Compensated Horner Scheme

S. Graillat, Ph. Langlois, N. Louvet

stef.graillat@univ-perp.fr
philippe.langlois@univ-perp.fr
nicolas.louvet@univ-perp.fr

30 août 2005

Compensated Horner Scheme

S. Graillat, Ph. Langlois, N. Louvet

stef.graillat@univ-perp.fr
philippe.langlois@univ-perp.fr
nicolas.louvet@univ-perp.fr

30 août 2005

Abstract

We present a compensated Horner scheme, that is an accurate and fast algorithm to evaluate univariate polynomials in floating point arithmetic. The accuracy of the computed result is similar to the one given by the Horner scheme computed in twice the working precision. This compensated Horner scheme runs at least as fast as existing implementations producing the same output accuracy. We also propose to compute in pure floating point arithmetic a valid error estimate that bound the actual accuracy of the compensated evaluation. Numerical experiments involving ill-conditioned polynomials illustrate these results. All algorithms are performed at a given working precision and are portable assuming the floating point arithmetic satisfies the IEEE-754 standard.

Keywords: IEEE-754 floating point arithmetic, error-free transformations, extended precision, polynomial evaluation, compensated Horner scheme, running error bound

Résumé

Nous présentons un schéma de Horner compensé, c'est-à-dire un algorithme précis et rapide pour évaluer des polynômes univariés en arithmétique flottante. La précision du résultat ainsi obtenu est équivalente à celle fournie par le schéma de Horner exécuté en précision double de la précision courante. Ce schéma de Horner compensé s'exécute au moins aussi rapidement que les implémentations existantes qui permettent la même précision du résultat. Nous proposons aussi de calculer en arithmétique flottante une borne garantie de l'erreur effective de l'évaluation compensée. Des expérimentations numériques sur des polynômes trés mal conditionnés illustrent ces résultats. Tous ces algorithmes sont exécutés à une précision courante donnée et sont portables dès lors que les hypothèses de l'arithmétique flottante IEEE-754 sont satisfaites.

Mots-clés: arithmétique flottante IEEE-754, précision étendue, évaluation polynomiale, schéma de Horner compensé, borne d'erreur dynamique

AMS subject classifications: 65G, 65Y99

Contents

1 Introduction

Polynomials appear in many areas of scientific computing and engineering. Computer Aided Design and Modeling, Mechanical Systems Design, Signal Processing and Filter Design, Civil Engineering, Robotics, Simulation are for instance quoted in [6, 5]. Developing fast algorithms and reliable implementations of polynomial solvers are of challenging interest. Numerical approaches include iterative methods like Newton's method or homotopy continuation methods. These iterative methods needs to evaluate polynomials and their derivatives. Higham [10, chap. 5] devotes an entire chapter to polynomials and more especially to polynomial evaluation.

In this paper we present an accurate and fast algorithm to evaluate univariate polynomials in floating point arithmetic. By accurate, we mean that the accuracy of the computed result is similar to the one given by the Horner scheme computed in twice the working precision. By fast, we mean that the algorithm run at least as fast as existing counterparts that produce the same output accuracy. We also present how to compute in pure floating point arithmetic a valid error error bound to check the actual accuracy of the proposed polynomial evaluation. These algorithms are performed at a given working precision (no higher precision is necessary) and are portable assuming the floating point arithmetic satisfies the IEEE-754 standard.

Since we improve the Horner schema similarly as the well known Kahan's compensated summation method [12], the proposed evaluation algorithm is presented as a *compensated Horner scheme*. The recent accurate sum and dot product algorithms by Ogita-Rump-Oishi [20] strongly motivates this paper (the proofs of presented error bounds use techniques introduced in this latter reference).

1.1 Numerical Polynomial Evaluation

The classic Horner scheme is the optimal algorithm with respect to algebraic complexity for evaluating a polynomial p with given coefficients in the monomial basis. Horner scheme is often provided by numerical and scientific libraries, *e.g.* SPOLY and DPOLY in IBM ESSL, gsl_poly_eval in GNU GSL, ... The small backward error the Horner scheme introduce when computing in finite precision justifies its practical interest in floating point arithmetic for instance. It is well known that the computed evaluation of $p(x)$ is the exact value at x of a polynomial obtained by making relative perturbations of at most size $2n\,\mathbf{u}$ to the coefficients of p where n denotes the polynomial degree and $\mathbf{u}$ the finite precision of the computation [10].

The relative accuracy of the computed evaluation $\widehat{p}(x)$ with the Horner scheme verifies the classic rule of thumb that links the forward error to the condition number and the backward error. The classic condition number of the evaluation of $p(x) = \sum_{i=0}^{n} a_i x^i$ at a given data x is

$$\mathrm{cond}(p, x) = \frac{\sum_{i=0}^{n} |a_i||x|^i}{|\sum_{i=0}^{n} a_i x^i|} = \frac{\widetilde{p}(x)}{|p(x)|}. \tag{1}$$

The classic rule of thumb tells us that the accuracy of computed $\widehat{p}(x)$ is bounded as

$$\frac{|p(x) - \widehat{p}(x)|}{|p(x)|} \leq \alpha(n)\,\mathbf{u} \times \mathrm{cond}(p, x),$$

where $\alpha(n)$ is a (reasonable) linear function of the polynomial degree n (here $\alpha(n) \approx 2n$). Of course, the computed result $\widehat{p}(x)$ can be arbitrary less accurate than the working precision $\mathbf{u}$ when evaluating $p(x)$ is ill-conditioned. This is the case for example in the neighborhood of

multiple roots where all the digits or even the order of the computed value of $p(x)$ could be false.

How can we accurately perform an ill-conditioned polynomial evaluation? Before describing the main existing tools, let us distinguish two levels of (polynomial) ill-condition. When the computing precision is $\mathbf{u}$, evaluating $p(x)$ is ill-conditioned when $1 \ll \mathrm{cond}(p, x) \leq 1/\mathbf{u}$. There is no sense to consider (arbitrary) more ill-conditioned polynomials, *i.e.*, polynomials such that $\mathrm{cond}(p, x) > 1/\mathbf{u}$, except for example when the coefficients are exact in precision $\mathbf{u}$. In the following, we consider and define as ill-conditioned polynomials both ill-conditioned polynomials and arbitrary ill-conditioned polynomials assuming the latter satisfy this kind of necessary condition for significance.

Numerous multiprecision libraries are available when the computing precision $\mathbf{u}$ is not sufficient to guarantee a prescribed accuracy. Fixed-length expansions such as "double-double" or "quad-double" libraries [9] are actual and effective solutions to simulate twice or four times the IEEE-754 double precision [11]. For example a quad-double number is an unevaluated sum of four IEEE-754 double precision numbers and its associated arithmetic provides at least 212 bits of significand. These fixed-length expansions are currently embedded in major developments such for example within the new extended and mixed precision BLAS [16]. In this context the natural way to improve the accuracy of a given subroutine is to perform its internal computation within the extended precision these libraries provide and to return a result rounded to the (external) working precision. A more detailed discussion and references about existing implementations of expansions are for example available in [20].

1.2 The compensated Horner scheme improves the classic rule of thumb for the computed solution accuracy

We focus on the Horner scheme and constraint all computations to be performed in a fixed precision, for instance the IEEE-754 double precision. In this paper and in its companion paper [7] we present an alternative strategy to the fixed-length expansion libraries. As mentioned before, we propose accurate and fast algorithms to evaluate univariate polynomials in floating point arithmetic. By accurate, we mean that the accuracy of the computed result is similar to the one given by the Horner scheme computed in higher precision, for example using fixed-length expansions. This higher precision corresponds to twice the working precision for the proposed compensated Horner scheme. In the companion paper [7], we generalize this compensated Horner scheme such that this higher precision is k-fold the working precision. Of course the accuracy of the result still depends on the condition number $\mathrm{cond}(p, x)$. By fast, we mean that these algorithms run at least as fast as the fixed-length expansion challenger that produce the same output accuracy. Here the corresponding fixed-length expansions are the double-double format [1].

The proposed compensated Horner scheme requires no branch nor access to the mantissa, and uses only one working precision. We prove that the computed result res by the compensated Horner scheme is as accurate as if computed in doubled working precision. This means that the accuracy of r now satisfies a "compensated rule of thumb" being of the following form,

$$\frac{|\mathsf{res} - p(x)|}{|p(x)|} \leq \mathbf{u} + \beta(n)\,\mathbf{u}^2 \times \mathrm{cond}(p, x). \tag{2}$$

Again $\beta(n)$ is a (reasonable) linear function of the polynomial degree n and $\mathbf{u}$ is the precision of the computation. Such a result is given in next Corollary 6.

3

The second summand in the right hand side of Relation (2) reflects the accuracy of a backward stable computation performed in doubled working precision $\mathbf{u}^2$. The first one comes from the final rounding back to the working precision $\mathbf{u}$ and means that the accuracy improvement the second summand guarantee can not of course yield more accuracy than the available precision. Hence we can first expect an evaluation as accurate as possible for polynomials such that $\text{cond}(p, x) \leq 1/(\beta(n)\,\mathbf{u}^2)$, and then for more ill-conditioned polynomials, an accuracy that satisfies the classic rule of thumb with doubled working precision $\mathbf{u}^2$. Next Figure 1 illustrates such a behavior.

We also provide a dynamic bound for the accuracy of the compensated Horner scheme. This bound is proved to be valid and computable in pure floating point arithmetic. Together with the compensated evaluation, this bound is useful to replace the classic couple Horner scheme and its associated running error bound (see [10, chap. 5] for instance) when more accuracy is necessary. This is the case for example when implementing good stopping criteria for the Newton's method in the neighborhood of ill-conditioned roots.

1.3   Using error-free transformations to provide more accuracy

These compensated algorithms reduce the effect in the Horner scheme of the rounding errors generated by the finite precision arithmetic. The key tool to introduce more accuracy is what Ogita, Rump and Oishi call *error-free transformations* in [20]: "it is for long known that the approximation error of a floating point operation is itself a floating point number". Let $\mathsf{fl}(x)$ denotes the rounding to the working precision evaluation of the real value x. It means that for two floating point numbers a and b, and $\circ$ an arithmetic operator in $\{+, -, \times\}$, it exists a floating point number e, computable with floating point operations, such that

$$a \circ b = \mathsf{fl}(a \circ b) + e.$$

We later detail the corresponding algorithms for the summation and the product from Knuth [13] and Dekker [3]. Other error-free transformations exist for the division and the Fused-Multiply-and-Add FMA operator [2] – $\mathsf{FMA}(a, b, c) = \mathsf{fl}(a \times b + c)$.

The accuracy improvement is actually a correction of the global rounding error $p(x) - \widehat{p}(x)$, where $\widehat{p}(x)$ is the result of the Horner scheme performed in working precision. Such a correction has also been experimented by Pichat for summation [21] and mentioned for the Horner scheme in [22].

Using Tienari [23] and Linnainmaa [17, 18] results about linearization error, Langlois developed a method and a software that computes the first order corrected version of any algorithm [14, 15]. He also proved that the final error of the Horner scheme is linear with respect to the generated errors and so could be corrected computing (exactly) the first order term of $p(x) - \widehat{p}(x)$. The generic correction in [14] relies on error-free transformations and algorithmic differentiation. It can be used as a tool to identify algorithms that could be improved by this first order correction. An efficient implementation of such a corrected algorithm is derived in-lining the computation of the correcting term in the original algorithm. Some algorithms we present hereafter (and in [7]) have been designed like this.

We have mentioned that our algorithms are at least as fast as the fixed-length expansion counterparts that produce the same output accuracy. The practical efficiency of algorithms derived from error-free transformations is emphasized in [20] and motivates this article. From a theoretical point of view, the computation of every step of the compensated algorithm is similar to the

corresponding "double-double" computation but without performing the renormalization algorithm required by the non-overlapping "double-double" representation. In term of measured computing time, our experiments show that the compensated Horner scheme is therefore more than twice faster than the Horner scheme with "double-double". See Table 3 at the end of this paper for such ratios.

1.4 Outline of the paper

The paper is organized as follows. We introduce the classic assumptions and notations for floating point arithmetic and error analysis in Section 2. In Section 3, we briefly review the algorithms for the error-free transformations of the summation and the product of two floating point numbers, and we introduce the error-free transformation of the Horner scheme. In Section 4, we describe our compensated algorithm for the polynomial evaluation, and we prove that the computed result is of the same accuracy as if computed in doubled working precision. We also provide a valid and computable error bound for the compensated Horner scheme. Numerical experiments for extremely ill-conditioned evaluations are presented in Section 5. We compare the compensated algorithm, in term of computing time, to other algorithms with the same output accuracy.

At the first reading, one who know Ogita-Rump-Oishi's paper [20] can jump to the EFT of the polynomial evaluation (Theorem 2) and then to the compensated Horner scheme (Algorithm 9), its accuracy (Theorem 5) and the associated dynamic bound (Theorem 8). As in [20] we provide extended results that take care of the possible underflow; proofs for this case are gather in Annex A.

2 Standard model of floating point arithmetic and the Horner scheme

2.1 Standard model

The notations used throughout the paper are presented hereafter. Most of them come from [10, chap. 2]. As in the previous section, $\mathsf{fl}\left(\cdot\right)$ denotes the result of a floating point computation, where all the operations inside the parenthesis are performed in the working precision. We also introduce the symbols $\oplus$, $\ominus$, $\otimes$ and $\oslash$, representing respectively the floating point addition, subtraction, multiplication and division (*e.g.*, $a \oplus b = \mathsf{fl}\left(a + b\right)$). We adopt MATLAB like notations for algorithms.

The presented results are valid for any IEEE-754 like floating point arithmetic [11], with round to the nearest. We constraint all the computations to be performed in one working precision. We assume that no overflow occur during the computations, but we take into account the gradual underflow. Let us denote $\mathbb{F}$ the set of the floating point numbers, and let

- $\mathbf{u}$ be the relative error unit,

- λ be the smallest positive normalized floating point number,

- $\mathbf{v} = \lambda\,\mathbf{u}$ be the underflow unit which is half the spacing between two consecutive subnormal numbers.

For IEEE-754 double precision with round to the nearest and gradual underflow, we have $\mathbf{u} = 2^{-53} \approx 1.11 \cdot 10^{-16}$, $\lambda = 2^{-1022} \approx 2.22 \cdot 10^{-308}$, and $\mathbf{v} = 2^{-1075} \approx 2.47 \cdot 10^{-324}$.

When no underflow occur, the following standard model describes the accuracy of every considered floating point computation. For a and b in $\mathbb{F}$ (or in $\mathbb{F}^*$ if necessary) and for $\circ$ in $\{+, -, \times, /\}$, the floating point evaluation $\mathsf{fl}(a \circ b)$ of $a \circ b$ is such that

$$\mathsf{fl}(a \circ b) = (a \circ b)(1 + \varepsilon_1) = (a \circ b)/(1 + \varepsilon_2), \tag{3}$$

with $|\varepsilon_1|, |\varepsilon_2| \leq \mathbf{u}$. Addition and subtraction are exact in case of underflow [8]. To deal with possible gradual underflow, the standard model is extended by replacing Relation (3) by the following one for multiplication and division [4]. For $\circ$ in $\{\times, /\}$, the floating point evaluation of $a \circ b$ is such that

$$\mathsf{fl}(a \circ b) = (a \circ b)(1 + \varepsilon_1) + \eta_1 = (a \circ b)/(1 + \varepsilon_2) + \eta_2, \tag{4}$$

with $|\varepsilon_1|, |\varepsilon_2| \leq \mathbf{u}$, $|\eta_1|, |\eta_2| \leq \mathbf{v}$, and $\varepsilon_1 \eta_1 = \varepsilon_2 \eta_2 = 0$ (at most one of ε_1 and η_1 or ε_2 and η_2 is nonzero).

Remark 1. Let a and b be two nonnegative floating point numbers, and let $\circ$ be in $\{+, \times\}$. From the standard model (3), it follows that

$$0 \leq a \circ b \leq (1 + u)\,\mathsf{fl}\,(a \circ b) \quad \text{and} \quad 0 \leq a \circ b \leq (1 - u)^{-1}\,\mathsf{fl}\,(a \circ b).$$

To deal with gradual underflow in the case of the product, Relation (4) yields

$$0 \leq a \times b \leq (1 + u)\,\mathsf{fl}\,(a \times b) + \mathbf{v} \quad \text{and} \quad 0 \leq a \times b \leq (1 - u)^{-1}\,\mathsf{fl}\,(a \times b) + \mathbf{v}.$$

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

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

The relative error counters verify $\langle j \rangle \langle k \rangle = \langle j \rangle / \langle k \rangle = \langle j + k \rangle$. The quantities γ_n are defined as usual to be

$$\gamma_n = \frac{n\,\mathbf{u}}{1 - n\,\mathbf{u}}.$$

When using γ_n, we always implicitly assume $n\,\mathbf{u} < 1$. When $\langle n \rangle$ denotes any error counter, there exists a quantity θ_n such that

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

Remark 2. Next relations (about γ_n) will be useful in the sequel. We verify the following inequalities for any positive integer n,

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

2.2 The Horner scheme

The Horner scheme is the classic method for evaluating a polynomial $p(x) = \sum_{i=0}^{n} a_i x^i$ (Algorithm 1). For any floating point value x we denote $\mathsf{Horner}\,(p, x)$ the result of the floating point evaluation of the polynomial p at x using the Horner scheme.

Algorithm 1. Horner scheme

```
function [r_0] = Horner (p, x)
r_n = a_n
for i = n − 1 : −1 : 0
    r_i = r_{i+1} ⊗ x ⊕ a_i
end
```

A forward error bound for the result of Algorithm 1 is (see [10, p.95])

$$|p(x) - \text{Horner}\,(p, x)\,| \le \gamma_{2n}\,\widetilde{p}(x), \tag{5}$$

where

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

So, the accuracy of the computed evaluation is linked to the condition number of the polynomial evaluation (1) satisfying the previously mentioned rule of thumb,

$$\frac{|p(x) - \text{Horner}\,(p, x)\,|}{|p(x)|} \le \gamma_{2n}\,\text{cond}(p, x). \tag{6}$$

Clearly, the condition number $\text{cond}(p, x)$ can be arbitrarily large. In particular, when $\text{cond}(p, x) > 1/\gamma_{2n}$, we cannot guarantee that the computed result $\text{Horner}\,(p, x)$ contains any correct digit.

If a FMA instruction is present on the architecture, then we can change the line $r_i = r_{i+1} \otimes x \oplus a_i$ in Algorithm 1 by $r_i = \text{FMA}\,(r_{i+1}, x, a_i)$. This enable to improve the previous error bound since we have now,

$$\frac{|p(x) - \text{Horner}\,(p, x)\,|}{|p(x)|} \le \gamma_n\,\text{cond}(p, x).$$

3 Error-free transformations (EFT)

In this section, we review well known results concerning the error-free transformations of the elementary floating point operations $+$, $-$ and $\times$. We also introduce a new EFT for the polynomial evaluation using the Horner scheme.

3.1 EFT for the elementary operations

Let $\circ$ be in $\{+, -, \times\}$, a and b be two floating point numbers, and $\widehat{x} = \text{fl}(a \circ b)$. The *elementary rounding error* in the computation of $\widehat{x}$ is

$$y = (a \circ b) - \text{fl}(a \circ b), \tag{7}$$

that is the difference between the exact result and the computed result of the operation. In particular, for $\circ$ in $\{+, -, \times\}$, the elementary rounding error y both belongs to $\mathbb{F}$, and is computable using only the operations defined within $\mathbb{F}$. Thus, for $\circ$ in $\{+, -, \times\}$, any pair of inputs (a, b) in $\mathbb{F}^2$ can be transformed into an output pair $(\widehat{x}, y)$ in $\mathbb{F}^2$ such that

$$a \circ b = \widehat{x} + y \quad \text{and} \quad \widehat{x} = \text{fl}(a \circ b).$$

Let us emphasize that this relation between these four floating point values relies on real operators and exact equality (*i.e.*, not on approximate floating point counterparts). Ogita *et al.* [20] call such a transformation an *error-free transformation* (EFT).

The EFT for the addition ($\circ = +$) is given by the well known TwoSum algorithm by Knuth [13]. TwoSum (Algorithm 2) requires 6 flops (floating point operations).

Algorithm 2. EFT of the sum of two floating point numbers.

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

If the two floating point inputs a and b are such $|a| \leq |b|$, then we can use the following algorithm FastTwoSum for the EFT of the addition. It satisfies the same properties as TwoSum but requires only 3 flops. Nevertheless, if we count absolute value and comparison as one flop, this algorithm requires 6 flops. In practice, FastTwoSum is up to 50 % slower than TwoSum du to the presence of branching.

Algorithm 3. EFT of the sum of two floating point numbers when $|a| \leq |b|$.

function $[x, y] = $ FastTwoSum (a, b)
  $x = a \oplus b$
  $y = b \ominus (x \ominus a)$

For the EFT of the product, we first need to split the input arguments into two parts. It is done using Algorithm 4 by Dekker [3]. If q is the number of bits of the mantissa, let $r = \lceil q/2 \rceil$. Algorithm 4 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 $r - 1 = 26$ bits. The trick is that one bit sign is used for the splitting.

Algorithm 4. Splitting of a floating point number into two parts.

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

Then, Algorithm 5 by Veltkamp (see [3]) can be used for the EFT of the product. This algorithm is commonly called TwoProduct and requires 17 flops.

Algorithm 5. EFT of the product of two floating point numbers.

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

8

The next theorem exhibits the main properties of TwoSum and TwoProd, even in presence of underflow.

Theorem 1 ([20]). *Let a, b in $\mathbb{F}$ and $x, y \in \mathbb{F}$ such that $[x, y] = $ TwoSum(a, b) (Algorithm 2). Then, also in the presence of underflow,*

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

Algorithm TwoSum requires 6 flops.
Let $a, b \in \mathbb{F}$ and $x, y \in \mathbb{F}$ such that $[x, y] = $ TwoProduct(a, b) (Algorithm 5). Then, if no underflow occurs,

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

and, in the presence of underflow,

$$a \times b = x + y + 5\eta, \quad x = a \otimes b, \quad |y| \leq \mathbf{u}|x| + 5\mathbf{v}, \quad |y| \leq \mathbf{u}|a \times b| + 5\mathbf{v} \quad \text{with} \quad |\eta| \leq \mathbf{v}.$$

Algorithm TwoProduct requires 17 flops.

TwoProduct can be rewritten in a very straightforward way for processors that provide a Fused-Multiply-and-Add operator (FMA), such as Intel Itanium or IBM PowerPC. For a, b and c in $\mathbb{F}$, FMA(a, b, c) is the exact result $a \times b + c$ rounded to the nearest floating point value. Thus $y = a \times b - a \otimes b = $ FMA$(a, b, -(a \otimes b))$ and TwoProduct can be replaced by following Algorithm 6 requiring only 2 flops.

Algorithm 6. EFT of the sum of two floating point numbers with a FMA.

function $[x, y] = $ TwoProductFMA(a, b)
  $x = a \otimes b$
  $y = $ FMA$(a, b, -x)$

We notice that algorithms TwoSum, TwoProduct and TwoProductFMA require only well optimizable floating point operations. They do not use branches, nor access to the mantissa that can be time consuming.

In the sequel of the paper, we assume that no FMA operation is used except in algorithm TwoProductFMA. Our goal is to design algorithms whose proofs are valid on any IEEE-754 compliant computer. All the flop counts reported in the sequel of the paper have been done under this assumption.

3.2 An EFT for the Horner scheme

We now propose an EFT for the polynomial evaluation with the Horner scheme.

Theorem 2. *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 Algorithm 7 computes both*

i) the floating point evaluation Horner(p, x) and

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

9

and we write
$$[\mathsf{Horner}\,(p,x)\,,p_\pi,p_\sigma] = \mathsf{EFTHorner}\,(p,x)\,.$$

Then, if no underflow occurs,
$$p(x) = \mathsf{Horner}\,(p,x) + (p_\pi + p_\sigma)(x), \tag{8}$$

and, in the presence of underflow,
$$p(x) = \mathsf{Horner}\,(p,x) + (p_\pi + p_\sigma)(x) + 5\sum_{i=0}^{n-1}\eta_i x^i, \quad \text{with} \quad |\eta_i| \le \mathbf{v}. \tag{9}$$

Algorithm 7 requires $23n$ flops. If $\mathsf{TwoProductFMA}$ is used instead of $\mathsf{TwoProduct}$, then the flops count drops to $8n$.

Algorithm 7. EFT for the Horner scheme

function $[\mathsf{Horner}\,(p,x)\,,p_\pi,p_\sigma] = \mathsf{EFTHorner}(p,x)$
$s_n = a_n$
for $i = n-1 : -1 : 0$
   $[p_i, \pi_i] = \mathsf{TwoProduct}(s_{i+1}, x)$
   $[s_i, \sigma_i] = \mathsf{TwoSum}(p_i, a_i)$
   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
$\mathsf{Horner}\,(p,x) = s_0$

If no underflow occurs during the computation, Relation (8) means that $\mathsf{EFTHorner}$ is an EFT for the polynomial evaluation with the Horner scheme. In the presence of underflow, we do not have an EFT anymore, but we still write $[\mathsf{Horner}\,(p,x)\,,p_\pi,p_\sigma] = \mathsf{EFTHorner}\,(p,x)$.

Proof of Theorem 2 (without underflow). Since $\mathsf{TwoProduct}$ and TwoSum are EFT from Theorem 1 it follows that $s_{i+1}x = p_i + \pi_i$ and $p_i + a_i = s_i + \sigma_i$. Thus we have
$$s_i = s_{i+1}x + a_i - \pi_i - \sigma_i, \quad \text{for} \quad i = 0, \dots, n-1.$$

Since $s_n = a_n$, the whole for loop yields
$$s_0 = \left[\sum_{i=0}^{n} a_i x^i\right] - \left[\sum_{i=0}^{n-1} \pi_i x^i\right] - \left[\sum_{i=0}^{n-1} \sigma_i x^i\right],$$

and $\mathsf{Horner}\,(p,x) = p(x) - (p_\pi + p_\sigma)(x)$. □

The following proposition is useful to prove the accuracy bound on compensated Horner scheme in next section. This result proves that the conditioning of the (polynomial) error evaluation is better by a factor of the precision $\mathbf{u}$ than the conditioning of the initial polynomial evaluation. This property justifies the interest to apply recursively such compensated accuracy improvement in [7].

Proposition 3. *Given $p(x) = \sum_{i=0}^{n} a_i x^i$ a polynomial of degree n with floating point coefficients, and x a floating point value. Let y be the floating point value, p_π and p_σ be the two polynomials of degree $n-1$, with floating point coefficients, such that $[y, p_\pi, p_\sigma] = \mathsf{EFTHorner}(p,x)$ (Algorithm 7). Then, if no underflow occurs,*
$$(\widetilde{p_\pi} + \widetilde{p_\sigma})(x) \le \gamma_{2n}\,\widetilde{p}(x),$$

and, in the presence of underflow,

$$(\widetilde{p_\pi} + \widetilde{p_\sigma})(x) \leq \gamma_{2n}\, \widetilde{p}(x) + (5 + \gamma_{2n})\, \mathbf{v} \sum_{i=0}^{n-1} |x^i|.$$

Proof (without underflow). Applying the standard model of floating point arithmetic (3), for $i = 1, \ldots, n$, the two computations in the loop of Algorithm 7 verify

$$|p_{n-i}| = |s_{n-i+1} \otimes x| \leq (1 + \mathbf{u})|s_{n-i+1}||x| \quad \text{and} \quad |s_{n-i}| = |p_{n-i} \oplus a_{n-i}| \leq (1 + \mathbf{u})(|p_{n-i}| + |a_{n-i}|).$$

Let us prove by induction that, for $i = 1, \ldots, n$,

$$|p_{n-i}| \quad \leq \quad (1 + \gamma_{2i-1}) \sum_{j=1}^{i} |a_{n-i+j}||x^j|, \quad \text{and} \qquad (10)$$

$$|s_{n-i}| \quad \leq \quad (1 + \gamma_{2i}) \sum_{j=0}^{i} |a_{n-i+j}||x^j|. \qquad (11)$$

For $i = 1$, since $s_n = a_n$ we have $|p_{n-1}| \leq (1 + \mathbf{u})|a_n||x| \leq (1 + \gamma_1)|a_n||x|$ and (10) is satisfied. On the other hand, $|s_{n-1}| \leq (1 + \mathbf{u})\left((1 + \gamma_1)|a_n||x| + |a_{n-1}|\right) \leq (1 + \gamma_2)\left(|a_n||x| + |a_{n-1}|\right)$, and (11) is also satisfied. Now we suppose that (10) and (11) are true for some integer i such that $1 \leq i < n$. Then we have,

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

From the induction hypothesis, we derive,

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

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

Therefore we have,

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

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

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

Relation (10) and Relation (11) are proved by induction. Thus, for $i = 1, \ldots, n$,

$$|p_{n-i}||x^{n-i}| \leq (1 + \gamma_{2i-1})\, \widetilde{p}(x) \quad \text{and} \quad |s_{n-i}||x^{n-i}| \leq (1 + \gamma_{2i})\, \widetilde{p}(x).$$

From Theorem 1, since TwoSum and TwoProd are EFT, for $i = 0, \cdots, n-1$, we have $|\pi_i| \leq \mathbf{u}|p_i|$ and $|\sigma_i| \leq \mathbf{u}|s_i|$. Therefore,

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

11

and we obtain

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

Since $2n \, \mathbf{u}(1 + \gamma_{2n}) = \gamma_{2n}$, we finally obtain $(\widetilde{p_\pi} + \widetilde{p_\sigma})(x) \le \gamma_{2n} \, \widetilde{p}(x)$. $\qquad\qquad\square$

The proof in case of underflow is presented in Annex A.

4 Compensated Horner scheme

From Theorem 2 the global forward error affecting the floating point evaluation of p at x according to the Horner scheme is

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

The coefficients of these polynomials are exactly computed by Algorithm 7, together with $\mathsf{Horner}\,(p, x)$. Indeed, if $[\mathsf{Horner}\,(p, x), p_\pi, p_\sigma] = \mathsf{EFTHorner}(p, x)$, then p_π and p_σ are two exactly representable polynomials such that $p(x) = \mathsf{Horner}\,(p, x) + (p_\pi + p_\sigma)(x)$. Therefore, the key of the algorithm proposed in this section is to compute an approximate of the global error $e(x)$ in working precision, and then to compute a corrected result

$$\mathsf{res} = \mathsf{Horner}\,(p, x) \oplus \mathsf{fl}(e(x)).$$

We say that $c = \mathsf{fl}(e(x))$ is a corrective term for $\mathsf{Horner}\,(p, x)$. The corrected result res is expected to be more accurate than the first result $\mathsf{Horner}\,(p, x)$ as proved in the sequel of the section.

4.1 Evaluation of the sum of two polynomials

Our aim is now to compute the corrective term $c = \mathsf{fl}\,((p_\pi + p_\sigma)(x))$. Two different ways to evaluate this sum of polynomials are

i) evaluate the polynomial whose coefficients are those of $p_\pi + p_\sigma$ rounded to the nearest floating point value,

ii) compute $\mathsf{Horner}\,(p_\pi, x)$ and $\mathsf{Horner}\,(p_\sigma, x)$ and then sum the two results.

In this subsection, we consider the first solution which is here better in term of computing time.

Let p and q be two polynomials with floating point coefficients, such that $p(x) = \sum_{i=0}^{n} a_i x^i$ and $q(x) = \sum_{i=0}^{n} b_i x^i$. The coefficients of $(p + q)(x) = \sum_{i=0}^{n}(a_i + b_i)x^i$ are not necessarily floating point numbers. We compute an approximate of $(p+q)(x)$ by evaluating the polynomial whose coefficients are those of $p + q$ rounded to the nearest floating point value. This process is described by Algorithm 8.

Algorithm 8. Evaluation of the sum of two polynomials.

function $[r_0] = \mathsf{HornerSum}\,(p, q, x)$
$r_n = a_n \oplus b_n$
for i=n-1:-1:0
$\quad r_i = r_{i+1} \otimes x \oplus (a_i \oplus b_i)$
end

Lemma 4. *Let us consider the floating point evaluation of $(p + q)(x)$ computed with* HornerSum (p, q, x) *(Algorithm 8). Then, in case no underflow occurs, the computed result satisfies the following forward error bound,*

$$|\mathsf{HornerSum}\,(p, q, x) - (p + q)(x)| \le \gamma_{2n+1}(\widetilde{p} + \widetilde{q})(x),$$

and, in the presence of underflow,

$$|\mathsf{HornerSum}\,(p, q, x) - (p + q)(x)| \le (\widetilde{p} + \widetilde{q})(x) + (1 + \gamma_{2n-1})\,\mathbf{v}\sum_{i=0}^{n-1}|x^i|.$$

Algorithm HornerSum *requires $3n + 1$ flops.*

Proof (without underflow). Considering Algorithm 8, we have $r_n = a_n \oplus b_n = (a_n + b_n)\langle 1\rangle$, and for $i = n - 1, \cdots, 0$,

$$r_i = r_{i+1} \otimes x \oplus (a_i \oplus b_i) = r_{i+1}x\langle 2\rangle + (a_i + b_i)\langle 2\rangle.$$

Therefore it can be proved by induction that

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

Thus there exist quantities $\theta_{2n+1}, \theta_{2n}, \cdots, \theta_1$, bounded according to $|\theta_i| \le \gamma_i$, such that

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

Since $r_0 = \mathsf{HornerSum}\,(p, q, x)$, we finally obtain

$$\left|\mathsf{res} - \sum_{i=0}^{n}(a_i + b_i)x^i\right| \le \gamma_{2n+1}\sum_{i=0}^{n}|a_i + b_i||x^i| \le \gamma_{2n+1}(\widetilde{p} + \widetilde{q})(x).$$

$\square$

The proof in case of underflow is presented in Annex A.

4.2 The compensated Horner scheme and its error bound

In the previous subsection, we have chosen algorithm HornerSum to compute an approximate of the evaluation of the sum of two polynomials at a given value. This algorithm is used with EFTHorner (Algorithm 7) to compute the corrective term for the polynomial evaluation with the Horner scheme.

Algorithm 9. Compensated Horner scheme

```
function [res] = CompensatedHorner (p, x)
[h, p_π, p_σ] = EFTHorner (p, x)
c = HornerSum (p_π, p_σ, x)
res = h ⊕ c
```

We prove hereafter that the result of a polynomial evaluation computed with this compensated Horner scheme (Algorithm 9) is as accurate as if computed by the classic Horner scheme using twice the working precision and then rounded to the working precision.

Theorem 5. *Given a polynomial $p = \sum_{i=0}^{n} a_i x^i$ of degree n with floating point coefficients, and x a floating point value. We consider the result* CompensatedHorner (p, x) *computed by Algorithm 9. Then, if no underflow occurs,*

$$|\mathsf{CompensatedHorner}\,(p, x) - p(x)| \leq \mathbf{u}|p(x)| + \gamma_{2n}^2 \, \widetilde{p}(x), \tag{12}$$

and, in the presence of underflow,

$$|\mathsf{CompensatedHorner}\,(p, x) - p(x)| \leq \mathbf{u}|p(x)| + \gamma_{2n}^2 \, \widetilde{p}(x) + K\,\mathbf{v} \sum_{i=0}^{n-1} |x^i|,$$

with $K \leq 7$. CompensatedHorner *requires $26n + 3$ flops. If* TwoProductFMA *is used instead of* TwoProduct*, then the flops count drops to $11n - 1$.*

Proof (without underflow). The absolute forward error generated by Algorithm 9 is

$$|\mathsf{res} - p(x)| = |(h \oplus c) - p(x)| = |(1 + \varepsilon)(h + c) - p(x)| \quad \text{with} \quad |\varepsilon| \leq \mathbf{u}.$$

Let $e(x) = (p_\pi + p_\sigma)(x)$. From Theorem 2 we have $h = \mathsf{Horner}\,(p, x) = p(x) - e(x)$, thus

$$
\begin{aligned}
|\mathsf{res} - p(x)| &= |(1 + \varepsilon)\,(p(x) - e(x) + c) - p(x)| \\
&\leq \mathbf{u}|p(x)| + (1 + \mathbf{u})|c - e(x)|.
\end{aligned}
$$

Since p_π and p_σ are two polynomials of degree $n - 1$, and $c = \mathsf{HornerSum}\,(p_\pi, p_\sigma, x)$, applying Lemma 4, we write

$$|c - e(x)| \leq \gamma_{2n-1}(\widetilde{p_\pi} + \widetilde{p_\sigma})(x).$$

Then we use Proposition 3 to bound $(\widetilde{p_\pi} + \widetilde{p_\sigma})(x)$ as

$$|c - e(x)| \leq \gamma_{2n-1}\gamma_{2n}\,\widetilde{p}(x).$$

Since $(1 + \mathbf{u})\gamma_{2n-1} \leq \gamma_{2n}$, we finally write the expected bound

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

$\square$

The proof in case of underflow is presented in Annex A.

It is very interesting to interpret the previous theorem in terms of the condition number(1) of the polynomial evaluation of p at x. Combining the error bound in Theorem 5 with the expression of the condition number $|\widetilde{p}(x)|/|p(x)|$ for the polynomial evaluation gives the following result.

Corollary 6. *Given p a polynomial of degree n with floating point coefficients, and x a floating point value. If no underflow occurs,*

$$\frac{|\mathsf{CompensatedHorner}\,(p, x) - p(x)|}{|p(x)|} \leq \mathbf{u} + \gamma_{2n}^2 \,\mathrm{cond}(p, x). \tag{13}$$

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 summand $\mathbf{u}$ for rounding back the result to the working precision. In particular, if $\mathrm{cond}(p, x) < \gamma_{2n}^{-1}$, then the relative accuracy of the result is bounded by a constant of the order $\mathbf{u}$. This means that the compensated Horner scheme 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, Corollary 6 tells us that the computed result is as accurate as if computed by the classic Horner scheme with twice the working precision $\mathbf{u}^2$ and then rounded to the working precision $\mathbf{u}$.

4.3 A dynamic error bound

The error bound (12) for the result of a polynomial evaluation with algorithm Compensated-Horner is entirely adequate for theoretical purposes. However, it is an *a priori* error bound that takes no account of the actual rounding errors. Moreover, it is not computable in practical applications since it involves the exact result $p(x)$ of the polynomial evaluation. We introduce here how to compute a valid error bound in pure floating point arithmetic in round to the nearest, which is also less pessimistic than the error estimate (12). Since underflow is rare and the quantities involved are almost always negligible, we do not take into account underflow in the following analysis.

First, we state the following lemma. When the coefficients of the polynomials p and q, and the argument x are all nonnegative floating point numbers, it gives a bound on $(p+q)(x)$ with respect to $\mathsf{HornerSum}\,(p, q, x)$.

Lemma 7. *Given $p(x) = \sum_{i=0}^{n} a_i x^i$ and $q(x) = \sum_{i=0}^{n} b_i x^i$ two polynomials such that all their coefficients are nonnegative floating point numbers, and given x a nonnegative floating point number. The following inequality holds*

$$0 \leq \sum_{i=0}^{n} (a_i + b_i) x^i \leq (1 + \mathbf{u})^{2n+1} \mathsf{HornerSum}\,(p, q, x).$$

Proof (without underflow). We consider Algorithm 8, and the intermediate variables r_i, for $i = n - 1, \ldots, 0$. Let us prove by induction that, for $i = 0, \ldots, n$,

$$\sum_{j=0}^{i} (a_{n-i+j} + b_{n-i+j}) x^j \leq (1 + \mathbf{u})^{2i+1} r_{n-i}. \tag{14}$$

For $i = 0$, $(a_n + b_n) \leq (1 + \mathbf{u})(a_n \oplus b_n) = (1 + \mathbf{u})r_n$, so Relation (14) is satisfied. Now we assume that Relation (14) is true for some integer i such that $0 \leq i < n$. Then

$$\sum_{j=0}^{i+1} (a_{n-(i+1)+j} + b_{n-(i+1)+j}) x^j = \left[\sum_{j=0}^{i} (a_{n-i+j} + b_{n-i+j}) x^j \right] x + (a_{n-i} + b_{n-i}).$$

By induction hypothesis we have

$$
\begin{aligned}
\sum_{j=0}^{i+1} (a_{n-(i+1)+j} + b_{n-(i+1)+j}) x^j &\leq (1 + \mathbf{u})^{2i+1} r_{n-i} x + (a_{n-i} + b_{n-i}) \\
&\leq (1 + \mathbf{u})^{2i+1} (1 + \mathbf{u})^2 \left(r_{n-i} \otimes x \oplus (a_{n-i} \oplus b_{n-i}) \right) \\
&\leq (1 + \mathbf{u})^{2(i+1)+1} r_{n-(i+1)}.
\end{aligned}
$$

Therefore Relation (14) is proved by induction, which in turn proves the lemma. $\qquad \square$

The proof in case of underflow is presented in Annex A.

Theorem 8. *Given a polynomial p with floating point coefficients, and a floating point value x, we consider $\mathsf{res} = \mathsf{CompensatedHorner}\,(p, x)$ the accurate evaluation of p at x (Algorithm 9). The absolute forward error affecting the evaluation is bounded according to*

$$|\mathsf{CompensatedHorner}\,(p, x) - p(x)| \leq \mathsf{fl}\left(\mathbf{u}|\mathsf{res}| + \left(\gamma_{4n+2} \mathsf{HornerSum}\,(|p_\pi|, |p_\sigma|, |x|) + 2\,\mathbf{u}^2 |\mathsf{res}| \right) \right). \tag{15}$$

For the proof of Theorem 8, we need the following two relations.

   i) For $(k+1)\,\mathbf{u} < 1$, we have $\gamma_k \leq (1-\mathbf{u})\gamma_{k+1}$. Indeed, if $(k+1)\,\mathbf{u} < 1$ then $k\,\mathbf{u} < (1-\mathbf{u})(k+1)\,\mathbf{u}$ and therefore

$$\gamma_k \leq \frac{(1-\mathbf{u})(k+1)\,\mathbf{u}}{1-(k+1)\,\mathbf{u}} = (1-\mathbf{u})\gamma_{k+1}.$$

   ii) We know that $\mathsf{fl}(ku) = ku \in \mathbb{F}$. Moreover, if $k\,\mathbf{u} < 1$, then $\mathsf{fl}(1-ku) = 1-ku \in \mathbb{F}$. So only the division suffers from a rounding error in the computation of γ_k. Thus

$$\gamma_k = \frac{k\,\mathbf{u}}{1-k\,\mathbf{u}} \leq (1-\mathbf{u})^{-1}\left[(k\,\mathbf{u}) \oslash (1-k\,\mathbf{u})\right] = (1-\mathbf{u})^{-1}\,\mathsf{fl}(\gamma_k).$$

Proof of Theorem 8 (without underflow). The key of the proof is to use relations in Remark 1 to bound real quantities with computable expressions. The result $\mathsf{res} = \mathsf{CompensatedHorner}\,(p_\pi, p_\sigma, x)$ computed by Algorithm 9 suffers from the following absolute forward error,

$$
\begin{aligned}
|\mathsf{res} - p(x)| &= |\mathsf{Horner}\,(p,x) \oplus \mathsf{HornerSum}\,(p_\pi, p_\sigma, x) - p(x)| \\
&\leq |(\mathsf{Horner}\,(p,x) \oplus \mathsf{HornerSum}\,(p_\pi, p_\sigma, x)) - (\mathsf{Horner}\,(p,x) + \mathsf{HornerSum}\,(p_\pi, p_\sigma, x))| \\
&\quad + |(\mathsf{Horner}\,(p,x) + \mathsf{HornerSum}\,(p_\pi, p_\sigma, x)) - p(x)|.
\end{aligned}
$$

As before, let $e(x) = (p_\pi + p_\sigma)(x)$. From Theorem 2, the EFT of $p(x)$ satisfies $p(x) = \mathsf{Horner}\,(p,x) + e(x)$. Thus

$$
\begin{aligned}
|\mathsf{res} - p(x)| &\leq \mathbf{u}\,|\mathsf{Horner}\,(p,x) \oplus \mathsf{HornerSum}\,(p_\pi, p_\sigma, x)| + |\mathsf{HornerSum}\,(p_\pi, p_\sigma, x) - e(x)| \\
&\leq \mathbf{u}|\mathsf{res}| + |\mathsf{HornerSum}\,(p_\pi, p_\sigma, x) - e(x)|.
\end{aligned}
$$

Now we bound the rightmost absolute value. Since p_π and p_σ are of degree $n-1$, Lemma 4 yields

$$|\mathsf{HornerSum}\,(p_\pi, p_\sigma, x) - e(x)| \leq \gamma_{2n-1}(\widetilde{p_\pi} + \widetilde{p_\sigma})(x),$$

and from Lemma 7, we write

$$|\mathsf{HornerSum}\,(p_\pi, p_\sigma, x) - e(x)| \leq (1+\mathbf{u})^{2n-1}\gamma_{2n-1}\mathsf{HornerSum}\,(|p_\pi|, |p_\sigma|, |x|).$$

Let $E = \mathsf{HornerSum}\,(|p_\pi|, |p_\sigma|, |x|)$. Since $(1+\mathbf{u})^{2n-1}\gamma_{2n-1} \leq \gamma_{4n-2} \leq (1-\mathbf{u})^4\gamma_{4n+2}$, it follows

$$|\mathsf{res} - p(x)| \leq \mathbf{u}|\mathsf{res}| + (1-\mathbf{u})^4\gamma_{4n+2}E.$$

Since $\gamma_{4n+2}E \leq (1-\mathbf{u})^{-1}\,\mathsf{fl}(\gamma_{4n+2})E \leq (1-\mathbf{u})^{-2}\,\mathsf{fl}(\gamma_{4n+2}E)$, we deduce

$$
\begin{aligned}
|\mathsf{res} - p(x)| &\leq \mathbf{u}|\mathsf{res}| + (1-\mathbf{u})^2\,\mathsf{fl}(\gamma_{4n+2}E) \\
&\leq (1-\mathbf{u})\,\mathbf{u}|\mathsf{res}| + (1-\mathbf{u})^2\,\mathsf{fl}(\gamma_{4n+2}E) + \mathbf{u}^2|\mathsf{res}|.
\end{aligned}
$$

We notice that $\mathbf{u}^2|\mathsf{res}|$ and $2\,\mathbf{u}^2|\mathsf{res}|$ are representable floating point values, since we assume that no underflow occurs. We can always assume that $2(1-\mathbf{u})^2 \geq 1$, thus

$$
\begin{aligned}
|\mathsf{res} - p(x)| &\leq (1-\mathbf{u})\,\mathbf{u}|\mathsf{res}| + (1-\mathbf{u})^2\left[\mathsf{fl}(\gamma_{4n+2}E) + 2\,\mathbf{u}^2|\mathsf{res}|\right] \\
&\leq (1-\mathbf{u})\left[\mathbf{u}|\mathsf{res}| + \mathsf{fl}(\gamma_{4n+2}E + 2\,\mathbf{u}^2|\mathsf{res}|)\right] \\
&\leq \mathsf{fl}\left(\mathbf{u}|\mathsf{res}| + (\gamma_{4n+2}E + 2\,\mathbf{u}^2|\mathsf{res}|)\right).
\end{aligned}
$$

$\square$

Relation (15) is easily evaluated concurrently with the computation of $p(x)$ according to Algorithm 9. One possible use of this error estimate is to provide a stopping criterion for a polynomial root finder. For instance, if $|\mathsf{CompensatedHorner}\,(p, x)|$ is of the same order as the computed error bound, then further iteration serves no purpose, as x could be a zero. We present some experiments to show the accuracy of the computable error estimate (15) compared to the accuracy of the *a priori* error bound (12) at the end of next Section 5.

5   Experimental results

All our experiments are performed using IEEE-754 double precision.

5.1   DDHorner is the Horner scheme with internal double-double computation

We compare the CompensatedHorner algorithm to an implementation of the classic Horner scheme that use internally the double-double format and denoted as DDHorner. Our implementation is based on the one proposed by the authors of [9, 16].

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 $|a_l| \le \mathbf{u}|a_h|$. To implement the Horner scheme using the double-double format, we only need two basic operations: i) the product of a double-double number by a double number, and ii) the addition of a double number to a double-double number. For the first operation we use Algorithm 10 that requires 25 flops. If TwoproductFMA is used instead of Twoproduct, then the flop count drops to 10. For the second operation, we use Algorithm 11 that requires 9 flops.

Horner scheme with internal double-double, DDHorner (Algorithm 12), requires $34n$ flops. If TwoProductFMA is used then the flops count drops to $19n$ flops.

Algorithm 10. Product of the double-double number (a_h, a_l) by the double number b

function $[c_h, c_l] = \mathsf{prod\_dd\_d}\,(a_h, a_l, b)$
$[s_h, s_l] = \mathsf{TwoProduct}\,(a_h, b)$
$[t_h, t_l] = \mathsf{FastTwoSum}\,(s_h, (a_l \otimes b))$
$[c_h, c_l] = \mathsf{FastTwoSum}\,(t_h, (t_l \oplus s_l))$

Algorithm 11. Addition of the double number b to the double-double number (a_h, a_l)

function $[c_h, c_l] = \mathsf{add\_dd\_d}\,(a_h, a_l, b)$
$[t_h, t_l] = \mathsf{TwoSum}\,(a_h, b)$
$[c_h, c_l] = \mathsf{FastTwoSum}\,(t_h, (t_l \oplus a_l))$

Algorithm 12. Horner scheme with internal double-double computations

function $[c_h, c_l] = \mathsf{DDHorner}\,(p, x)$
$s_h = a_n$
$s_l = 0$
for $i = n - 1 : -1 : 0$
$\quad [p_h, p_l] = \mathsf{prod\_dd\_d}\,(s_h, s_l, x)$
$\quad [s_h, s_l] = \mathsf{add\_dd\_d}\,(p_h, p_l, a_i)$
end
res$= s_h$

17

Table 1: Description of the routines experimented for the doubled working precision

routine	description of the corresponding Horner scheme
Horner	IEEE-754 double precision (Algorithm 1)
CompensatedHorner	Compensated Horner scheme (Algorithm 9)
DDHorner	Horner scheme with internal double-double computation (Algorithm 12)
MPFRHorner	Horner scheme in 106-bits precision arithmetic from MPFR library

5.2 Accuracy of the compensated Horner scheme

We test the expanded form of the polynomial $p_n(x) = (x-1)^n$. The argument x is chosen near to the unique real root 1 of p_n, and with many significant bits so that a lot of rounding errors occur during the evaluation of $p_n(x)$. We increment the degree n from 1 until a sufficiently large range has been covered by the condition number $\mathrm{cond}(p_n, x)$. Here we have

$$\mathrm{cond}(p_n, x) = \frac{\widetilde{p_n}(x)}{|p_n(x)|} = \left| \frac{1+x}{1-x} \right|^n,$$

and $\mathrm{cond}(p_n, x)$ grows exponentially with respect to n. In the experiments reported on Figure 1, $\mathrm{cond}(p_n, x)$ varies from 10^2 to 10^{40} (for $x = \mathsf{fl}(1.333)$, that corresponds to the degree range $n = 3, \ldots, 42$). These huge condition numbers have a sense since here the coefficients of p and the value x are chosen to be exact floating point numbers.

We experiment both Horner, CompensatedHorner and DDHorner (see Table 1). For each polynomial p_n, the exact value $p_n(x)$ is approximate with a high accuracy thanks to the arbitrary accurate MPFR library [19]. Figure 1 presents the relative accuracy $|y - p_n(x)|/|p_n(x)|$ of the evaluation y computed by the three algorithms. We set to the value one relative errors greater than one, which means that almost no useful information is left. The dotted lines represent the *a priori* error estimates (6) and (13).

We observe that the compensated algorithm exhibits the expected behavior, *i.e.*, the compensated rule of thumb. The full precision solution is computed as long as the condition number is smaller than $\mathbf{u}^{-1} \approx 10^{16}$. Then, for condition numbers between $\mathbf{u}^{-1}$ and $\mathbf{u}^{-2} \approx 10^{32}$, the relative error degrades to no accuracy at all. However, when the condition number is beyond $\mathbf{u}^{-1}$, the *a priori* error estimate (13) is always pessimistic by 2 or 3 order of magnitude.

5.3 Accuracy of the dynamic error bound

We experiment the accuracy of the dynamic error bound (15), compared to the *a priori* error bound (12) and to the actual forward error. We evaluate the expanded form of $p_5(x) = (1-x)^5$ for 1024 points near $x = 1$. For each value of the argument x, we compute CompensatedHorner (p_5, x), the associated dynamic error bound, and the actual forward error. The results are reported on Figure 2.

As already noticed in the previous paragraph, 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. The proposed dynamic error bound is more accurate as it takes into account the rounding errors that occur during the computation.
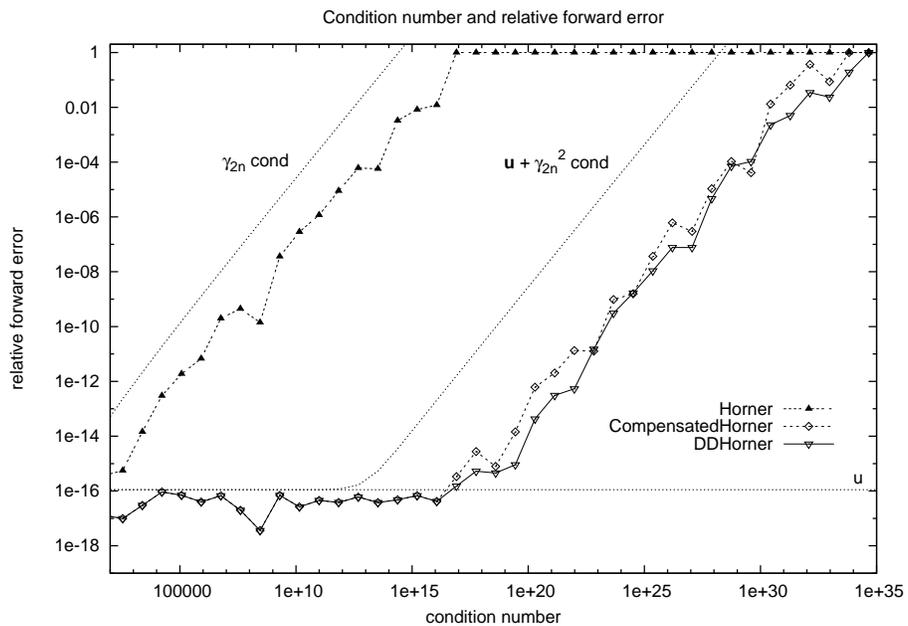
Figure 1: Accuracy of the Horner scheme performed in IEEE-754 precision compared to the accuracy of two algorithms CompensatedHorner and DDHorner.
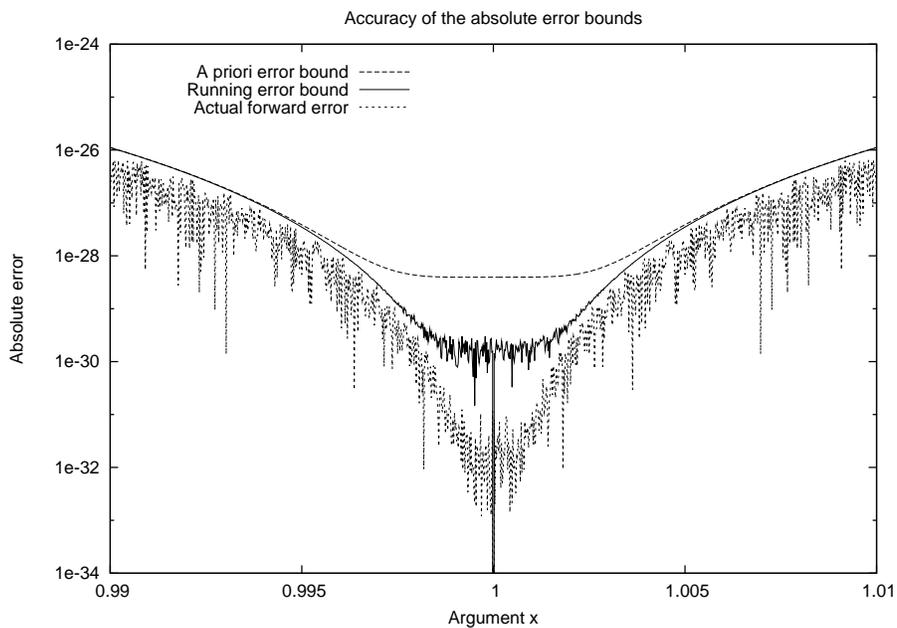
Figure 2: The dynamic error bound (15) compared to the theoretical bound (12) and to the actual absolute forward error.

Table 2: Experimental environments

environment	description
(I)	Intel Celeron, 2.4GHz, 1024kB L2 cache. GNU Compiler Collection 3.4.1
(II)	Intel Pentium, 3.0GHz, 1024kB L2 cache. GNU Compiler Collection 3.4.1

Table 3: Measured time performances for CompensatedHorner, DDHorner and MPFRHorner.

environment	CompensatedHorner/Horner				DDHorner/Horner				MPFRHorner/Horner		
	min.	mean	max.	theo.	min.	mean	max.	theo.	min.	mean	max.
(I)	1.4	3.1	3.4	13	2.3	8.4	9.4	17	22.4	97.5	124.4
(II)	1.5	2.9	3.2	13	2.3	8.4	9.4	17	18.1	83.7	96.8

5.4 Time performances

All the algorithms are implemented in C-code. In particular we use essentially the same programming techniques for the implementations of the routines CompensatedHorner and DDHorner. The experimental environments we have considered are listed in Table 2. Our measures are performed with polynomials whose degrees vary from 5 to 500 by steps of 5. We choose the coefficients and the arguments at random. For each degree, the routines are tested on the same polynomial with the same argument. Figure 3 displays the timings of CompensatedHorner, DDHorner and MPFRHorner normalized (dividing them) by the timing of the Horner routine. The minimum, the mean and the maximum of these normalized timings are reported in Table 3. For CompensatedHorner and DDHorner, the theoretical ratios are also reported, resulting from the number of flops involved by each algorithm.

First, we have to notice that the measured slowdown factor introduced either by CompensatedHorner or DDHorner is always significantly smaller than theoretically expected. This is an astonishing fact since the code for these functions is designed to be easily portable, and no algorithmic optimizations are performed, neither in CompensatedHorner, nor in DDHorner. This interesting property seems to be due to the fact that the classic algorithm performs only one
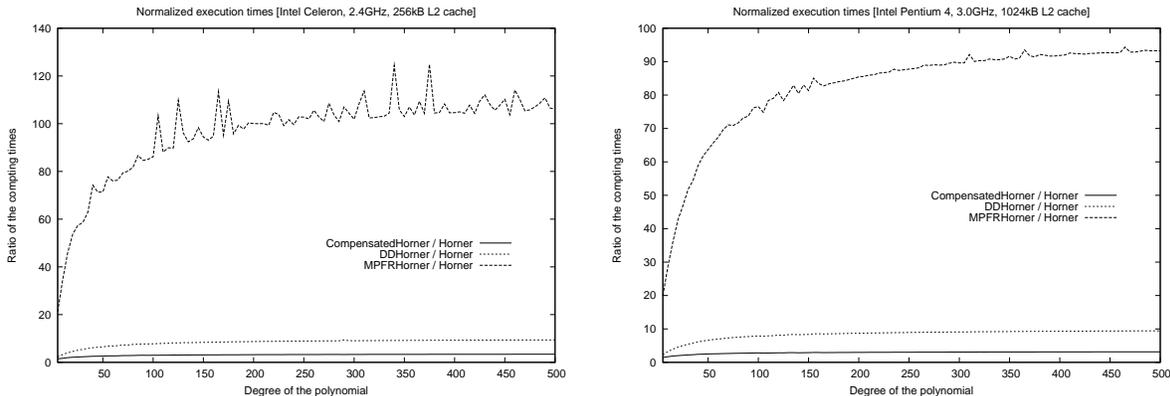
Figure 3: Normalized execution times of the three routines CompensatedHorner, DDHorner and MPFRHorner.

operation with each coefficient of the polynomial, whereas CompensatedHorner and DDHorern perform much more operations with each coefficient. Most of these operations are performed at the register level, without incurring much memory traffic. This practical efficiency is emphasized in [20] and motivates this kind of development.

The results reported in Table 3 show that the compensated algorithm CompensatedHorner is about 3 times slower than the classic Horner scheme. The same slowdown factor is about 8 for algorithm DDHorner. From a practical point of view, we can state that the proposed algorithm is more than twice faster than the Horner scheme with double-doubles. Table 3 also shows us that comparison with the MPFR library is not entirely fair in this context. Indeed, the routine MPFRHorner exhibits a slowdown factor of more than 80. It is not surprising since the MPFR library is specially designed to handle floating point numbers with extremely large mantissa.

6   Concluding remarks

We presented a compensated version of the Horner scheme to evaluate univariate polynomials in floating point arithmetic. We proved that the accuracy of the result computed by this compensated algorithm is similar to the one given by the Horner scheme performed in doubled working precision. The only assumption we made is that the floating point arithmetic available on the computer satisfies the IEEE-754 floating point standard. The same frame applies to the presented algorithm that compute the associated dynamic bound and to the recursive implementation in [7]. These low requirement make it highly portable and so these compensated algorithms could be easily integrated into numerical libraries or in-lined in specific subroutines.

This compensated algorithm uses only basic floating point operations and only the same working precision as the data. It uses no branch nor access to the mantissa that can be time consuming on modern architectures. As a result, it is fast not only in term of flop count but also in term of measured computing time. In particular, the slowdown factor due to the improvement of the accuracy is much smaller than theoretically expected. Our numerical experiments show than compensated Horner runs only about three times slower than the classic Horner scheme on nowadays computers.

Ogita-Rump-Oishi stress the interest to benefit from error-free transformations as, *e.g.*, TwoSum, TwoProd, available directly from the processor [20]. This paper emphasizes such an interest to provide more accurate and reliable numerical algorithms at a reasonable cost.

References

[1] David H. Bailey. *A Fortran-90 double-double library*, 2001. Available at URL = http://crd.lbl.gov/~dhbailey/mpdist/index.html.

[2] Sylvie Boldo and Jean-Michel Muller. Some functions computable with a fused-mac. In IEEE, editor, *Proceedings of the 17th IEEE Symposium on Computer Arithmetic, 2005, Cape Cod, Massachusetts, USA*. IEEE Computer Society Press, 2005.

[3] Theodorus J. Dekker. A floating-point technique for extending the available precision. *Numer. Math.*, 18:224–242, 1971.

[4] James Demmel. Underflow and the reliability of numerical software. *SIAM J. Sci. Stat. Comput.*, 5(4):887–919, 1984.

[5] FRISCO - a framework for integrated symbolic/numeric computation. Available at http://www.nag.co.uk/local/projects/FRISCO.html.

[6] Johannes Grabmeier, Erich Kaltofen, and Volker Weispfenning, editors. *Computer Algebra Handbook*. Springer-Verlag, Berlin, 2003.

[7] Stef Graillat, Philippe Langlois, and Nicolas Louvet. Recursive compensated Horner scheme. Research Report, DALI Research Project, Laboratory LP2A, Université de Perpignan Via Domitia, France, July 2005. (In progress).

[8] John R. Hauser. Handling floating-point exceptions in numeric programs. *ACM Trans. Program. Lang. Syst.*, 18(2):139–174, 1996.

[9] Yozo Hida, Xiaoye S. Li, and David H. Bailey. Algorithms for quad-double precision floating point arithmetic. In Neil Burgess and Luigi Ciminiera, editors, *Proceedings of the 15th Symposium on Computer Arithmetic, Vail, Colorado*, pages 155–162, Los Alamitos, CA, USA, 2001. Institute of Electrical and Electronics Engineers.

[10] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.

[11] 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(2):9-25, 1987.

[12] William Kahan. Further remarks on reducing truncation errors. *Comm. ACM*, 8(1):40, 1965.

[13] Donald Ervin Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley, Reading, MA, USA, third edition, 1998.

[14] Philippe Langlois. Automatic linear correction of rounding errors. *BIT*, 41(3):515–539, September 2001.

[15] Philippe Langlois. More accuracy at fixed precision. *J. Comp. Appl. Math.*, 162(1):57–77, January 2004.

[16] Xiaoye S. Li, James W. Demmel, David H. Bailey, Greg Henry, Yozo Hida, Jimmy Iskandar, William Kahan, Suh Y. Kang, Anil Kapur, Michael C. Martin, Brandon J. Thompson, Teresa Tung, and Daniel J. Yoo. Design, implementation and testing of extended and mixed precision BLAS. *ACM Trans. Math. Software*, 28(2):152–205, 2002.

[17] Seppo Linnainmaa. Towards accurate statistical estimation of rounding errors in floating-point computations. *BIT*, 15(2):165–173, 1975.

[18] Seppo Linnainmaa. Error linearization as an effective tool for experimental analysis of the numerical stability of algorithms. *BIT*, 23(3):346–359, 1983.

[19] The MPFR library. Available at http://www.mpfr.org.

[20] Takeshi Ogita, Siegfried M. Rump, and Shin'ichi Oishi. Accurate sum and dot product. *SIAM J. Sci. Comput.*, 26(6):1955–1988, 2005.

[21] Michèle Pichat. Correction d'une somme en arithmétique à virgule flottante. (French) [correction of a sum in floating-point arithmetic]. *Numer. Math.*, 19:400–406, 1972.

[22] Michèle Pichat. *Contributions à l'étude des erreurs d'arrondi en arithmétique à virgule flottante. (French) [Contributions to the error analysis of rounding errors in floating-point arithmetic].* Thèse, Université Scientifique et Médicale de Grenoble, 1976.

[23] Martti Tienari. A statistical model of roundoff error for varying length floating-point arithmetic. *BIT*, 10:355–365, 1970.

A   Proofs in case of underflow

Proof of Theorem 2 (with underfow). Considering Algorithm 7, for $i = 0 \cdots n - 1$, we have $s_{i+1}x = p_i + \pi_i + 5\eta_i$ with $|\eta_i| \leq \mathbf{v}$, and $p_i + a_i = s_i + \sigma_i$, thus

$$s_i = s_{i+1}x + a_i - \pi_i - \sigma_i - 5\eta_i.$$

Since $s_n = a_n$ we have therefore

$$s_0 = p(x) - \left[\sum_{i=0}^{n-1} \pi_i x^i\right] - \left[\sum_{i=0}^{n-1} \sigma_i x^i\right] - 5\left[\sum_{i=0}^{n-1} \eta_i x^i\right] \quad \text{with } |\eta_i| \leq \mathbf{v}.$$

$\square$

Proof of Proposition 3 (with underfow). Applying the standard model of floating point arithmetic with underflow (4), for $i = 1, \ldots, n$, the two computations in the loop of Algorithm 7 verify

$$
\begin{aligned}
|p_{n-i}| &= |s_{n-i+1} \otimes x| \leq (1 + \mathbf{u})|s_{n-i+1}||x| + \mathbf{v}, \quad \text{and} \\
|s_{n-i}| &= |p_{n-i} \oplus a_{n-i}| \leq (1 + \mathbf{u})(|p_{n-i}| + |a_{n-i}|).
\end{aligned}
$$

Let us prove by induction that, for $i = 1, \ldots, n$,

$$|p_{n-i}| \leq (1 + \gamma_{2i-1})\sum_{j=1}^{i} |a_{n-i+j}||x^j| + (1 + \gamma_{2i-2})\mathbf{v}\sum_{j=0}^{i-1} |x^j|, \quad \text{and} \tag{16}$$

$$|s_{n-i}| \leq (1 + \gamma_{2i})\sum_{j=0}^{i} |a_{n-i+j}||x^j| + (1 + \gamma_{2i-1})\mathbf{v}\sum_{j=0}^{i-1} |x^j|. \tag{17}$$

For $i = 1$, since $s_n = a_n$ we have $|p_{n-1}| \leq (1 + \mathbf{u})|a_n||x| + \mathbf{v} \leq (1 + \gamma_1)|a_n||x| + \mathbf{v}$ and (16) is satisfied. On the other hand, $|s_{n-1}| \leq (1 + \mathbf{u})\left((1 + \gamma_1)|a_n||x| + \mathbf{v} + |a_{n-1}|\right) \leq (1 + \gamma_2)\left(|a_n||x| + |a_{n-1}|\right) + (1 + \gamma_1)\mathbf{v}$, and (17) is also satisfied. Now we suppose that (16) and (17) are true for integers i such that $1 \leq i < n$. Then

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

By induction hypothesis,

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

23

Therefore we have

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

Relation (16) and Relation (17) are proved by induction. Thus, for $i = 1, \ldots, n$,

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

$$
|s_{n-i}||x^{n-i}| \leq (1+\gamma_{2n})\,\widetilde{p}(x) + (1+\gamma_{2n-1})\,\mathbf{v}\sum_{j=0}^{n-1}|x^j|.
$$

From Theorem 1, since TwoSum and TwoProd are EFT, for $i = 0, \cdots, n-1$, we have $|\pi_i| \leq \mathbf{u}|p_i| + 5\,\mathbf{v}$ and $|\sigma_i| \leq \mathbf{u}|s_i|$. Therefore

$$
(\widetilde{p_\pi} + \widetilde{p_\sigma})(x) = \sum_{i=0}^{n-1}(|\pi_i| + |\sigma_i|)|x^i| \leq \mathbf{u}\sum_{i=1}^{n}(|p_{n-i}| + |\sigma_{n-i}|)|x^{n-i}| + 5\,\mathbf{v}\sum_{i=0}^{n-1}|x^i|.
$$

And we obtain

$$
\begin{aligned}
(\widetilde{p_\pi} + \widetilde{p_\sigma})(x) &\leq n\,\mathbf{u}\left[(2+\gamma_{2n-1}+\gamma_{2n})\,\widetilde{p}(x) + (2+\gamma_{2n-2}+\gamma_{2n-1})\,\mathbf{v}\sum_{i=0}^{n-1}|x^i|\right] + 5\,\mathbf{v}\sum_{i=0}^{n-1}|x^i| \\
&\leq 2n\,\mathbf{u}(1+\gamma_{2n})\,\widetilde{p}(x) + [5 + 2n\,\mathbf{u}(1+\gamma_{2n-1})]\,\mathbf{v}\sum_{i=0}^{n-1}|x^i|
\end{aligned}
$$

Since $2n\,\mathbf{u}(1+\gamma_{2n}) = \gamma_{2n}$ and $2n\,\mathbf{u}(1+\gamma_{2n-1}) \leq \gamma_{2n}$, we finally obtain $(\widetilde{p_\pi} + \widetilde{p_\sigma})(x) \leq \gamma_{2n}\,\widetilde{p}(x) + (5 + \gamma_{2n})\,\mathbf{v}\sum_{i=0}^{n-1}|x^i|$. $\qquad\square$

Proof of Lemma 4 (with underflow). Considering Algorithm 8, we have $r_n = a_n \oplus b_n = (a_n + b_n)\langle 1 \rangle$, and for $i = n-1, \cdots, 0$,

$$
r_i = r_{i+1} \otimes x \oplus (a_i \oplus b_i) = \langle 2 \rangle r_{i+1}x + \langle 2 \rangle(a_i + b_i) + \eta_i, \quad \text{with} \quad |\eta_i| \leq \mathbf{v}.
$$

Therefore it can be proved by induction that

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

Since $r_0 = \mathsf{HornerSum}\,(p, q, x)$, we finally obtain

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

$\qquad\square$

Proof of Theorem 5 (with underfow). As before, we use the notation $e(x) = (p_\pi + p_\sigma)(x)$. From Theorem 2,

$$
\begin{aligned}
|\mathsf{res} - p(x)| &= |(1 + \varepsilon)(s_0 + c) - p(x)| \\
&= \left| (1 + \varepsilon) \left(p(x) - e(x) - 5 \left(\sum_{i=0}^{n-1} \eta_i x^i \right) + c \right) - p(x) \right| \\
&\leq \mathbf{u}|p(x)| + (1 + \mathbf{u})|c - e(x)| + 5(1 + u)\,\mathbf{v} \sum_{i=0}^{n-1} |x^i|.
\end{aligned}
$$

Since p_π and p_σ are two polynomials of degree $n-1$, and $c = \mathsf{HornerSum}\,(p_\pi, p_\sigma, x)$, applying Lemma 4, we have

$$
|c - e(x)| \leq \gamma_{2n-1}(\widetilde{p_\pi} + \widetilde{p_\sigma})(x) + (1 + \gamma_{2n-1})\,\mathbf{v} \sum_{i=0}^{n-2} |x^i|.
$$

Then we apply Proposition 3 to bound $(\widetilde{p_\pi} + \widetilde{p_\sigma})(x)$. We write

$$
|c - e(x)| \leq \gamma_{2n-1}\gamma_{2n}\,\widetilde{p}(x) + \gamma_{2n-1}(5 + \gamma_{2n})\,\mathbf{v} \sum_{i=0}^{n-1} + (1 + \gamma_{2n-1})\,\mathbf{v} \sum_{i=0}^{n-2} |x^i|.
$$

Since $(1 + \mathbf{u})\gamma_{2n-1} \leq \gamma_{2n}$, we finally write,

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

with

$$
\alpha = (1 + \mathbf{u})\,[\gamma_{2n-1}(5 + \gamma_{2n}) + 5]\,\mathbf{v} \sum_{i=0}^{n-1} |x^i| + (1 + \mathbf{u})(1 + \gamma_{2n-1})\,\mathbf{v} \sum_{i=0}^{n-2} |x^i|.
$$

As $\mathbf{v}$ is a very small constant, we simply bound the term α as follow.

$$
\alpha \leq (1 + \mathbf{u})\,[6 + 6\gamma_{2n-1} + \gamma_{2n-1}\gamma_{2n}]\,\mathbf{v} \sum_{i=0}^{n-1} |x^i| \leq K\,\mathbf{v} \sum_{i=0}^{n-1} |x^i|,
$$

with $K \leq 7$. $\qquad\square$