Compensated Horner scheme in complex floating point arithmetic

Stef Graillat, Valérie Ménissier-Morain
UPMC Univ Paris 06, UMR 7606, LIP6
4 place Jussieu, F-75252, Paris cedex 05, France
stef.graillat@lip6.fr,valerie.menissier-morain@lip6.fr

Abstract

Several different techniques and softwares intend to improve the accuracy of results computed in a fixed finite precision. Here we focus on a method to improve the accuracy of polynomial evaluation via Horner's scheme. Such an algorithm exists for polynomials with real floating point coefficients. In this paper, we provide a new algorithm which deals with polynomials with complex floating point coefficients. We show that the computed result is as accurate as if computed in twice the working precision. The algorithm is simple since it only requires addition, subtraction and multiplication of floating point numbers in the same working precision as the given data. Such an algorithm can be useful for example to compute zeros of polynomial by Newton-like methods.

1 Introduction

It is well-known that computing with finite precision implies some rounding errors. These errors can lead to inexact results for a computation. An important tool to try to avoid this are *error-free transformations*: to compute not only a floating point approximation but also an exact error term without overlapping. This can be viewed as a *double-double* floating point numbers [10] but without the renormalisation step.

Error-free transformations have been widely used to provide some new accurate algorithms in real floating point arithmetic (see [12, 14] for accurate sum and dot product and [5] for polynomial evaluation). Complex error-free transformations are then the next step for providing accurate algorithms using complex numbers.

The rest of the paper is organized as follows. In Section 2, we recall some results on real floating point arithmetic and error-free transformations. In Section 3, we present the complex floating point arithmetic and we propose some new error-free transformations for this arithmetic. In Section 4, we study different polynomial evaluation algorithms. We first describe the Horner scheme in complex floating point arithmetic. We then present the compensated Horner scheme in complex arithmetic. We provide an error analysis for both versions of the Horner scheme and we conclude by presenting some numerical experiments confirming the accuracy of our algorithm.

2 Real floating point arithmetic

In this section, we first recall the principle of real floating point arithmetic. Then we present the well-known error-free transformations associated with the classical operations addition, subtraction, multiplication.

2.1 Notations and fundamental property of real floating point arithmetic

Throughout the paper, we assume to work with a floating point arithmetic adhering to IEEE 754 floating point standard [8]. We assume that no overflow nor underflow occur. The set of floating point numbers is denoted by $\mathbb{F}$, the relative rounding error by eps. For IEEE 754 double precision, we have $\mathsf{eps} = 2^{-53}$ and for single precision $\mathsf{eps} = 2^{-24}$.

We denote by $\mathrm{fl}(\cdot)$ the result of a floating point computation, where all operations inside parentheses are done in floating point working precision. Floating point operations in IEEE 754 satisfy [7]

$$\mathrm{fl}(a \circ b) = (a \circ b)(1+\varepsilon_1) = (a \circ b)/(1+\varepsilon_2) \text{ for } \circ = \{+,-,\cdot,/\} \text{ and } |\varepsilon_\nu| \le \mathsf{eps}.$$

This implies that

$$|a \circ b - \mathrm{fl}(a \circ b)| \le \mathsf{eps}|a \circ b| \text{ and } |a \circ b - \mathrm{fl}(a \circ b)| \le \mathsf{eps}|\,\mathrm{fl}(a \circ b)| \text{ for } \circ = \{+,-,\cdot,/\}. \quad (2.1)$$

We use standard notation for error estimations. The quantities γ_n are defined as usual [7] by

$$\gamma_n := \frac{n\mathsf{eps}}{1-n\mathsf{eps}} \quad \text{for } n \in \mathbb{N},$$

where we implicitly assume that $n\mathsf{eps} \le 1$ and we will use inequality $\mathsf{eps} \le \sqrt{2}\gamma_2$ in the following proofs.

2.2 Error-free transformations in real floating point arithmetic

One can notice that $a \circ b \in \mathbb{R}$ and $\mathrm{fl}(a \circ b) \in \mathbb{F}$ but in general we do not have $a \circ b \in \mathbb{F}$. It is known that for the basic operations $+,-,\cdot$, the approximation error of a floating point operation is still a floating point number (see for example [3]):

$$\begin{array}{lll} x = \mathrm{fl}(a \pm b) & \Rightarrow \quad a \pm b = x + y & \text{with } y \in \mathbb{F}, \\ x = \mathrm{fl}(a \cdot b) & \Rightarrow \quad a \cdot b = x + y & \text{with } y \in \mathbb{F}. \end{array} \quad (2.2)$$

These are *error-free* transformations of the pair (a, b) into the pair (x, y). Fortunately, the quantities x and y in (2.2) can be computed exactly in floating point arithmetic.
We use Matlab-like notations to describe the algorithms.

2.2.1 Addition

For addition, we can use the following algorithm by Knuth [9, Thm B. p.236].

Algorithm 2.1 (Knuth [9]). ERROR-FREE TRANSFORMATION OF THE SUM OF TWO FLOATING POINT NUMBERS

```
function $[x, y] = \mathtt{TwoSum}(a, b)$
  $x = \mathrm{fl}(a + b)$;   $z = \mathrm{fl}(x - a)$;   $y = \mathrm{fl}((a - (x - z)) + (b - z))$
```

Another algorithm to compute an error-free transformation is the following algorithm from Dekker [3]. The drawback of this algorithm is that we have $x + y = a + b$ provided that $|a| \geq |b|$. Generally, on modern computers, a comparison followed by a branching and 3 operations costs more than 6 operations. As a consequence, `TwoSum` is generally more efficient than `FastTwoSum`.

Algorithm 2.2 (Dekker [3]). Error-free transformation of the sum of two floating point numbers with $|a| \geq |b|$

```
function $[x, y] = \texttt{FastTwoSum}(a, b)$
  $x = \mathrm{fl}(a + b)$;    $y = \mathrm{fl}((a - x) + b)$
```

2.2.2 Multiplication

For the error-free transformation of a product, we first need to split the input argument into two parts.

Splitting

Let p be the integer number given by $\mathsf{eps} = 2^{-p}$ and define $s = \lceil p/2 \rceil$ and $\mathsf{factor} = \mathrm{fl}(2^s + 1)$. For example, if the working precision is IEEE 754 double precision, then $p = 53$ and $s = 27$ and $\mathsf{factor} = 1.\underbrace{00\ldots00}_{26}1\underbrace{00\ldots00}_{26}\,2^{27}$ allows by multiplying a number to split the mantissa of this number into its most and least significant halves. The quantities p, s and factor are constants of the floating point arithmetic.

The following algorithm by Dekker [3] splits a floating point number $a \in \mathbb{F}$ into two parts x and y such that

$$a = x + y \quad \text{and} \quad x \text{ and } y \text{ non overlapping with } |y| \leq |x|.$$

Two floating point values x and y with $|y| \leq |x|$ are nonoverlapping if the least significant nonzero bit of x is more significant than the most significant nonzero bit of y.

Algorithm 2.3 (Dekker [3]). Error-free split of a floating point number into two parts

```
function $[x, y] = \texttt{Split}(a, b)$
  $c = \mathrm{fl}(\mathsf{factor} \cdot a)$;    $x = \mathrm{fl}(c - (c - a))$;    $y = \mathrm{fl}(a - x)$
```

Product

An algorithm from Veltkamp (see [3]) makes it possible to compute an error-free transformation for the product of two floating point numbers by splitting the two arguments.
This algorithm returns two floating point numbers x and y such that

$$a \cdot b = x + y \quad \text{with } x = \mathrm{fl}(a \cdot b).$$

Algorithm 2.4 (Veltkamp [3]). Error-free transformation of the product of two floating point numbers

```
function $[x, y] = \texttt{TwoProduct}(a, b)$
  $x = \mathrm{fl}(a \cdot b)$
  $[a_1, a_2] = \texttt{Split}(a)$;    $[b_1, b_2] = \texttt{Split}(b)$
  $y = \mathrm{fl}(a_2 \cdot b_2 - (((x - a_1 \cdot b_1) - a_2 \cdot b_1) - a_1 \cdot b_2))$
```

2.2.3 Properties

The following theorem summarizes the properties of algorithms `TwoSum` and `TwoProduct`.

Theorem 2.1 (Ogita, Rump and Oishi [12])**.**

Addition *Let $a, b \in \mathbb{F}$ and let $x, y \in \mathbb{F}$ such that $[x, y] = \texttt{TwoSum}(a, b)$ (Algorithm 2.1). Then,*

$$a + b = x + y, \quad x = \mathrm{fl}(a + b), \quad |y| \le \mathsf{eps}|x|, \quad |y| \le \mathsf{eps}|a + b|. \tag{2.3}$$

The algorithm `TwoSum` *requires* 6 *flops.*

Product *Let $a, b \in \mathbb{F}$ and let $x, y \in \mathbb{F}$ such that $[x, y] = \texttt{TwoProduct}(a, b)$ (Algorithm 2.4). Then,*

$$a \cdot b = x + y, \quad x = \mathrm{fl}(a \cdot b), \quad |y| \le \mathsf{eps}|x|, \quad |y| \le \mathsf{eps}|a \cdot b|. \tag{2.4}$$

The algorithm `TwoProduct` *requires* 17 *flops.*

2.2.4 Multiplication with FMA

The `TwoProduct` algorithm can be re-written in a very simple way if a Fused-Multiply-and-Add (`FMA`) operator is available on the targeted architecture [11, 1]. This means that for $a, b, c \in \mathbb{F}$, the result of $\texttt{FMA}(a, b, c)$ is the nearest floating point number of $a \cdot b + c \in \mathbb{R}$. The `FMA` operator satisfies

$$\texttt{FMA}(a, b, c) = (a \cdot b + c)(1 + \varepsilon_1) = (a \cdot b + c)/(1 + \varepsilon_2) \text{ with } |\varepsilon_\nu| \le \mathsf{eps}.$$

Algorithm 2.5 (Ogita, Rump and Oishi [12])**.** Error-free transformation of the product of two floating point numbers using an `FMA`

```
function $[x, y] = \texttt{TwoProductFMA}(a, b)$
  $x = \mathrm{fl}(a \cdot b)$;   $y = \texttt{FMA}(a, b, -x)$
```

The `TwoProductFMA` algorithm requires only 2 flops.

3 Complex floating point arithmetic

3.1 Notations and fundamental property of complex floating point arithmetic

We denote by $\mathbb{F}+i\mathbb{F}$ the set of complex floating point numbers. As in the real case, we denote by $\mathrm{fl}(\cdot)$ the result of a floating point computation, where all operations inside parentheses are done in floating point working precision in the obvious way [7, p.71]. The following properties hold [7, 13] for $x, y \in \mathbb{F} + i\mathbb{F}$,

$$\mathrm{fl}(x \circ y) = (x \circ y)(1 + \varepsilon_1) = (x \circ y)/(1 + \varepsilon_2), \text{ for } \circ = \{+, -\} \text{ and } |\varepsilon_\nu| \le \mathsf{eps}, \tag{3.5}$$

and

$$\mathrm{fl}(x \cdot y) = (x \cdot y)(1 + \varepsilon_1), \qquad |\varepsilon_1| \le \sqrt{2}\gamma_2. \tag{3.6}$$

This implies that

$$|a \circ b - \mathrm{fl}(a \circ b)| \le \mathsf{eps}|a \circ b| \text{ and } |a \circ b - \mathrm{fl}(a \circ b)| \le \mathsf{eps}|\,\mathrm{fl}(a \circ b)| \text{ for } \circ = \{+, -\}$$

and

$$|x \cdot y - \mathrm{fl}(x \cdot y)| \leq \sqrt{2}\gamma_2 |x \cdot y|.$$

For the complex multiplication, we can replace the term $\sqrt{2}\gamma_2$ by $\sqrt{5}\mathsf{eps}$ which is nearly optimal (see [2]). As a consequence, in the sequel, all the bounds for algorithms involving a multiplication can be improved by a small constant factor.

We will also use the notation $\widetilde{\gamma}_n$ for the quantities

$$\widetilde{\gamma}_n := \frac{n\sqrt{2}\gamma_2}{1 - n\sqrt{2}\gamma_2}.$$

And we will use inequalities $(1+\sqrt{2}\gamma_2)(1+\widetilde{\gamma}_n) \leq (1+\widetilde{\gamma}_{n+1})$ and $(1+\sqrt{2}\gamma_2)\widetilde{\gamma}_{n-1} \leq \widetilde{\gamma}_n$.

3.2 Sum and product

The error-free transformations presented hereafter were first described in [6]. The sum requires still only one error term as for the real case but the product needs three error terms.

3.2.1 Addition

Algorithm 3.1. Error-free transformation of the sum of two complex floating point numbers $x = a + ib$ and $y = c + id$

```
function $[s, e] = \texttt{TwoSumCplx}(x, y)$
    $[s_1, e_1] = \texttt{TwoSum}(a, c)$;    $[s_2, e_2] = \texttt{TwoSum}(b, d)$
    $s = s_1 + is_2$;    $e = e_1 + ie_2$
```

Theorem 3.1. *Let $x, y \in \mathbb{F} + i\mathbb{F}$ and let $s, e \in \mathbb{F} + i\mathbb{F}$ such that $[s, e] = \texttt{TwoSumCplx}(x, y)$ (Algorithm 3.1). Then,*

$$x + y = s + e, \quad s = \mathrm{fl}(x + y), \quad |e| \leq \mathsf{eps}|s|, \quad |e| \leq \mathsf{eps}|x + y|. \tag{3.7}$$

The algorithm `TwoSumCplx` *requires* 12 *flops.*

Proof. From Theorem 2.1 with `TwoSum`, we have $s_1 + e_1 = a + c$ and $s_2 + e_2 = b + d$. It follows that $s + e = x + y$ with $s = \mathrm{fl}(x + y)$. From (3.5), we derive that $|e| \leq \mathsf{eps}|s|$ and $|e| \leq \mathsf{eps}|x + y|$. □

3.2.2 Multiplication

Algorithm 2.4 cannot be straightforward generalized to complex multplication. We need the new following algorithm.

Algorithm 3.2. Error-free transformation of the product of two complex floating point numbers $x = a + ib$ and $y = c + id$

```
function $[p, e, f, g] = \texttt{TwoProductCplx}(x, y)$
    $[z_1, h_1] = \texttt{TwoProduct}(a, c)$;    $[z_2, h_2] = \texttt{TwoProduct}(b, d)$
    $[z_3, h_3] = \texttt{TwoProduct}(a, d)$;    $[z_4, h_4] = \texttt{TwoProduct}(b, c)$
    $[z_5, h_5] = \texttt{TwoSum}(z_1, -z_2)$;    $[z_6, h_6] = \texttt{TwoSum}(z_3, z_4)$
    $p = z_5 + iz_6$;    $e = h_1 + ih_3$;    $f = -h_2 + ih_4$;    $g = h_5 + ih_6$
```

Theorem 3.2. *Let $x, y \in \mathbb{F}+i\mathbb{F}$ and let $p, e, f, g \in \mathbb{F}+i\mathbb{F}$ such that $[p, e, f, g] =$ `TwoProductCplx`(x, y) (Algorithm 2.4). Then,*

$$x \cdot y = p + e + f + g \quad p = \text{fl}(x \cdot y), \quad |e + f + g| \leq \sqrt{2}\gamma_2 |x \cdot y|, \tag{3.8}$$

The algorithm `TwoProductCplx` *requires* 80 *flops.*

Proof. From Theorem 2.1, it holds that $z_1 + h_1 = a \cdot c$, $z_2 + h_2 = b \cdot d$, $z_3 + h_3 = a \cdot d$, $z_4 + h_4 = b \cdot c$, $z_5 + h_5 = z_1 - z_2$ and $z_6 + h_6 = z_3 + z_4$. By the definition of p, e, f, g, we conclude that $x \cdot y = p + e + f + g$ with $p = \text{fl}(x \cdot y)$. From (3.6), we deduce that $|e + f + g| = |x \cdot y - \text{fl}(x \cdot y)| \leq \sqrt{2}\gamma_2 |x \cdot y|$. □

Optimization of the algorithm

In Algorithm 3.2, in each call to `TwoProduct`, we have to split the two arguments. Yet, we split the same numbers a, b, c and d twice. With only one split for each of these numbers, the cost is 64 flops. The previous algorithm can be expanded as follows:

Algorithm 3.3. Error-free transformation of the product of two complex floating point numbers $x = a + ib$ and $y = c + id$ with single splitting

```
function $[p, e, f, g]$ = TwoProductCplxSingleSplitting$(x, y)$
    $[a_1, a_2]$ = Split$(a)$, $[b_1, b_2]$ = Split$(b)$, $[c_1, c_2]$ = Split$(c)$, $[d_1, d_2]$ = Split$(d)$
    $z1 = \text{fl}(a \cdot c)$,   $z2 = \text{fl}(b \cdot d)$,   $z3 = \text{fl}(a \cdot d)$,   $z4 = \text{fl}(b \cdot c)$
    $h_1 = \text{fl}(a_2 \cdot c_2 - (((x - a_1 \cdot c_1) - a_2 \cdot c_1) - a_1 \cdot c_2))$
    $h_2 = \text{fl}(b_2 \cdot d_2 - (((x - b_1 \cdot d_1) - b_2 \cdot d_1) - b_1 \cdot d_2))$
    $h_3 = \text{fl}(a_2 \cdot d_2 - (((x - a_1 \cdot d_1) - a_2 \cdot d_1) - a_1 \cdot d_2))$
    $h_4 = \text{fl}(b_2 \cdot c_2 - (((x - b_1 \cdot c_1) - b_2 \cdot c_1) - b_1 \cdot c_2))$
    $[z_5, h_5]$ = TwoSum$(z_1, -z_2)$,   $[z_6, h_6]$ = TwoSum$(z_3, z_4)$
    $p = z_5 + iz_6$,   $e = h_1 + ih_3$,   $f = -h_2 + ih_4$,   $g = h_5 + ih_6$
```

3.2.3 Multiplication with FMA

Of course we obtain a much faster algorithm if we use `TwoProductFMA` instead of `TwoProduct`. In that case, the numbers of flops falls down to 20.

Algorithm 3.4. Error-free transformation of the product of two complex floating point numbers $x = a + ib$ and $y = c + id$ using `FMA`

```
function $[p, e, f, g]$ = TwoProductFMACplx$(x, y)$
    $[z_1, h_1]$ = TwoProductFMA$(a, c)$;   $[z_2, h_2]$ = TwoProductFMA$(b, d)$
    $[z_3, h_3]$ = TwoProductFMA$(a, d)$;   $[z_4, h_4]$ = TwoProductFMA$(b, c)$
    $[z_5, h_5]$ = TwoSum$(z_1, -z_2)$;   $[z_6, h_6]$ = TwoSum$(z_3, z_4)$
    $p = z_5 + iz_6$;   $e = h_1 + ih_3$;   $f = -h_2 + ih_4$;   $g = h_5 + ih_6$
```

The 8.5:1 ratio between the cost of `TwoProduct` and `TwoProductFMA` algorithms and the 3.2:1 ratio between the cost of `TwoProductCplxSingleSplitting` and `TwoProductFMACplx` algorithms show that the availability of an `FMA` is crucial for fast error-free transformations in real and complex arithmetic.

4 Accurate polynomial evaluation

First of all we describe the classical Horner scheme to evaluate polynomial p with complex floating point coefficients on x a complex floating point value. The computed value `res` is generally not the mathematical value $p(x)$ rounded to the working precision. We want then to reduce the gap between these values so we modify this algorithm to compute `res` and additionally four polynomial error terms that we will have to evaluate on x to deduce a complex floating point correction term c that we have to add to `res`. Afterwards we will study mathematically and experimentally the improvement of the accuracy consisting in replacing `res` by $\mathrm{fl}(\texttt{res} + c)$.

4.1 Classical Horner scheme for complex floating point arithmetic

The classical method for evaluating a polynomial

$$p(x) = \sum_{i=0}^{n} a_i x^i, \quad a_i, x \in \mathbb{F} + i\mathbb{F}$$

is the Horner scheme which consists on the following algorithm:

Algorithm 4.1. Polynomial evaluation with Horner's scheme

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

Proposition 4.1. *A forward error bound is*

$$|p(x) - \texttt{Horner}(p, x)| \le \widetilde{\gamma}_{2n} \sum_{i=0}^{n} |a_i||x|^i = \widetilde{\gamma}_{2n} \widetilde{p}(|x|) \tag{4.9}$$

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

Proof. This is a straightforward adaptation of the proof found in [7, p.95] using (3.5) and (3.6) for complex floating point arithmetic. □

The classical condition number that describes the evaluation of $p(x) = \sum_{i=0}^{n} a_i x^i$ at x is

$$\mathrm{cond}(p, x) = \frac{\sum_{i=0}^{n} |a_i||x|^i}{|\sum_{i=0}^{n} a_i x^i|} = \frac{\widetilde{p}(|x|)}{|p(x)|}. \tag{4.10}$$

Thus if $p(x) \ne 0$, Equations (4.9) and (4.10) can be combined so that

$$\frac{|p(x) - \texttt{Horner}(p, x)|}{|p(x)|} \le \widetilde{\gamma}_{2n}\, \mathrm{cond}(p, x). \tag{4.11}$$

4.2 Compensated Horner scheme

We now propose an error-free transformation for polynomial evaluation with the Horner scheme. We produce four polynomial error terms monomial-by-monomial: a monomial for each polynomial at each iteration.

Algorithm 4.2. Error-free transformation for the Horner scheme

```
function $[\texttt{res}, p_\pi, p_\mu, p_\nu, p_\sigma] = \texttt{EFTHorner}(p, x)$
  $s_n = a_n$
  for $i = n-1 : -1 : 0$
    $[p_i, \pi_i, \mu_i, \nu_i] = \texttt{TwoProductCplx}(s_{i+1}, x)$
    $[s_i, \sigma_i] = \texttt{TwoSumCplx}(p_i, a_i)$
    Set $\pi_i$, $\mu_i$, $\nu_i$, $\sigma_i$ respectively as the coefficient of degree $i$ in $p_\pi$, $p_\mu$, $p_\nu$, $p_\sigma$
  end
  $\texttt{res} = s_0$
```

The next theorems and proofs are very similar to the ones of [5]. It is just necessary to change real error-free transformations into complex error-free transformations and to change eps into $\sqrt{2}\gamma_2$. This leads to change the γ_n into $\widetilde{\gamma}_n$.

Theorem 4.2 (Equality). *Let $p(x) = \sum_{i=0}^{n} a_i x^i$ be a polynomial of degree n with complex floating point coefficients, and let x be a complex floating point value. Then Algorithm 4.2 computes both*

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

ii) four polynomials p_π, p_μ, p_ν and p_σ of degree $n-1$ with complex floating point coefficients,

Then,

$$p(x) = \texttt{res} + (p_\pi + p_\sigma + p_\mu + p_\nu)(x), \tag{4.12}$$

Proof. Thanks to the error-free transformations, we have $p_i + \pi_i + \mu_i + \nu_i = s_{i+1}.x$ and $s_i + \sigma_i = p_i + a_i$. By induction, it is easy to show that

$$\sum_{i=0}^{n} a_i x^i = s_0 + \sum_{i=0}^{n-1} \pi_i x^i + \sum_{i=0}^{n-1} \mu_i x^i + \sum_{i=0}^{n-1} \nu_i x^i + \sum_{i=0}^{n-1} \sigma_i x^i,$$

which is exactly (4.12). □

Proposition 4.3 (Bound on the error). *Given $p(x) = \sum_{i=0}^{n} a_i x^i$ a polynomial of degree n with complex floating point coefficients, and x a complex floating point value. Let* res *be the floating point value, p_π, p_μ, p_ν and p_σ be the four polynomials of degree $n-1$, with complex floating point coefficients, such that $[\texttt{res}, p_\pi, p_\mu, p_\nu, p_\sigma] = \texttt{EFTHorner}(p, x)$. Then,*

$$(\widetilde{(p_\pi + p_\mu + p_\nu)} + \widetilde{p_\sigma})(|x|) \leq \widetilde{\gamma}_{2n}\widetilde{p}(|x|).$$

Proof. The proof is organized as follows: we prove a bound on $|p_{n-i}||x|^{n-i}$ and $|s_{n-i}||x|^{n-i}$ from which we deduce a bound on $|\pi_i + \mu_i + \nu_i|$ and $|\sigma_i|$ and we use these bounds on each coefficient of the polynomial error terms to obtain finally the expected bound on these polynomials.

- By definition, for $i = 1, \dots, n$, $p_{n-i} = s_{n-i+1} \cdot x$ and $s_{n-i} = p_{n-i} + a_{n-i}$. From Equation(3.6), we deduce $\mathrm{fl}(s_{n-i+1} \cdot x) = (1+\varepsilon_1)s_{n-i+1} \cdot x$ with $|\varepsilon_1| \le \sqrt{2}\gamma_2$. From Equation (3.5), we deduce $\mathrm{fl}(p_{n-i} + a_{n-i}) = (1+\varepsilon_2)(p_{n-i} + a_{n-i})$ with $|\varepsilon_2| \le \mathsf{eps} \le \sqrt{2}\gamma_2$. Consequently

$$|p_{n-i}| \le (1+\sqrt{2}\gamma_2)\,|s_{n-i+1}||x| \quad \text{and} \quad |s_{n-i}| \le (1+\sqrt{2}\gamma_2)\,(|p_{n-i}| + |a_{n-i}|). \tag{4.13}$$

- These two bounds will be used in the basic case and the inductive case of the following double property: for $i = 1, \dots, n$,

$$|p_{n-i}| \le (1+\widetilde{\gamma}_{2i-1}) \sum_{j=1}^{i} |a_{n-i+j}||x^j| \quad \text{and} \quad |s_{n-i}| \le (1+\widetilde{\gamma}_{2i}) \sum_{j=0}^{i} |a_{n-i+j}||x^j|. \tag{4.14}$$

For $i = 1$:

Since $s_n = a_n$ the bound of Equation (4.13) can be rewritten as $|p_{n-1}| \le (1+\sqrt{2}\gamma_2)|a_n||x| \le (1+\widetilde{\gamma}_1)|a_n||x|$. We combine this bound on $|p_{n-1}|$ to (4.13) to obtain $|s_{n-1}| \le (1+\sqrt{2}\gamma_2)\left((1+\widetilde{\gamma}_1)|a_n||x| + |a_{n-1}|\right) \le (1+\widetilde{\gamma}_2)\left(|a_n||x| + |a_{n-1}|\right)$. Thus (4.14) is satisfied for $i = 1$.

Let us now suppose that (4.14) is true for some integer i such that $1 \le i < n$. According to (4.13), we have $|p_{n-(i+1)}| \le (1+\sqrt{2}\gamma_2)|s_{n-i}||x|$. Thanks to the induction hypothesis, we derive,

$$|p_{n-(i+1)}| \le (1+\sqrt{2}\gamma_2)(1+\widetilde{\gamma}_{2i}) \sum_{j=0}^{i} |a_{n-i+j}||x^{j+1}| \le (1+\widetilde{\gamma}_{2(i+1)-1}) \sum_{j=1}^{i+1} |a_{n-(i+1)+j}||x^j|.$$

Let us combine (4.13) with this inequality, we have,

$$\begin{aligned} |s_{n-(i+1)}| &\le (1+\sqrt{2}\gamma_2)(|p_{n-(i+1)}| + |a_{n-(i+1)}|) \\ &\le (1+\sqrt{2}\gamma_2)(1+\widetilde{\gamma}_{2(i+1)-1}) \left[\sum_{j=1}^{i+1} |a_{n-(i+1)+j}||x^j| + |a_{n-(i+1)}|\right] \\ &\le (1+\widetilde{\gamma}_{2(i+1)}) \sum_{j=0}^{i+1} |a_{n-(i+1)+j}||x^j|. \end{aligned}$$

So (4.14) is proved by induction. We bound each of these sums by $p(|x|)/|x^{n-i}|$ and obtain for $i = 1, \dots, n$,

$$|p_{n-i}||x^{n-i}| \le (1+\widetilde{\gamma}_{2i-1})\widetilde{p}(|x|) \quad \text{and} \quad |s_{n-i}||x^{n-i}| \le (1+\widetilde{\gamma}_{2i})\widetilde{p}(|x|). \tag{4.15}$$

- From Theorem 3.1 and Theorem 3.2, for $i = 0, \dots, n-1$, we have $|\pi_i + \mu_i + \nu_i| \le \sqrt{2}\gamma_2|p_i|$ and $|\sigma_i| \le \mathsf{eps}|s_i| \le \sqrt{2}\gamma_2|s_i|$. Therefore,

$$(\widetilde{p_\pi + p_\mu + p_\nu) + \widetilde{p_\sigma}})(|x|) = \sum_{i=0}^{n-1} (|\pi_i + \mu_i + \nu_i| + |\sigma_i|)|x^i| \le \sum_{i=0}^{n-1} \left(\sqrt{2}\gamma_2|p_i||x^i|\right) + \sum_{i=0}^{n-1} \left(\sqrt{2}\gamma_2|s_i||x^i|\right).$$

We now transform the summation into

$$(\widetilde{p_\pi + p_\mu + p_\nu) + \widetilde{p_\sigma}})(|x|) \le \sqrt{2}\gamma_2 \sum_{i=1}^{n} (|p_{n-i}||x^{n-i}| + |s_{n-i}||x^{n-i}|)$$

and use the preceding equation (4.15) and the growth of the sequence $\widetilde{\gamma}_k$ so that

$$\begin{aligned}(\widetilde{p_\pi + p_\mu + p_\nu) + \widetilde{p_\sigma}}(|x|) &\leq \sqrt{2}\gamma_2 \sum_{i=1}^{n} ((1+\widetilde{\gamma}_{2i-1})\widetilde{p}(|x|) + (1+\widetilde{\gamma}_{2i})\widetilde{p}(|x|)) \\ &\leq \sqrt{2}\gamma_2 \sum_{i=1}^{n} 2(1+\widetilde{\gamma}_{2n})\widetilde{p}(|x|) = 2n\sqrt{2}\gamma_2(1+\widetilde{\gamma}_{2n})\widetilde{p}(|x|).\end{aligned}$$

Since $2n\sqrt{2}\gamma_2(1+\widetilde{\gamma}_{2n}) = \widetilde{\gamma}_{2n}$, we finally obtain $(\widetilde{p_\pi + p_\mu + p_\nu) + \widetilde{p_\sigma}}(|x|) \leq \widetilde{\gamma}_{2n}\widetilde{p}(|x|)$. □

From Theorem 4.2 the forward error affecting the evaluation of p at x according to the Horner scheme is

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

The coefficients of these polynomials are exactly computed by Algorithm 4.2, together with $\texttt{Horner}(p,x)$.

If we try to compute a complete error-free transformation for the evaluation of a polynomial of degree n, we will have to perform recursively the same computation for four polynomial of degree $n-1$ and so on. This will produce at the end of the computation $\sum_{i=0}^{n} 4^i = \frac{4^{n+1}-1}{4-1}$ error terms (for example for a polynomial of degree 10 we would obtain more than one million error terms), almost all of which are null with underflow and the other ones do not have the essential non-overlapping property. It will takes a very long time to compute this result (even more probably than with exact symbolic computation) and we will have to make a drastic selection on the huge amount of data to keep only a few meaningful terms as a usable result. We only consider here intentionally the first-order error term to obtain a really satisfactory improvement of the result of the evaluation with a reasonable running time.

Consequently we compute here a single complex floating point number as the first-order error term, the most significant correction term. The key is then to compute an approximate of the error $e(x)$ in working precision, and then to compute a corrected result $\texttt{res}' = \mathrm{fl}(\texttt{Horner}(p,x) + e(x))$.

Our aim is now to compute the correction term $c = \mathrm{fl}(e(x)) = \mathrm{fl}((p_\pi + p_\sigma + p_\mu + p_\nu)(x))$. For that we evaluate the polynomial P whose coefficients are those of $p_\pi + p_\sigma + p_\mu + p_\nu$ faithfully rounded[1] since the sums of the coefficients $p_i + q_i + r_i + s_i$ are not necessarily floating point numbers. We compute the coefficients of polynomial P thanks to Accsum algorithm [14]. This can also be done via other accurate summation algorithms (see [4] for example). We modify the classical Horner scheme applied to P, to compute P at the same time.

Algorithm 4.3. Evaluation of the sum of four polynomials with degree n

```
function $c = \texttt{HornerSumAcc}(p,q,r,s,x)$
  $v_n = \texttt{Accsum}(p_n + q_n + r_n + s_n)$
  for $i = n-1 : -1 : 0$
    $v_i = \mathrm{fl}(v_{i+1} \cdot x + \texttt{Accsum}(p_i + q_i + r_i + s_i))$
  end
  $c = v_0$
```

[1]Faithful rounding means that the computed result is equal to the exact result if the latter is a floating point number and otherwise is one of the two adjacent floating point numbers of the exact result.

Lemma 4.4. *Let us consider the floating point evaluation of* $(p+q+r+s)(x)$ *computed with* $\texttt{HornerSumAcc}(p,q,r,s,x)$*. Then, the computed result satisfies the following forward error bound,*

$$|\texttt{HornerSumAcc}(p,q,r,s,x) - (p+q+r+s)(x)| \leq \widetilde{\gamma}_{2n+1}(\widetilde{(p+q+r)} + \widetilde{s})(|x|),$$

Proof. We will use as in [7, p.68] the notation $\langle k\rangle$ to denote the product of k terms of the form $1+\varepsilon_i$ for some ε_i such that $|\varepsilon_i| \leq \sqrt{2}\gamma_2$. A product of j such terms multiplied by the product of k such terms is a product of $j+k$ such terms and consequently we have $\langle j\rangle\langle k\rangle = \langle j+k\rangle$.

Considering Algorithm 4.3, we have $v_n = \texttt{Accsum}(p_n+q_n+r_n+s_n)$ so according to the property of the `Accsum` algorithm we have $v_n = (p_n+q_n+r_n+s_n)\langle 1\rangle$.

For $i = n-1, \dots, 0$, the computation of v_i from v_{i+1} leads to an error term for the product and for the `Accsum` algorithm and then another on the sum and we have

$$v_i = \mathrm{fl}(v_{i+1}x + \texttt{Accsum}(p_i+q_i+r_i+s_i)) = v_{i+1}x\langle 2\rangle + (p_i+q_i+r_i+s_i)\langle 2\rangle.$$

Therefore we can prove by induction on i that

$$v_{n-i} = (p_n+q_n+r_n+s_n)x^i\langle 2i+1\rangle + \sum_{k=0}^{i-1}(p_{n-i+k}+q_{n-i+k}+r_{n-i+k}+s_{n-i+k})x^k\langle 2(k+1)\rangle$$

and then for $i = n$ we obtain

$$c = v_0 = (p_n+q_n+r_n+s_n)x^n\langle 2n+1\rangle + \sum_{k=0}^{n-1}(p_k+q_k+r_k+s_k)x^k\langle 2(k+1)\rangle.$$

Consequently we have

$$c - \sum_{i=0}^{n}(p_i+q_i+r_i+s_i)x^i = (p_n+q_n+r_n+s_n)x^n(\langle 2n+1\rangle - 1) + \sum_{k=0}^{n-1}(p_k+q_k+r_k+s_k)x^k(\langle 2(k+1)\rangle - 1).$$

Since for any ε implied in $\langle k\rangle$ notation, we have $|\varepsilon| \leq \sqrt{2}\gamma_2$, we have

$$|\langle k\rangle - 1| \leq (1+\sqrt{2}\gamma_2)^k - 1 \leq \frac{1}{1-k\sqrt{2}\gamma_2} - 1 = \frac{k\sqrt{2}\gamma_2}{1-k\sqrt{2}\gamma_2} = \widetilde{\gamma_k}$$

and the $\widetilde{\gamma_k}$ sequence is growing, thus $|\langle k\rangle - 1| \leq \widetilde{\gamma_k} \leq \widetilde{\gamma_{2n+1}}$ pour tout $k \leq 2n+1$. We finally obtain

$$\left|c - \sum_{i=0}^{n}(p_i+q_i+r_i+s_i)x^i\right| \leq \widetilde{\gamma}_{2n+1}\sum_{i=0}^{n}(|p_i+q_i+r_i| + |s_i|)|x^i| \leq \widetilde{\gamma}_{2n+1}(\widetilde{(p+q+r)} + \widetilde{s})(|x|).$$

□

We combine now the error-free transformation for the Horner scheme that produces four polynomials and the algorithm for the evaluation of the sum of four polynomials to obtain a compensated Horner scheme algorithm that improves the numerical accuracy of the classical Horner scheme on complex numbers.

Algorithm 4.4. Compensated Horner scheme

```
function $\texttt{res'} = \texttt{CompHorner}(p,x)$
  $[\texttt{res}, p_\pi, p_\mu, p_\nu, p_\sigma] = \texttt{EFTHorner}(p,x)$
  $c = \texttt{HornerSumAcc}(p_\pi, p_\mu, p_\nu, p_\sigma, x)$
  $\texttt{res'} = \mathrm{fl}(\texttt{res} + c)$
```

We prove hereafter that the result of a polynomial evaluation computed with the compensated Horner scheme (4.4) is as accurate as if computed by the classic Horner scheme using twice the working precision and then rounded to the working precision.

Theorem 4.5. *Given a polynomial $p = \sum_{i=0}^{n} p_i x^i$ of degree n with floating point coefficients, and x a floating point value. We consider the result* $\texttt{CompHorner}(p,x)$ *computed by Algorithm 4.4. Then,*

$$|\texttt{CompHorner}(p,x) - p(x)| \le \mathsf{eps}|p(x)| + \widetilde{\gamma}_{2n}^2 \widetilde{p}(|x|), \tag{4.16}$$

Proof. As $\texttt{res}' = \mathrm{fl}(\texttt{res} + c)$ so, according to Theorem 3.1, $\texttt{res}' = (1+\varepsilon)(\texttt{res} + c)$ with $|\varepsilon| \le \mathsf{eps} \le \sqrt{2}\gamma_2$. Thus we have $|\texttt{res}' - p(x)| = |\,\mathrm{fl}(\texttt{res}+c) - p(x)| = |(1+\varepsilon)(\texttt{res}+c-p(x)) + \varepsilon p(x)|$. Since $p(x) = \texttt{res} + e(x)$, we have $|\texttt{res}' - p| = |(1+\varepsilon)(c - e(x)) + \varepsilon p(x)| \le \mathsf{eps}|p(x)| + (1+\mathsf{eps})|e(x) - c|$. By Lemma 4.4 applied to four polynomials of degree $n-1$, we have

$$|e(x) - c| \le \widetilde{\gamma}_{2n-1}((\widetilde{p_\pi + p_\mu + p_\nu}) + \widetilde{p_\sigma})(|x|).$$

By Proposition 4.3 we have also $((\widetilde{p_\pi + p_\mu + p_\nu}) + \widetilde{p_\sigma})(|x|) \le \widetilde{\gamma}_{2n}\widetilde{p}(|x|)$. We combine these two bounds and obtain $|e(x) - c| \le \widetilde{\gamma}_{2n-1}\widetilde{\gamma}_{2n}\widetilde{p}(|x|)$. As a consequence, $|\texttt{res}' - p(x)| \le \mathsf{eps}|p(x)| + (1+\sqrt{2}\gamma_2)\widetilde{\gamma}_{2n-1}\widetilde{\gamma}_{2n}\widetilde{p}(|x|)$. Since $(1+\sqrt{2}\gamma_2)\widetilde{\gamma}_{2n-1} \le \widetilde{\gamma}_{2n}$, it follows that $|\texttt{res}' - p(x)| \le \mathsf{eps}|p(x)| + \widetilde{\gamma}_{2n}^2\widetilde{p}(x)$. □

4.3 Numerical experiments

Equation (4.16) can be written

$$\frac{|\texttt{CompHorner}(p,x) - p(x)|}{|p(x)|} \le \mathsf{eps} + \widetilde{\gamma}_{2n}^2 \operatorname{cond}(p,x). \tag{4.17}$$

The comparison with the bound (4.11) for the classical Horner scheme shows that the coefficient of the condition number vanish from $\widetilde{\gamma}_{2n}$ to $\widetilde{\gamma}_{2n}^2$.

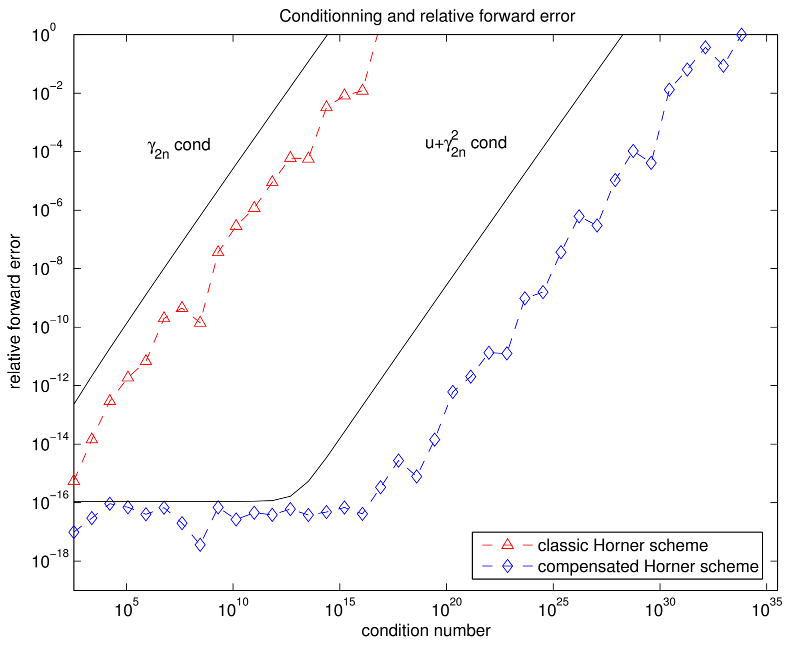
We present here comparison curves for the classical and the compensated Horner scheme.

All our experiments are performed using the IEEE 754 double precision with Matlab 7. When needed, we use the Symbolic Math Toolbox to accurately compute the polynomial evaluation (in order to compute the relative forward error).

We test the compensated Horner scheme on the expanded form of the polynomial $p_n(x) = (x - (1+i))^n$ at $x = \mathrm{fl}(1.333 + 1.333i)$ for $n = 3 : 42$. The condition number $\operatorname{cond}(p_n, x)$ varies from 10^3 to 10^{33}.

The following figure shows the relative accuracy $|\texttt{res} - p_n(x)|/|p_n(x)|$ where $\texttt{res}$ is the computed value by the two algorithms 4.1 and 4.4. We also plot the *a priori* error estimation (4.11) and (4.17).

As we can see below, the compensated Horner scheme exhibits the expected behavior, that is to say, the compensated rule of thumb (4.17). As long as the condition number is less than $\mathsf{eps}^{-1} \approx 10^{16}$, the compensated Horner scheme produces results with full precision (forward relative error of the order of $\mathsf{eps} \approx 10^{-16}$). For condition numbers greater than $\mathsf{eps}^{-1} \approx 10^{16}$, the accuracy decreases until no accuracy at all when the condition number is greater than $\mathsf{eps}^{-2} \approx 10^{32}$.



5 Conclusion and future work

In this article, we derived some new error-free transformations for complex floating point arithmetic. This makes it possible to provide a complex version of the compensated Horner scheme.

Nevertheless, the error bound provided in this article is a theoretical one since it contains the quantity $|p(x)|$. It would be very interesting to derive a validated error bound $\alpha \in \mathbb{F}$ that can be computed in floating point arithmetic satisfying $|\texttt{CompHorner}(p, x) - p(x)| \leq \alpha$. This can be done via a kind of running error analysis [15].

References

[1] Sylvie Boldo and Jean-Michel Muller. Some functions computable with a Fused-mac. In *Proceedings of the 17th Symposium on Computer Arithmetic*, Cape Cod, USA, 2005.

[2] Richard Brent, Colin Percival, and Paul Zimmermann. Error bounds on complex floating-point multiplication. *Math. Comp.*, 76(259):1469–1481 (electronic), 2007.

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

[4] James W. Demmel and Yozo Hida. Accurate and efficient floating point summation. *SIAM J. Sci. Comput.*, 25(4):1214–1248 (electronic), 2003.

[5] Stef Graillat, Nicolas Louvet, and Philippe Langlois. Compensated Horner scheme. Research Report 04, Équipe de recherche DALI, Laboratoire LP2A, Université de Perpignan Via Domitia, France, July 2005.

[6] Stef Graillat and Valérie Menissier-Morain. Error-free transformations in real and complex floating point arithmetic. In *Proceedings of the International Symposium on Nonlinear Theory and its Applications*, pages 341–344, Vancouver, Canada, September 16-19, 2007.

[7] Nicholas J. Higham. *Accuracy and stability of numerical algorithms.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second edition, 2002.

[8] *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985.* Institute of Electrical and Electronics Engineers, New York, 1985. Reprinted in SIGPLAN Notices, 22(2):9–25, 1987.

[9] Donald E. Knuth. *The Art of Computer Programming, Volume 2, Seminumerical Algorithms.* Addison-Wesley, Reading, MA, USA, third edition, 1998.

[10] 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. Softw.*, 28(2):152–205, 2002.

[11] Yves Nievergelt. Scalar fused multiply-add instructions produce floating-point matrix arithmetic provably accurate to the penultimate digit. *ACM Trans. Math. Software*, 29(1):27–48, 2003.

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

[13] S. M. Rump. Verification of positive definiteness. *BIT*, 46(2):433–452, 2006.

[14] Siegfried M. Rump, Takeshi Ogita, and Shin'ichi Oishi. Accurate floating-point summation. Technical Report 05.12, Faculty for Information and Communication Sciences, Hamburg University of Technology, nov 2005.

[15] James H. Wilkinson. *Rounding errors in algebraic processes.* Prentice-Hall Inc., Englewood Cliffs, N.J., 1963.