[h_i, p_{2i}, p_{2i+1}] = \texttt{EFTHorner}(p_i, x), \quad \text{for } i \in \mathcal{N}_I, \tag{5.1}$$

with $h_i \in \mathbb{F}$ for $i \in \mathcal{N}_I$ and p_i being a polynomial with floating point coefficients for every $i \in \mathcal{N}_T$. According to Theorem 7, every h_i defined by the previous relation is the evaluation of the polynomial p_i at x by the Horner algorithm, i.e.,

$$h_i = \texttt{Horner}(p_i, x), \quad \text{for } i \in \mathcal{N}_I.$$

Since `EFTHorner` is an EFT for the Horner algorithm, Theorem 7 also yields

$$p_i(x) = h_i + (p_{2i} + p_{2i+1})(x), \quad \text{for } i \in \mathcal{N}_I. \tag{5.2}$$

The floating point values $h_{i\in\mathcal{N}_I}$ and the polynomials $p_{i\in\mathcal{N}_L}$ are computed thanks to the next `EFTHornerK` algorithm.

ALGORITHM 15. *Recursive application of* `EFTHorner` *to* $K-1$ *levels.*

function $[h_{i\in\mathcal{N}_I}, p_{i\in\mathcal{N}_L}] = \texttt{EFTHornerK}(p_1, x)$
  for $i \in \mathcal{N}_I$, $[h_i, p_{2i}, p_{2i+1}] = \texttt{EFTHorner}(p_i, x)$

5.2. Numerical properties of `EFTHornerK`

First we prove that `EFTHornerK` (Algorithm 15) is actually an EFT for the evaluation of $p_1(x)$.

THEOREM 16. *Given an integer* K *with* $2 \le K \le n+1$*, we consider the floating point numbers* $h_{i\in\mathcal{N}_I}$ *and the polynomials* $p_{i\in\mathcal{N}_L}$*, such that* $[h_{i\in\mathcal{N}_I}, p_{i\in\mathcal{N}_L}] =$ `EFTHornerK`(p_1, x) *(*Algorithm 15*). The following relation holds:*

$$p_1(x) = \sum_{i\in\mathcal{N}_I} h_i + \sum_{i\in\mathcal{N}_L} p_i(x). \tag{5.3}$$

Algorithm `EFTHornerK` computes the evaluation $h_i = \texttt{Horner}(p_i, x)$ of every polynomial p_i, for $i \in \mathcal{N}_I$. For the proof of Theorem 16, we also need to consider the evaluation of the polynomials $p_i(x)$, for $i \in \mathcal{N}_L$. So let us also denote $h_i = \texttt{Horner}(p_i, x)$, for $i \in \mathcal{N}_L$.

Proof (see [14] *for details).* We proceed by induction on K. For $K = 2$, according to Theorem 7 we have $p_1(x) = \texttt{Horner}(p_1, x) + (p_2 + p_3)(x) = h_1 + p_2(x) + p_3(x)$, and therefore $[h_1, p_2, p_3] = \texttt{EFTHorner}(p_1, x)$. The induction step is easy to derive using Relations (5.1) and (5.2). □

PROPOSITION 17. *With the same hypothesis as in* Theorem 16*, the following relations hold:*

$$\left| p_1(x) - \sum_{i\in\mathcal{N}_T} h_i \right| = \left| \sum_{i\in\mathcal{N}_L} p_i(x) - h_i \right| \le \gamma_{2(n-K+1)} \gamma_{4n}^{K-1} \widetilde{p_1}(x). \tag{5.4}$$

The error generated when approximating $p_1(x)$ by the exact sum $\sum_{i\in\mathcal{N}_T} h_i$ is therefore equal to the sum of the errors generated when approximating every $p_i(x)$ by $h_i = \texttt{Horner}(p_i, x)$, for $i \in \mathcal{N}_L$. The previous proposition also provides an a priori bound on this error with respect to $\widetilde{p_1}(x) = \sum_{i=0}^{n} |a_i|\,|x|^i$. See [14] for the proof of Proposition 17.

Our approach is motivated by Inequality (5.4). This inequality shows that when the parameter K is incremented by one, the distance between $p_1(x)$ and the exact sum $\sum_{i\in\mathcal{N}_T} h_i$ decreases by a factor γ_{4n}, that corresponds roughly to an accuracy improved by a factor $4n\mathbf{u}$.

6. Algorithm `CompHornerK`

Now we formulate the algorithm `CompHornerK` (Algorithm 19), and then we provide an a priori error bound for this recursive compensated evaluation.

6.1. Principle of the algorithm

As before, the floating point values $h_{i\in\mathcal{N}_T}$ are defined according to the relations $[h_{i\in\mathcal{N}_I}, p_{i\in\mathcal{N}_L}] = \texttt{EFTHornerK}(p_1, x)$, and $h_i = \texttt{Horner}(p_i, x)$ for $i \in \mathcal{N}_L$. Then Inequality (5.4) shows that

$$\left| p_1(x) - \sum_{i\in\mathcal{N}_T} h_i \right| \leq (4n\mathbf{u})^K \widetilde{p_1}(x) + \mathcal{O}(\mathbf{u}^{K+1}). \tag{6.1}$$

The principle of the `CompHornerK` algorithm is to compute $\texttt{CompHornerK}(p, x, K)$, an approximate of $\sum_{i\in\mathcal{N}_T} h_i$ in K times the working precision, so that

$$\begin{aligned} &\frac{|\texttt{CompHornerK}(p, x, K) - p_1(x)|}{|p_1(x)|} \\ &\leq (\mathbf{u} + \mathcal{O}(\mathbf{u}^2)) + ((4n\mathbf{u})^K + \mathcal{O}(\mathbf{u}^{K+1}))\,\mathrm{cond}(p_1, x). \end{aligned} \tag{6.2}$$

In this inequality the term $(4n\mathbf{u})^K + \mathcal{O}(\mathbf{u}^{K+1})$ reflects that the intermediate computation is as accurate as if performed in precision $\mathbf{u}^K$. The first term $\mathbf{u} + \mathcal{O}(\mathbf{u}^2)$ reflects the final rounding of the result to the working precision $\mathbf{u}$. This accuracy bound corresponds to the introductory Relation (1.3). For the final computation of $\sum_{i\in\mathcal{N}_T} h_i$, we use the `SumK` algorithm proposed by Ogita, Rump and Oishi in [18]. This algorithm allows us to compute an approximate value of $\sum_{i\in\mathcal{N}_T} h_i$ with the same accuracy as if it was computed in K times the working precision. The following theorem summarizes the properties of algorithm `SumK`.

THEOREM 18 (Proposition 4.10 in [18]). *Given a vector $z = (z_1, \ldots, z_n)$ of n floating point values, let us define $s = \sum_{i=1}^{n} z_i$ and $\tilde{s} = \sum_{i=1}^{n} |z_i|$. We assume $4n\mathbf{u} < 1$ and we denote by $\bar{s}$ the floating point number such that $\bar{s} = \texttt{SumK}(z, K)$. Then, even in presence of underflow,*

$$|\bar{s} - s| \leq \left(\mathbf{u} + 3\gamma_{n-1}^2\right)|s| + \gamma_{2n-2}^K \tilde{s}. \tag{6.3}$$

6.2. `CompHornerK` and its a priori error bound

Now we formulate our compensated algorithm `CompHornerK`. We prove that it is as accurate as the Horner algorithm performed in K times the working precision.

ALGORITHM 19. *Compensated Horner algorithm providing K times the working precision.*

---

function $\bar{r} = \texttt{CompHornerK}(p_1, x, K)$
  $[h_{i\in\mathcal{N}_I}, p_{i\in\mathcal{N}_L}] = \texttt{EFTHornerK}(p_1, x)$
  for $i \in \mathcal{N}_L$, $h_i = \texttt{Horner}(p_i, x)$
  $\bar{r} = \texttt{SumK}(h_{i\in\mathcal{N}_T}, K)$

In algorithm `EFTHornerK`, polynomials $p_{i\in\mathcal{N}_I}$ are of degree at most n. Applying `EFTHorner` to each of these polynomials requires $\mathcal{O}(n)$ flop. Since $\mathrm{card}(\mathcal{N}_I) = 2^{K-1} - 1$, the cost of `EFTHornerK` is therefore $\mathcal{O}(n2^K)$ flop. In `CompHornerK`, the evaluation of the 2^{K-1} polynomials $p_{i\in\mathcal{N}_L}$ also requires $\mathcal{O}(n2^K)$ flop. Finally, $\texttt{SumK}(h_{i\in\mathcal{N}_T}, K)$ involves $(6K-5)(2^{K-1}-1) = \mathcal{O}(n2^K)$ flop. Overall, the cost of algorithm `CompHornerK` is therefore $\mathcal{O}(n2^K)$ flop. Of course this exponential complexity does not reduce the practical efficiency of the proposed algorithm while $K \leq 4$.

Next theorem gives an a priori bound for the forward error in `CompHornerK`.

THEOREM 20. *Let K be an integer such that $2 \leq K \leq n+1$. We assume $(2^K-2)\gamma_{2n+1} \leq 1$ and $2n(2n+1)\mathbf{u} < 1$. Then the forward error in the compensated evaluation of $p_1(x)$ with $\bar{r} = \texttt{CompHornerK}(p_1, x)$ (*Algorithm 19*) is bounded as follows:*

$$\begin{aligned} |\bar{r} - p_1(x)| \leq\ & \big(\mathbf{u} + 3\gamma_{2^K-2}^2 + \gamma_{2^{K+1}-4}^K\big)|p_1(x)| \\ & + \big(\gamma_{4n}^K + \gamma_{2n+1}\gamma_{2^{K+1}-4}^K + \gamma_{4n}^{K+1}\big)\widetilde{p_1}(x). \end{aligned} \tag{6.4}$$

For the proof of Theorem 20, we use the following lemma to bound the absolute condition number for the final summation of the floating point numbers $h_{i\in\mathcal{N}_T}$.

LEMMA 21. *With the same notations as in* Algorithm 19*, assuming $2n(2n+1)\mathbf{u} < 1$ and $(2^K-2)\gamma_{2n+1} \leq 1$, we have $\sum_{i\in\mathcal{N}_T}|h_i| \leq |p_1(x)| + \gamma_{4n}\widetilde{p_1}(x)$.*

Proof. We decompose the sum as follows, $\sum_{i\in\mathcal{N}_T}|h_i| = |h_1| + \sum_{i\in\mathcal{N}_T-\{1\}}|h_i|$. Since $h_1 = \texttt{Horner}(p_1, x)$, we have $|h_1| \leq |p_1(x)| + \gamma_{2n}\widetilde{p_1}(x)$. Moreover, for $i \in \mathcal{N}_T - \{1\}$ we also have $h_i = \texttt{Horner}(p_i, x)$ with p_i of degree at most $n-1$ and $\widetilde{p_i}(x) \leq \gamma_{2n}\widetilde{p_1}(x)$, thus

$$\begin{aligned} |h_i| &\leq |p_i(x)| + \gamma_{2(n-1)}\widetilde{p_i}(x) \leq (1+\gamma_{2(n-1)})\widetilde{p_i}(x) \\ &\leq (1+\gamma_{2(n-1)})\gamma_{2n}\widetilde{p_1}(x) \leq \gamma_{2n+1}\widetilde{p_1}(x). \end{aligned}$$

Therefore $\sum_{i\in\mathcal{N}_T}|h_i| \leq |p_1(x)| + \gamma_{2n}(1+(2^K-2)\gamma_{2n+1})\widetilde{p_1}(x)$. By assumption $(2^K-2)\gamma_{2n+1} \leq 1$, so that $\gamma_{2n}(1+(2^K-2)\gamma_{2n+1}) \leq 2\gamma_{2n} \leq \gamma_{4n}$, which proves the lemma. □

Proof of Theorem 20. Defining $e_1 := \left|p_1(x) - \sum_{i\in\mathcal{N}_T} h_i\right|$ and $e_2 := \left|\sum_{i\in\mathcal{N}_T} h_i - \bar{r}\right|$, we have $|\bar{r} - p_1(x)| \le e_1 + e_2$. According to Proposition 17 it follows that $e_1 \le \gamma_{2n}\gamma_{4n}^{K-1}\widetilde{p_1}(x)$. The second term e_2 denotes the error occurring in the final summation with algorithm `SumK`. Using the error bound (6.3), we deduce $e_2 \le \left(\mathbf{u} + 3\gamma_{2^K-2}^2\right)|s| + \gamma_{2^{K+1}-4}^K \tilde{s}$, with $s = \sum_{i\in\mathcal{N}_T} h_i$ and $\tilde{s} = \sum_{i\in\mathcal{N}_T} |h_i|$. Using Theorem 16 and Proposition 17, we have $|s| \le |p_1(x)| + \gamma_{2n}\gamma_{4n}^{K-1}\widetilde{p_1}(x)$. On the other hand $\tilde{s}$ is bounded according to Lemma 21. Thus we write

$$\begin{aligned} e_2 &\le \left(\mathbf{u} + 3\gamma_{2^K-2}^2\right)\left(|p_1(x)| + \gamma_{2n}\gamma_{4n}^{K-1}\widetilde{p_1}(x)\right) + \gamma_{2^{K+1}-4}^K(|p_1(x)| + \gamma_{4n}\widetilde{p_1}(x)) \\ &\le \left(\mathbf{u} + 3\gamma_{2^K-2}^2 + \gamma_{2^{K+1}-4}^K\right)|p_1(x)| + \left(\gamma_{4n}\gamma_{2^{K+1}-4}^K + \gamma_{4n}^{K+1}\right)\widetilde{p_1}(x). \end{aligned}$$

Therefore we have the inequality

$$\begin{aligned} |\bar{r} - p_1(x)| \le\ & \left(\mathbf{u} + 3\gamma_{2^K-2}^2 + \gamma_{2^{K+1}-4}^K\right)|p_1(x)| \\ & + \left(\gamma_{2n}\gamma_{4n}^{K-1} + \gamma_{4n}\gamma_{2^{K+1}-4}^K + \gamma_{4n}^{K+1}\right)\widetilde{p_1}(x), \end{aligned}$$

which proves Theorem 20. □

6.3. Numerical behavior of `CompHornerK`

To exhibit the numerical behavior of `CompHornerK` (Algorithm 19) with respect to the condition number we have generated 700 polynomials of degree 25 with condition number ranging from 10^2 to 10^{100} and coefficients being double precision values [12].

We report with Fig. 5 the relative accuracy of every polynomial evaluation performed with `Horner` and `CompHornerK` for successive iterates $K = 2, 3, 4$, compared to the condition number $\mathrm{cond}(p, x)$. We also represent the a priori relative error bounds (1.1) and (6.2).

As expected from the previous error bounds, the algorithm `CompHornerK` is in practice as accurate as the Horner algorithm performed in K times the working precision with a final rounding to the working precision. For every considered value of K (more than those here represented have been tested) the relative accuracy of the compensated evaluation is of the order of the working precision $\mathbf{u}$ as long as $\mathrm{cond}(p, x)$ is smaller than $\mathbf{u}^{-K}$. Of course `CompHornerK` exhibits the same numerical behavior as `CompHorner` when $K = 2$.

We also observe that the a priori bound (6.2) of the relative error in the computed evaluation is always pessimistic compared to the actual (measured) error by many orders of magnitude. Moreover this error bound is more and more pessimistic when the parameter K increases—this phenomenon is also observed in [18] for the compensated dot product algorithm `DotK`.

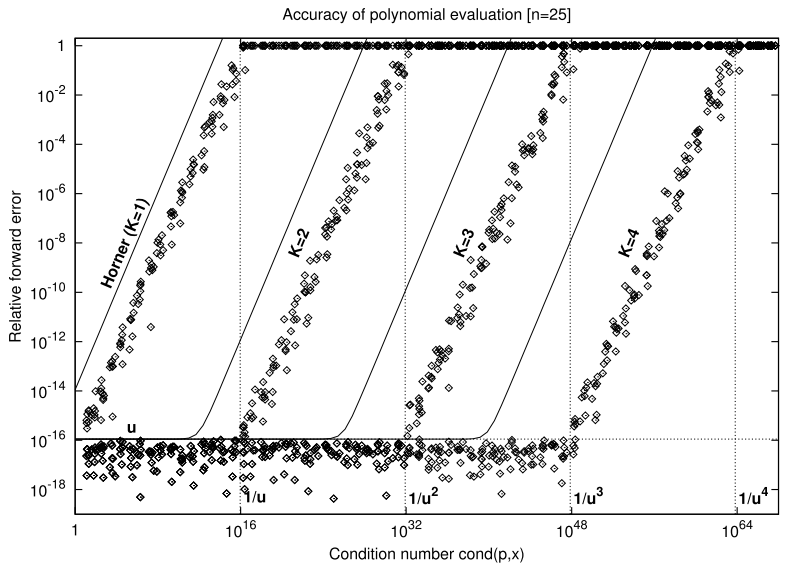


Fig. 5. Relative accuracy of `Horner` and of the compensated evaluations `CompHornerK` (Algorithm 19) with respect to $\mathrm{cond}(p, x)$, for $K = 2, 3, 4$. A priori error bounds are represented as continuous lines.

7. Running time performances

In this last section we demonstrate the practical efficiency in terms of running time comparing our algorithms and up-to-date challengers on several significant computing environments.

Let us first emphasize that the running time of these algorithms does not depend on the coefficients of the polynomial, nor on the argument x, but only on the degree n. All experiments are performed using IEEE-754 double precision with the following environments.

(I) Intel Pentium 4, 3.0 GHz, GNU Compiler Collection 4.1.2, fpu x87;
(II) AMD Athlon 64, 2.0 GHz, GNU Compiler Collection 4.1.2, fpu sse;
(III) Itanium 2, 1.5 GHz, GNU Compiler Collection 4.1.1;
(IV) Itanium 2, 1.5 GHz, Intel C Compiler 9.1.

We consider separately the experiments with `CompHorner` and `CompHornerK`.

7.1. `CompHorner` runs at least twice as fast as double-double Horner

Since double-doubles [1] are usually considered as the most efficient portable library to double the IEEE-754 double precision, we consider it as a reference in the comparisons with `CompHorner`. For our purpose, it suffices to know that a double-double number a is the pair (a_h, a_l) of IEEE-754 floating point numbers with $a = a_h + a_l$ and $|a_l| \leq \mathbf{u}|a_h|$. This property implies a renormalisation step after each arithmetic operation on double-double values. We denote by `DDHorner` our implementation of the Horner algorithm with the double-double format, derived from the implementation proposed by the authors of [15].

We implement the algorithms `CompHorner`, `CompHornerIsFaithful` and `DDHorner` in a C code to measure their overhead compared to the `Horner` algorithm. We program these tests straightforwardly with no other optimization than the ones performed by the compiler. All timings are done with the cache warmed to minimize the memory traffic over-cost. The measures are performed with polynomials whose degree vary from 5 to 200 by step of 5 (conditionning does not affect these tests). For every algorithm, we measure the ratio of its computing time over the computing time of the Horner algorithm; we display the average time ratio over all test cases in Table 2.

Table 2. Measured running time ratios to double the accuracy of the Horner algorithm.

			$\frac{\texttt{CompHorner}}{\texttt{Horner}}$	$\frac{\texttt{CompHornerIsFaithful}}{\texttt{Horner}}$	$\frac{\texttt{DDHorner}}{\texttt{Horner}}$
(I)	P4	gcc 4.1.2	2.8	3.5	8.6
(II)	AMD64	gcc 4.1.2	3.2	3.6	8.7
(III)	IA'64	gcc 4.1.1	2.8	3.4	6.7
(IV)		icc 9.1	1.5	1.7	5.9
			$\sim$ 2–3	$\sim$ 2–4	$\sim$ 5–9

The results in Table 2 show that the slowdown factor introduced by `CompHorner` compared to `Horner` roughly varies between 2 and 3. The same slowdown factor varies between 2 and 4 for `CompHornerIsFaithful` and between 5 and 9 for `DDHorner`. Therefore we can see that the over-cost due to the dynamic test for faithful rounding is quite reasonable. Anyway `CompHorner` and `CompHornerIsFaithful` run both significantly faster than `DDHorner`.

We provide time ratios for IA'64 architecture (Itanium 2). Tested algorithms take benefit from IA'64 instructions (as `FMA`) but are not described here—see [5] for details.

7.2. `CompHornerK` runs faster than challengers for $K \leq 4$.

First experiments study the performances of algorithm `CompHornerK` (Algorithm 19) assuming that K is an argument of the implemented routine. Since double precision is the working precision, `CompHornerK` simulates a precision of about $K \times 53$ bits. We compare `CompHornerK` to the Horner algorithm implemented with the MPFR library [16] using a precision of $K \times 53$ bits; we denote by `MPFRHornerK` this implementation. We use 39 random polynomials of degree 10 to 200, by step of 5. For every considered degree we measure the overhead introduced by the algorithms `CompHornerK` and `MPFRHornerK` compared to the classic Horner algorithm (we measure the ratio of the running time of `CompHornerK` over the running time of `Horner`, and we perform the same measurement for `MPFRHornerK`). We report the average overheads for both algorithms with respect to K on the left side of Fig. 6.

`CompHornerK` is clearly not competitive compared to `MPFRHornerK` for large values of K. The exponential complexity with respect to K of `CompHornerK` we previously exhibit certainly justifies this limitation. Nevertheless, in our experiments `CompHornerK` runs always faster than `MPFRHornerK` while K is smaller

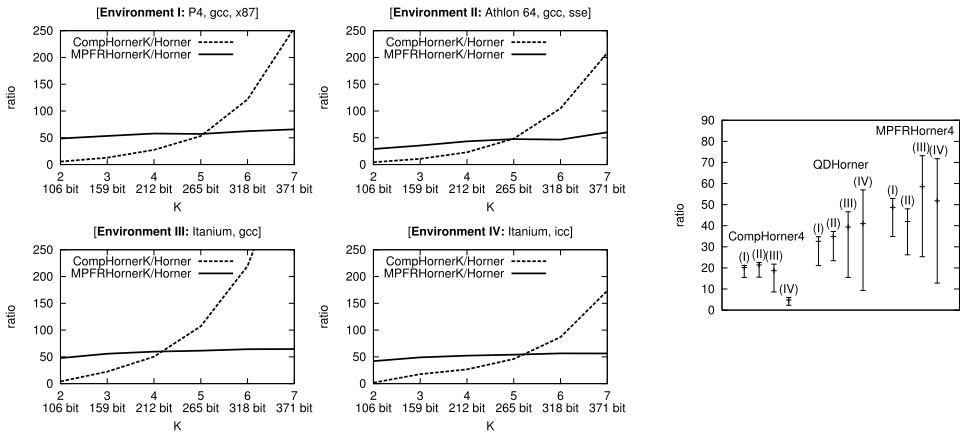


Fig. 6. Average measured running time ratios compared to `Horner` of `CompHornerK` and `MPFRHornerK` (left) and for `CompHorner4`, `QDHorner` and `MPFRHorner4` (right).

than 4. This illustrates the practical interest of `CompHorner` for simulating a small improvement of the working precision.

Next we study an optimized version of `CompHornerK` a priori setting a value for K. We name `CompHorner4` the corresponding implementation for $K = 4$. Setting the parameter K to a particular value does not change the principle of the algorithm but allows the compiler to perform more optimizations and provide better practical performances. We compare `CompHorner4` to `QDHorner`, the Horner algorithm implemented with quad-double arithmetic—that also simulates 4 times the IEEE-754 double precision [1]. For a fair comparison, our implementation of `QDHorner` inlines the quad-double arithmetic described in [6] and is also compiled with the same optimizing option as `CompHorner4`. We also compare `CompHorner4` to `MPFRHorner4` using the MPFR library with a working precision of 212 bits.

As before we use 39 random polynomials of degree varying from 10 to 200 by step of 5. For every polynomial, we measure the overhead of `CompHorner4` compared to the classic Horner algorithm. For every environment listed above we report the minimum, the average and the maximum values of this overhead on the right side of Fig. 6. We also report the same average overheads for `QDHorner` and `MPFRHorner4`.

Our `CompHorner4` is always significantly faster than both `QDHorner` and `MPFRHorner`. In particular `CompHorner4` runs about 8 times faster than `QDHorner` in the environment (IV) which is the Itanium architecture with the Intel compiler.

Acknowledgment. The authors thank T. Ogita, S.M. Rump and S. Oishi for their publication [18] that motivates and inspires this work and the referees for their interesting suggestions.

References

[1] High-Precision Software Directory. http://crd.lbl.gov/~dhbailey/mpdist.

[2] T.J. Dekker, A floating-point technique for extending the available precision. Numer. Math., **18** (1971), 224–242.

[3] J.W. Demmel, Applied Numerical Linear Algebra. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.

[4] S. Graillat, P. Langlois and N. Louvet, Compensated Horner scheme. Algebraic and Numerical Algorithms and Computer-Assisted Proofs, B. Buchberger, S. Oishi, M. Plum and S.M. Rump (eds.), Dagstuhl Seminar Proceedings, No. 05391, Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany, 2006.

[5] S. Graillat, P. Langlois and N. Louvet, Improving the compensated Horner scheme with a fused multiply and add. Proceedings of the 21st Annual ACM Symposium on Applied Computing, Vol. 2, Association for Computing Machinery, 2006, 1323–1327.

[6] Y. Hida, X.S. Li and D.H. Bailey, Quad-double arithmetic: Algorithms, implementation, and application. 15th IEEE Symposium on Computer Arithmetic, N. Burgess and L. Ciminiera (eds.), IEEE Computer Society, 2001, 155–162.

[7] N.J. Higham, Accuracy and Stability of Numerical Algorithms, 2nd edition. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002.

[8] C.M. Hoffmann, G. Park, J.-R. Simard and N.F. Stewart, Residual iteration and accurate polynomial evaluation for shape-interrogation applications. Proceedings of the 9th ACM Symposium on Solid Modeling and Applications, 2004, 9–14.

[9] IEEE Standards Committee 754, IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985. Institute of Electrical and Electronics Engineers, Los Alamitos, CA, USA, 1985, Reprinted in SIGPLAN Notices, **22** (1987), 9–25.

[10] D.E. Knuth, The Art of Computer Programming: Seminumerical Algorithms, Vol. 2, 3rd edition. Addison-Wesley, Reading, MA, USA, 1998.

[11] P. Langlois, More accuracy at fixed precision. J. Comp. Appl. Math., **162** (2004), 57–77.

[12] P. Langlois and N. Louvet, How to ensure a faithful polynomial evaluation with the compensated Horner algorithm. 18th IEEE International Symposium on Computer Arithmetic, P. Kornerup and J.-M. Muller (eds.), IEEE Computer Society, 2007, 141–149.

[13] P. Langlois and N. Louvet, More instruction level parallelism explains the actual efficiency of compensated algorithms. Technical Report hal-00165020, DALI Research Project, HAL-CCSD, 2007.

[14] P. Langlois and N. Louvet, Compensated Horner algorithm in K times the working precision. RNC-8, Real Numbers and Computer Conference, J. Brugera and M. Daumas (eds.), Santiago de Compostela, Spain, 2008.

[15] X.S. Li, J.W. Demmel, D.H. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, S.Y. Kang, A. Kapur, M.C. Martin, B.J. Thompson, T. Tung and D.J. Yoo, Design, implementation and testing of extended and mixed precision BLAS. ACM Trans. Math. Software, **28** (2002), 152–205.

[16] The MPFR Library (version 2.2.1). Available at http://www.mpfr.org.

[17] Y. Nievergelt, Scalar fused multiply-add instructions produce floating-point matrix arithmetic provably accurate to the penultimate digit. ACM Trans. Math. Software, **29** (2003), 27–48.

[18] T. Ogita, S.M. Rump and S. Oishi, Accurate sum and dot product. SIAM J. Sci. Comput., **26** (2005), 1955–1988.

[19] S.M. Rump, T. Ogita and S. Oishi, Accurate floating-point summation, Part I: Faithful rounding. SIAM J. Sci. Comput., **31** (2008), 189–224.