Error-free transformations in real and complex floating point arithmetic

Stef Graillat[†] and Valérie Ménissier-Morain[†]

†Laboratoire LIP6, Département Calcul Scientifique
Université Pierre et Marie Curie (Paris 6)
4 place Jussieu, F-75252, Paris cedex 05, France
Email: Stef.Graillat@lip6.fr, Valerie.Menissier-Morain@lip6.fr

Abstract— Error-free transformation is a concept that makes it possible to compute accurate results within a floating point arithmetic. Up to now, it has only be studied for real floating point arithmetic. In this short note, we recall the known error-free transformations for real arithmetic and we propose some new error-free transformations for complex floating point arithmetic. This will make it possible to design some new accurate algorithms for summation, dot product and polynomial evaluation with complex entries.

1. Introduction

It is well-known that computing with finite precision implies some rounding errors. These errors leads often to inexact results for a computation. An important tool to try to avoid this are the *error-free transformations*: a floating point approximation plus an exact error term. This can be view as a *double-double* floating point numbers [10] but without the renormalisation step. For classical problems such that summation, dot product and polynomial evaluation this step is not necessary.

Error-free transformations have been widely used to provide some new accurate algorithms in floating point arithmetic (see [12, 14] for accurate sum and dot product and [4] for polynomial evaluation).

Complex error-free transformations are the first step for providing accurate algorithms using complex numbers.

The rest of the paper is organised as follows. In Section 2, we recall some results on real floating point arithmetic and on the error-free transformations. In Section 3, we present the complex floating point arithmetic and we propose some new error-free transformations for this arithmetic.

2. Real floating point arithmetic

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

2.1. Definitions and notations

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 occurs, but allow underflow. The set of floating point numbers is denoted by $\mathbb{F}$, the relative rounding error by $\mathtt{eps}$ and the underflow unit by $\mathtt{eta}$. For IEEE 754 double precision, we have $\mathtt{eps} = 2^{-53}$ and $\mathtt{eta} = 2^{-1074}$.

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]

$$\mathrm{fl}(a \circ b) = (a \circ b)(1+\varepsilon_1) = (a \circ b)/(1+\varepsilon_2) \quad \text{for } \circ \in \{+,-\} \text{ and } |\varepsilon_\nu| \leq \mathtt{eps}$$

and

$$\mathrm{fl}(a \circ b) = (a \circ b)(1+\varepsilon_1) + \eta_1 = (a \circ b)/(1+\varepsilon_2) + \eta_2 \quad \text{for } \circ \in \{\cdot,/\} \text{ and } |\varepsilon_\nu| \leq \mathtt{eps},\ |\eta_\nu| \leq \mathtt{eta}.$$

Addition and subtraction are exact in case of underflow [6], and $\varepsilon_1\eta_1 = \varepsilon_2\eta_2 = 0$ for multiplication and division [7, p.56]. This implies that

$$\begin{aligned} &|a \circ b - \mathrm{fl}(a \circ b)| \leq \mathtt{eps}|a \circ b| \text{ and} \\ &|a \circ b - \mathrm{fl}(a \circ b)| \leq \mathtt{eps}|\,\mathrm{fl}(a \circ b)| \text{ for } \circ \in \{+,-\} \end{aligned} \quad (1)$$

and

$$\begin{aligned} &|a \circ b - \mathrm{fl}(a \circ b)| \leq \mathtt{eps}|a \circ b| + \mathtt{eta} \text{ and} \\ &|a \circ b - \mathrm{fl}(a \circ b)| \leq \mathtt{eps}|\,\mathrm{fl}(a \circ b)| + \mathtt{eta} \text{ for } \circ \in \{\cdot,/\}. \end{aligned} \quad (2)$$

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

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

where we implicitly assume that $n\mathtt{eps} \leq 1$.

2.2. Error-free transformations in real floating point arithmetic

One can notice that for a and b in $\mathbb{F}$, $a \circ b \in \mathbb{R}$ 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]):

$$x = \mathrm{fl}(a \circ b) \Rightarrow a \circ b = x + y \text{ with } \circ \in \{+, -, \cdot\} \text{ and } y \in \mathbb{F}, \tag{3}$$

where no underflow is assumed for multiplication.These are *error-free* transformations of the pair (a, b) into the pair (x, y).

Fortunately, the quantities x and y in (3) can be computed exactly in floating point arithmetic. For the algorithms, we use Matlab-like notations.

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 $[x, y]$ = 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 $|a| \geq |b|$.

Algorithm 2.2 (Dekker [3]). *Error-free transformation of the sum of two floating point numbers.*

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

For the error-free transformation of a product, we first need to split the input argument into two parts. Let p be given by $\texttt{eps} = 2^{-p}$ and define $s = \lceil p/2 \rceil$. For example, if the working precision is IEEE 754 double precision, then $p = 53$ and $s = 27$. The following algorithm by Dekker [3] splits a floating point number $a \in \mathbb{F}$ into two parts x and y such that

$$a = a_1 + a_2 \text{ and } a_1 \text{ and } a_2 \text{ non overlapping with } |a_2| \leq |a_1|.$$

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

```
function $[a_1, a_2]$ = Split$(a)$
  factor $= \mathrm{fl}(2^s + 1)$
  $c = \mathrm{fl}(\texttt{factor} \cdot a)$
  $a_1 = \mathrm{fl}(c - (c - a))$
  $a_2 = \mathrm{fl}(a - a_1)$
```

With this function, an algorithm from Veltkamp (see [3]) enables to compute an error-free transformation for the product of two floating point numbers. 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 $[x, y]$ = TwoProduct$(a, b)$
  $x = \mathrm{fl}(a \cdot b)$
  $[a_1, a_2]$ = Split$(a)$
  $[b_1, b_2]$ = Split$(b)$
  $y = \mathrm{fl}(a_2 \cdot b_2 - (((x - a_1 \cdot b_1) - a_2 \cdot b_1) - a_1 \cdot b_2))$
```

In case of underflow of any of the five multiplications in Algorithm 2.4, the error y' is no more a floating point number but we have $|y' - y| \leq 5\,\texttt{eta}$.

The following theorem summarises the properties of algorithms TwoSum and TwoProduct.

Theorem 2.1 (Ogita, Rump and Oishi [12]). *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, also in the presence of underflow,*

$$a + b = x + y, \quad x = \mathrm{fl}(a + b), \quad |y| \leq \texttt{eps}|x|, \quad |y| \leq \texttt{eps}|a + b|.$$

The algorithm TwoSum *requires* 6 *flops.*
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, if no underflow occurs,

$$a \cdot b = x + y, \quad x = \mathrm{fl}(a \cdot b), \quad |y| \leq \texttt{eps}|x|, \quad |y| \leq \texttt{eps}|a \cdot b|,$$

and in the presence of underflow

$$a \cdot b = x + y + 5\eta, \quad x = \mathrm{fl}(a \cdot b), \quad |y| \leq \texttt{eps}|x| + 5\texttt{eta}, \quad |y| \leq \texttt{eps}|a \cdot b| + 5\,\texttt{eta},$$

with $|\eta| \leq \texttt{eta}$. The algorithm TwoProduct *requires* 17 *flops.*

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 satisfies

$$\texttt{FMA}(a, b, c) = (a \cdot b + c)(1 + \varepsilon_1) + \eta_1 = (a \cdot b + c)/(1 + \varepsilon_2) + \eta_2 \quad \text{with } |\varepsilon_\nu| \leq \texttt{eps}, \ |\eta_\nu| \leq \texttt{eta}.$$

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

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

In the same way, if we suppose that we have $\texttt{ADD3}(a, b, c)$ which computes the nearest floating point number of the exact sum $a + b + c \in \mathbb{R}$ for $a, b, c \in \mathbb{F}$ then the algorithm TwoSum can be replaced by the following one.

Algorithm 2.6 (Ogita, Rump and Oishi [12]). *Error-free transformation of the sum of two floating point numbers using* ADD3.

```
function $[x, y] = \texttt{TwoSumADD3}(a, b)$
    $x = \mathrm{fl}(a + b)$
    $y = \texttt{ADD3}(a, b, -x)$
```

An error-free algorithm for the FMA has been recently given by Boldo and Muller [1]. The error cannot be represented by a single floating-point number anymore. It is now the sum of two floating point numbers.

Algorithm 2.7 (Boldo and Muller [1]). *Error-free transformation of the* FMA *of three floating point numbers.*

```
function $[x, y, z] = \texttt{ThreeFMA}(a, b, c)$
    $x = \texttt{FMA}(a, b, c)$
    $[u_1, u_2] = \texttt{TwoProductFMA}(a, b)$
    $[\alpha_1, z] = \texttt{TwoSum}(c, u_2)$
    $[\beta_1, \beta_2] = \texttt{TwoSum}(u_1, \alpha_1)$
    $y = \mathrm{fl}((\beta_1 - x) + \beta_2)$
```

It follows easily from [1] that the following proposition holds.

Proposition 2.2. *Let $a, b, c \in \mathbb{F}$ and let $x, y, z \in \mathbb{F}$ such that $[x, y, z] = \texttt{ThreeFMA}(a, b, c)$ (Algorithm 2.7). Then, if no underflow occurs,*

$$a{\cdot}b{+}c = x{+}y{+}z, \quad x = \mathrm{fl}(a{\cdot}b{+}c), \quad |y{+}z| \le \texttt{eps}|x|,$$
$$|y + z| \le \texttt{eps}|a \cdot b + c|, \quad y = 0 \text{ or } |y| > |z|,$$

and in the presence of underflow

$$a \cdot b + c = x + y + z + \eta, \quad x = \mathrm{fl}(a \cdot b + c),$$
$$|y{+}z| \le \texttt{eps}|x|{+}\texttt{eta}, \quad |y{+}z| \le \texttt{eps}|a{\cdot}b{+}c|{+}\texttt{eta},$$

and $y = 0$ or $|y| > |z|$ with $|\eta| \le$ eta. The algorithm ThreeFMA *requires* 17 *flops.*

3. 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 complex floating point computation, where all operations inside parentheses are done in floating point working precision. One can show [7, 13] that for $x, y \in \mathbb{F} + i\mathbb{F}$,

$$\mathrm{fl}(x \circ y) = (x \circ y)(1 + \varepsilon_1) = (x \circ y)/(1 + \varepsilon_2),$$
$$\text{for } \circ \in \{+, -\} \text{ and } |\varepsilon_\nu| \le \texttt{eps}, \quad (4)$$

and

$$\mathrm{fl}(x \cdot y) = (x \cdot y)(1 + \varepsilon_1) + \eta_1,$$
$$|\varepsilon_1| \le \sqrt{2}\gamma_2, \quad |\eta_1| \le 2\sqrt{2}\texttt{eta}. \quad (5)$$

It follows that we have

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

and

$$|x \cdot y - \mathrm{fl}(x \cdot y)| \le \sqrt{2}\gamma_2|x \cdot y| + 2\sqrt{2}\texttt{eta}.$$

For the complex multiplication, the term $\sqrt{2}\gamma_2$ can advantageously be replaced by $\sqrt{5}\texttt{eps}$ which is nearly optimal (see [2]).

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, also in the presence of underflow,*

$$x + y = s + e, \quad s = \mathrm{fl}(x + y), \quad |e| \le \texttt{eps}|s|,$$
$$|e| \le \texttt{eps}|x + y|.$$

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 (4), we derive that $|e| \le \texttt{eps}|s|$ and $|e| \le \texttt{eps}|x + y|$. □

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 $[p, e, f, g] = \texttt{TwoProductCplx}(x, y)$ (Algorithm 2.4). Then, if no underflow occurs,*

$$x{\cdot}y = p{+}e{+}f{+}g \quad p = \mathrm{fl}(x{\cdot}y), \quad |e{+}f{+}g| \le \sqrt{2}\gamma_2|x{\cdot}y|,$$

and in the presence of underflow

$$x \cdot y = p + e + f + g + \eta, \quad p = \mathrm{fl}(x \cdot y),$$
$$|e + f + g| \le \sqrt{2}\gamma_2 |x \cdot y| + 10\sqrt{2}\mathtt{eta},$$

with $|\eta| \le 10\sqrt{2}\mathtt{eta}$. *The algorithm* `TwoProductCplx` *requires* 80 *flops.*

Proof. If no underflow occurs, from Theorem 2.1, it holds that $z_1 + h_1 = a \cdot c$, $z_2 + h_2 = b \cdot d$, $z_3 + h_3 = a \cdot d$, $z_4 + h_3 = b \cdot c$, $z_5 + h_5 = z_1 - z_2$ 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 $p = \mathrm{fl}(x \cdot y)$. From (5), we deduce that $|e + f + g| = |x \cdot y - \mathrm{fl}(x \cdot y)| \le \sqrt{2}\gamma_2 |x \cdot y|$. In case of underflow, each use of `TwoProduct` produces a additional error of at most 5η so that the use of two `TwoProduct` both for the real and imaginary part produces an error of at most $10\sqrt{2}\mathtt{eta}$. □

If one use `TwoProductFMA` instead of `TwoProduct`, the numbers of flops falls down to 20, thus the availability of an `FMA` is crucial for fast error-free transformations in complex arithmetic.

4. Conclusion and future work

The aim for deriving error-free transformations is to provide some accurate algorithms for complex entries. The solution to split the arguments and the results into their real and imaginary parts is sufficient for summation or dot product but not for the polynomial evaluation. That is why these new error-free transformations for complex floating point arithmetic will make it possible to derive complex versions of the accurate algorithms to compute sum, dot product and evaluation of polynomials [12, 14, 4]. This description, and the comparison with the splitting approach for operations where it is well suited, will be done in a future paper [5].

As we mentioned before, `TwoProductCplxFMA` is really more efficient than `TwoProductCplx`. The use of the `FMA` seems to be really important when available.

For the multiplication, the operation $ab + cd$ is done both for the real and imaginary part. The presence of an operation `FTMA` (Fused Two Multiply and Add) computing $ab + cd$ with only one rounding would be very useful.

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] 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.

[5] Stef Graillat and Valérie Ménissier-Morain. Accurate summation, dot product and polynomial evaluation in complex floating point arithmetic. Technical report, LIP6, Université Pierre et Marie Curie (Paris 6), 2007. in preparation.

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

[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.