Accurate Floating Point Product

Stef Graillat
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

October 26, 2007

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 the product of floating point numbers. 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 the determinant of a triangular matrix and to evaluate a polynomial when represented by the root product form. It can also be used to compute the power of a floating point number.

Keywords: accurate product, exponentiation, finite precision, floating point arithmetic, faithful rounding, error-free transformations

AMS Subject Classification: 65-04, 65G20, 65G50

1. Introduction

In this paper, we present fast and accurate algorithms to compute the product of floating point numbers. Our aim is to increase the accuracy at a fixed precision. We show that the results have the same error estimates as if computed in twice the working precision and then rounded to working precision. Then we address the problem on how to compute a faithfully rounded result, that is to say one of the two adjacent floating point numbers of the exact result.

This paper was motived by papers (Ogita et al., 2005a; Rump et al., 2005; Graillat et al., 2005; Langlois and Louvet, 2007) and (Kornerup et al., 2007) where similar approaches are used to compute summation, dot product, polynomial evaluation and power.

The applications of our algorithms are multiple. One of the examples frequently used in Sterbenz's book (Sterbenz, 1974) is the computation of the product of some floating point numbers. Our algorithms can be used to compute the determinant of a triangle matrix. Another application is for evaluating a polynomial when represented by the root product form $p(x) = a_n \prod_{i=1}^{n}(x - x_i)$. We can also apply our algorithms to compute the power of a floating point number.

The rest of the paper is organized as follows. In Section 2, we recall notations and auxiliary results that will be needed in the sequel. We present the floating point arithmetic and the so-called error-free transformations. In Section 3, we present a classic algorithm to compute the product

of floating point numbers. We give an error estimate as well as a validated error bound. We also present a new compensated algorithm together with an error estimate and a validated error bound. We show that under mild assumptions, our algorithm gives a faithfully rounded result.

2. Notation and auxiliary results

2.1. Floating point arithmetic

Throughout the paper, we assume to work with a floating point arithmetic adhering to IEEE 754 floating point standard in rounding to nearest (IEEE Computer Society, 1985). We assume that no overflow nor underflow occur. The set of floating point numbers is denoted by $\mathbb{F}$, the relative rounding error by $\texttt{eps}$. For IEEE 754 double precision, we have $\texttt{eps} = 2^{-53}$ and for single precision $\texttt{eps} = 2^{-24}$.

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 (Higham, 2002)

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

This implies that

$$|a \circ b - \mathrm{fl}(a \circ b)| \leq \texttt{eps}|a \circ b| \text{ and } |a \circ b - \mathrm{fl}(a \circ b)| \leq \texttt{eps}|\,\mathrm{fl}(a \circ b)| \text{ for } \circ = \{+,-,\cdot,/\}. \tag{1}$$

2.2. Error-free transformations

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 (Dekker, 1971)):

$$\begin{array}{lll} x = \mathrm{fl}(a \pm b) & \Rightarrow & a \pm b = x + y \quad \text{with } y \in \mathbb{F}, \\ x = \mathrm{fl}(a \cdot b) & \Rightarrow & a \cdot b = x + y \quad \text{with } y \in \mathbb{F}. \end{array} \tag{2}$$

These are *error-free* transformations of the pair (a, b) into the pair (x, y).

Fortunately, the quantities x and y in (2) 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 (Knuth, 1998, Thm B. p.236).

ALGORITHM 2.1 (Knuth (Knuth, 1998)). *Error-free transformation of the sum of two floating point numbers*

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

Another algorithm to compute an error-free transformation is the following algorithm from Dekker (Dekker, 1971). 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 (Dekker, 1971)). *Error-free transformation of the sum of two floating point numbers with* $|a| \geq |b|$.

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

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 (Dekker, 1971) splits a floating point number $a \in \mathbb{F}$ into two parts x and y such that

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

ALGORITHM 2.3 (Dekker (Dekker, 1971)). *Error-free split of a floating point number into two parts*

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

With this function, an algorithm from Veltkamp (see (Dekker, 1971)) makes it possible 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 (Dekker, 1971)). *Error-free transformation of the product of two floating point numbers*

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

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

THEOREM 2.1 (Ogita, Rump and Oishi (Ogita et al., 2005a)). *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,*

$$a + b = x + y, \quad x = \mathrm{fl}(a + b), \quad |y| \leq \texttt{eps}|x|, \quad |y| \leq \texttt{eps}|a + b|. \tag{3}$$

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,

$$a \cdot b = x + y, \quad x = \mathrm{fl}(a \cdot b), \quad |y| \leq \texttt{eps}|x|, \quad |y| \leq \texttt{eps}|a \cdot b|. \tag{4}$$

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

3. Accurate floating point product

In this section, we present a new accurate algorithm to compute the product of floating point numbers. In Subsection 3.1, we recall the classic method and we give a theoretical error bound as well as a validated computable error bound. In Subsection 3.2, we present our new algorithm based on a compensated scheme together with a theoretical error bound. In Subsection 3.3, we give sufficient conditions on the number of floating point numbers so as to get a faithfully rounded result. Finally, in Subsection 3.4, we give a validated computable error bound for our new algorithm.

3.1. Classic method

The classic method for evaluating a product of n numbers $a = (a_1, a_2, \ldots, a_n)$

$$p = \prod_{i=1}^{n} a_i$$

is the following algorithm.

ALGORITHM 3.1. *Product evaluation*

function $\texttt{res} = \texttt{Prod}(a)$
  $p_1 = a_1$
  for $i = 2 : n$
    $p_i = \mathrm{fl}(p_{i-1} \cdot a_i)$
  end
  $\texttt{res} = p_n$

This algorithm requires $n - 1$ flops. Let us now analyse its accuracy.

We will use standard notations and standard results for the following error estimations (see (Higham, 2002)). The quantities γ_n are defined as usual (Higham, 2002) by

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

When using γ_n, we implicitly assume that $n\mathtt{eps} \leq 1$. A forward error bound is

$$|a_1 a_2 \cdots a_n - \mathtt{res}| = |a_1 a_2 \cdots a_n - \mathrm{fl}(a_1 a_2 \cdots a_n)| \leq \gamma_{n-1} |a_1 a_2 \cdots a_n|. \tag{5}$$

Indeed, by induction,

$$\mathtt{res} = \mathrm{fl}(a_1 a_2 \cdots a_n) = a_1 a_2 \cdots a_n (1+\varepsilon_2)(1+\varepsilon_3)\cdots(1+\varepsilon_n), \tag{6}$$

with $\varepsilon_i \leq \mathtt{eps}$ for $i = 2 : n$. It follows from Lemma 3.1 of (Higham, 2002, p.63) that $(1+\varepsilon_2)(1+\varepsilon_3)\cdots(1+\varepsilon_n) = 1+\theta_n$ where $|\theta_{n-1}| \leq \gamma_{n-1}$.

A convenient device for keeping track of power of $1+\varepsilon$ term is described in (Higham, 2002, p.68). The relative error counter $\langle k \rangle$ denotes the product

$$\langle k \rangle = \prod_{i=1}^{k} (1+\varepsilon_i), \quad |\varepsilon_i| \leq \mathtt{eps}.$$

A useful rule for the counter is $\langle j \rangle \langle k \rangle = \langle j+k \rangle$. Using this notation, Equation (6) can be written $\mathtt{res} = \mathrm{fl}(a_1 a_2 \cdots a_n) = a_1 a_2 \cdots a_n \langle n-1 \rangle$.

It is shown in (Ogita et al., 2005b) that for $a \in \mathbb{F}$, we have

$$(1+\mathtt{eps})^n \;\leq\; \frac{1}{(1-\mathtt{eps})^n} \leq \frac{1}{1-n\mathtt{eps}}, \tag{7}$$

$$\frac{|a|}{1-n\mathtt{eps}} \;\leq\; \mathrm{fl}\left(\frac{|a|}{1-(n+1)\mathtt{eps}}\right). \tag{8}$$

¿From Equation (6), it follows that

$$|a_1 a_2 \cdots a_n - \mathtt{res}| \leq (1+\mathtt{eps})^{n-1} \gamma_{n-1} |\mathtt{res}|.$$

If $m\mathtt{eps} \leq 1$ for $m \in \mathbb{N}$, $\mathrm{fl}(m\mathtt{eps}) = m\mathtt{eps}$ and $\mathrm{fl}(1-m\mathtt{eps}) = 1-m\mathtt{eps}$. Therefore,

$$\gamma_m \leq (1+\mathtt{eps})\,\mathrm{fl}(\gamma_m). \tag{9}$$

Hence,

$$\begin{aligned} |a_1 a_2 \cdots a_n - \mathtt{res}| \;&\leq\; (1+\mathtt{eps})^n\,\mathrm{fl}(\gamma_{n-1})|\mathtt{res}| \\ &\leq\; (1+\mathtt{eps})^{n+1}\,\mathrm{fl}(\gamma_{n-1}|\mathtt{res}|), \end{aligned}$$

and so

$$|a_1 a_2 \cdots a_n - \mathtt{res}| \leq \mathrm{fl}\left(\frac{\gamma_{n-1}|\mathtt{res}|}{1-(n+2)\mathtt{eps}}\right).$$

The previous inequality gives us a validated error bound that can be computed in pure floating point arithmetic in rounding to nearest.

3.2. Compensated method

We present hereafter a compensated scheme to evaluate the product of floating point numbers, i.e. the error of individual multiplication is somehow corrected. The technique used here is based on the paper (Ogita et al., 2005a).

ALGORITHM 3.2. *Product evaluation with a compensated scheme*

function `res` = `CompProd`(a)
  $p_1 = a_1$
  $e_1 = 0$
  for $i = 2 : n$
    $[p_i, \pi_i] =$ `TwoProduct`(p_{i-1}, a_i)
    $e_i = \mathrm{fl}(e_{i-1}a_i + \pi_i)$
  end
  `res` $= \mathrm{fl}(p_n + e_n)$

This algorithm requires $19n - 18$ flops. For error analysis, we note that

$$p_n = \mathrm{fl}(a_1 a_2 \cdots a_n) \quad \text{and} \quad e_n = \mathrm{fl}\left(\sum_{i=2}^{n} \pi_i a_{i+1} \cdots a_n\right).$$

We also have

$$p = a_1 a_2 \ldots a_n = \mathrm{fl}(a_1 a_2 \ldots a_n) + \sum_{i=2}^{n} \pi_i a_{i+1} \cdots a_n = p_n + e, \tag{10}$$

where $e = \sum_{i=2}^{n} \pi_i a_{i+1} \cdots a_n$.

Before proving the main theorem, we will need two technical lemmas. The next lemma makes it possible to obtain a bound on the individual error of the multiplication namely π_i in function of the inital data a_i.

LEMMA 3.1. *Suppose floating point numbers $\pi_i \in \mathbb{F}$, $2 \leq i \leq n$ are computed by the following algorithm*

$p_1 = a_1$
for $i = 2 : n$
  $[p_i, \pi_i] =$ `TwoProduct`(p_{i-1}, a_i)
end

Then,

$$|\pi_i| \leq \mathtt{eps}(1 + \gamma_{i-1})|a_1 \cdots a_i| \quad \textit{for } i = 2 : n.$$

Proof. From Equation (1), it follows that

$$|\pi_i| \leq \mathtt{eps}|p_i|.$$

Moreover, $p_i = \mathrm{fl}(a_1 \cdots a_i)$ so that from (5),

$$|p_i| \leq (1 + \gamma_{i-1})|a_1 \cdots a_i|.$$

Hence, $|\pi_i| \leq \mathtt{eps}(1 + \gamma_{i-1})|a_1 \cdots a_i|$. □

The following lemma enables us to bound the rounding errors during the computation of the error during the full product.

LEMMA 3.2. *Suppose floating point numbers $e_i \in \mathbb{F}$, $1 \leq i \leq n$ are computed by the following algorithm*

$e_1 = 0$
for $i = 2 : n$
  $[p_i, \pi_i] = \texttt{TwoProduct}(p_{i-1}, a_i)$
  $e_i = \mathrm{fl}(e_{i-1}a_i + \pi_i)$
end

Then,

$$|e_n - \sum_{i=2}^{n} \pi_i a_{i+1} \cdots a_n| \leq \gamma_{n-1}\gamma_{2n}|a_1 a_2 \cdots a_n|.$$

Proof. First, one notices that $e_n = \mathrm{fl}(\sum_{i=2}^{n}(\pi_i a_{i+1} \cdots a_n))$. We will use the error counters described above. For n floating point numbers x_i, it is easy to see that (Higham, 2002, chap.4)

$$\mathrm{fl}(x_1 + x_2 + \cdots + x_n) = x_1\langle n-1\rangle + x_2\langle n-1\rangle + x_3\langle n-2\rangle + \cdots + x_n\langle 1\rangle.$$

This implies that

$$e_n = \mathrm{fl}(\sum_{i=2}^{n}(\pi_i a_{i+1} \cdots a_n)) = \mathrm{fl}(\pi_2 a_3 \cdots a_n)\langle n-2\rangle + \mathrm{fl}(\pi_3 a_4 \cdots a_n)\langle n-2\rangle + \cdots + \mathrm{fl}(\pi_n)\langle 1\rangle.$$

Furthermore, we have shown before that $\mathrm{fl}(a_1 a_2 \cdots a_n) = a_1 a_2 \cdots a_n\langle n-1\rangle$. Consequently,

$$e_n = \pi_2 a_3 \cdots a_n\langle n-2\rangle\langle n-1\rangle + \pi_3 a_4 \cdots a_n\langle n-3\rangle\langle n-1\rangle + \cdots + \pi_n\langle 1\rangle.$$

A straightforward computation yields

$$|e_n - \sum_{i=2}^{n} \pi_i a_{i+1} \cdots a_n| \leq \gamma_{2n-3} \sum_{i=2}^{n} |\pi_i a_{i+1} \cdots a_n|.$$

¿From Lemma 3.1, we have $|\pi_i| \leq \texttt{eps}(1 + \gamma_{i-1})|a_1 \cdots a_i|$ and hence

$$|e_n - \sum_{i=2}^{n} \pi_i a_{i+1} \cdots a_n| \leq (n-1)\texttt{eps}(1 + \gamma_{n-1})\gamma_{2n-3}|a_1 a_2 \cdots a_n|.$$

Since $\texttt{eps}(1 + \gamma_{n-1}) = \gamma_{n-1}/(n-1)$ and $\gamma_{2n-3} \leq \gamma_{2n}$, we obtain the desired result. □

One may notice that the computation of e_n is similar to the Horner scheme. One could have directly applied a result on the error of the Horner scheme (Higham, 2002, Eq.(5.3),p.95).

We can finally state the main theorem.

THEOREM 3.3. *Suppose Algorithm 3.2 is applied to floating point number $a_i \in \mathbb{F}$, $1 \leq i \leq n$, and set $p = \prod_{i=1}^{n} a_i$. Then,*

$$|\texttt{res} - p| \leq \texttt{eps}|p| + \gamma_n\gamma_{2n}|p|.$$

Proof. The fact that $\texttt{res} = \mathrm{fl}(p_n + e_n)$ implies that $\texttt{res} = (1+\varepsilon)(p_n + e_n)$ with $\varepsilon \leq \texttt{eps}$. So it follows

$$\begin{aligned}
|\texttt{res} - p| &= |\,\mathrm{fl}(p_n + e_n) - p| = |(1+\varepsilon)(p_n + e_n - p) + \varepsilon p| \\
&= |(1+\varepsilon)(p_n + \sum_{i=2}^{n} \pi_i a_{i+1}\cdots a_n - p) + (1+\varepsilon)(e_n - \sum_{i=2}^{n} \pi_i a_{i+1}\cdots a_n) + \varepsilon p| \\
&= |(1+\varepsilon)(e_n - \sum_{i=2}^{n} \pi_i a_{i+1}\cdots a_n) + \varepsilon p| \quad \text{by (10)} \\
&\leq \texttt{eps}|p| + (1+\texttt{eps})|e_n - \sum_{i=2}^{n} \pi_i a_{i+1}\cdots a_n| \\
&\leq \texttt{eps}|p| + (1+\texttt{eps})\gamma_{n-1}\gamma_{2n}|a_1 a_2 \cdots a_n|.
\end{aligned}$$

Since $(1+\texttt{eps})\gamma_{n-1} \leq \gamma_n$, it follows that $|\texttt{res} - p| \leq \texttt{eps}|p| + \gamma_n\gamma_{2n}|p|$. □

It may be interesting to study the condition number of the product evaluation. Ones defines

$$\mathrm{cond}(a) = \lim_{\varepsilon\to 0} \sup \left\{ \frac{|(a_1+\Delta a_1)(a_2+\Delta a_2)\cdots(a_n+\Delta a_n) - a_1 a_2 \cdots a_n|}{\varepsilon |a_1 a_2 \cdots a_n|} : |\Delta a_i| \leq \varepsilon |a_i| \right\}.$$

A standard computation yields

$$\mathrm{cond}(a) = n.$$

COROLLARY 3.4. *Suppose Algorithm 3.2 is applied to floating point number $a_i \in \mathbb{F}$, $1 \leq i \leq n$, and set $p = \prod_{i=1}^{n} a_i$. Then,*

$$\frac{|\texttt{res} - p|}{|p|} \leq \texttt{eps} + \frac{\gamma_n\gamma_{2n}}{n}\,\mathrm{cond}(a).$$

3.3. Faithful rounding

We define the floating point predecessor and successor of a real number r satisfying $\min\{f : f \in \mathbb{R}\} < r < \max\{f : f \in \mathbb{F}\}$ by

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

DEFINITION 3.1. *A floating point number $f \in \mathbb{F}$ is called a* faithful rounding *of a real number $r \in \mathbb{R}$ if*

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

We denote this by $f \in \square(r)$. For $r \in \mathbb{F}$, this implies that $f = r$.

A faithful rounding is then one of the two adjacent floating point numbers of the exact result.

LEMMA 3.5 (Rump, Ogita and Oishi (Rump et al., 2005, lem. 2.5)). *Let $r, \delta \in \mathbb{R}$ and $\widetilde{r} := \mathrm{fl}(r)$. Suppose that $2|\delta| < \texttt{eps}|\widetilde{r}|$. Then $\widetilde{r} \in \square(r+\delta)$, that means $\widetilde{r}$ is a faithful rounding of $r + \delta$.*

Let res be the result of CompProd. Then we have $p = p_n + e$ and $\texttt{res} = \mathrm{fl}(p_n + e_n)$ with $e = \sum_{i=2}^{n} \pi_i a_{i+1} \cdots a_n$. It follows that $p = (p_n + e_n) + (e - e_n)$. This leads to the following lemma which gives a criterion to ensure that the result of CompProd is faithfully rounded.

LEMMA 3.6. *With the previous notations, if* $2|e - e_n| < \texttt{eps}|\texttt{res}|$ *then* res *is a faithful rounding of* p.

Since we have $|e - e_n| \leq \gamma_n\gamma_{2n}|p|$ and $(1 - \texttt{eps})|p| - \gamma_n\gamma_{2n}|p| \leq |\texttt{res}|$, a sufficient condition to ensure a faithful rounding is

$$2\gamma_n\gamma_{2n}|p| < \texttt{eps}((1 - \texttt{eps})|p| - \gamma_n\gamma_{2n}|p|)$$

that is

$$\gamma_n\gamma_{2n} < \frac{1 - \texttt{eps}}{2 + \texttt{eps}}\texttt{eps}.$$

Since $\gamma_n\gamma_{2n} \leq 2(n\texttt{eps})^2/(1 - 2n\texttt{eps})^2$, a sufficient condition is

$$2\frac{(n\texttt{eps})^2}{(1 - 2n\texttt{eps})^2} < \frac{1 - \texttt{eps}}{2 + \texttt{eps}}\texttt{eps}$$

which is equivalent to

$$\frac{n\texttt{eps}}{1 - 2n\texttt{eps}} < \sqrt{\frac{(1 - \texttt{eps})\texttt{eps}}{2(2 + \texttt{eps})}}$$

and then to

$$n < \frac{\sqrt{1 - \texttt{eps}}}{\sqrt{2}\sqrt{2 + \texttt{eps}} + 2\sqrt{(1 - \texttt{eps})\texttt{eps}}}\texttt{eps}^{-1/2}.$$

We have just shown that if $n < \alpha\texttt{eps}^{-1/2}$ where $\alpha \approx 1/2$ then the result is faithfully rounded. More precisely, in double precision where $\texttt{eps} = 2^{-53}$, if $n < 2^{25} \approx 5 \cdot 10^7$, we get a faithfully rounded result.

3.4. Validated error bound

We present here how to compute a valid error bound in pure floating point arithmetic in rounding to nearest. It holds that

$$\begin{aligned} |\texttt{res} - p| &= |\,\mathrm{fl}(p_n + e_n) - p| = |\,\mathrm{fl}(p_n + e_n) - (p_n + e_n) + (p_n + e_n) - p| \\ &\leq \texttt{eps}|\texttt{res}| + |p_n + e_n - p| \\ &\leq \texttt{eps}|\texttt{res}| + |e_n - e|. \end{aligned}$$

Since $|e_n - e| \leq \gamma_{n-1}\gamma_{2n}|p|$ and $|p| \leq (1 + \texttt{eps})^{n-1}\,\mathrm{fl}(|a_1a_2 \cdots a_n|)$ we obtain

$$\begin{aligned} |\texttt{res} - p| &\leq \texttt{eps}|\texttt{res}| + \gamma_{n-1}\gamma_{2n}|p| \\ &\leq \texttt{eps}|\texttt{res}| + \gamma_{n-1}\gamma_{2n}(1 + \texttt{eps})^{n-1}\,\mathrm{fl}(|a_1a_2 \cdots a_n|). \end{aligned}$$

Using (8) and (9), we get

$$\begin{aligned}
|\mathtt{res} - p| &\leq \mathrm{fl}(\mathtt{eps}|\mathtt{res}|) + (1+\mathtt{eps})^n\,\mathrm{fl}(\gamma_n)\,\mathrm{fl}(\gamma_{2n})\,\mathrm{fl}(|a_1a_2\cdots a_n|)\\
&\leq \mathrm{fl}(\mathtt{eps}|\mathtt{res}|) + (1+\mathtt{eps})^{n+2}\,\mathrm{fl}(\gamma_n\gamma_{2n}|a_1a_2\cdots a_n|)\\
&\leq \mathrm{fl}(\mathtt{eps}|\mathtt{res}|) + \mathrm{fl}\left(\frac{\gamma_n\gamma_{2n}|a_1a_2\cdots a_n|}{1-(n+3)\mathtt{eps}}\right)\\
&\leq (1+\mathtt{eps})\,\mathrm{fl}\left(\mathtt{eps}|\mathtt{res}| + \frac{\gamma_n\gamma_{2n}|a_1a_2\cdots a_n|}{1-(n+3)\mathtt{eps}}\right)\\
&\leq \mathrm{fl}\left(\left(\mathtt{eps}|\mathtt{res}| + \frac{\gamma_n\gamma_{2n}|a_1a_2\cdots a_n|}{1-(n+3)\mathtt{eps}}\right) / \,(1-2\mathtt{eps})\right).
\end{aligned}$$

We can summarize this as follows.

LEMMA 3.7. *Suppose Algorithm 3.2 is applied to floating point numbers $a_i \in \mathbb{F}$, $1 \leq i \leq n$ and set $p = \prod_{i=1}^{n} a_i$. Then, the absolute forward error affecting the product is bounded according to*

$$|\mathtt{res} - p| \leq \mathrm{fl}\left(\left(\mathtt{eps}|\mathtt{res}| + \frac{\gamma_n\gamma_{2n}|a_1a_2\cdots a_n|}{1-(n+3)\mathtt{eps}}\right) / \,(1-2\mathtt{eps})\right).$$

3.5. Validated error bound and faithful rounding

In the previous subsection, we have shown that

$$|e_n - e| \leq \mathrm{fl}\left(\frac{\gamma_n\gamma_{2n}|a_1a_2\cdots a_n|}{1-(n+3)\mathtt{eps}}\right). \tag{11}$$

Lemma 3.6 tells us that if $2|e - e_n| < \mathtt{eps}|\mathtt{res}|$ then `res` is a faithful rounding of p (where `res` is the result of `CompProd`).

As a consequence, if

$$\mathrm{fl}\left(2\frac{\gamma_n\gamma_{2n}|a_1a_2\cdots a_n|}{1-(n+3)\mathtt{eps}}\right) < \mathrm{fl}(\mathtt{eps}|\mathtt{res}|)$$

then we got a faitfully rounded result. This makes it possible to check *a posteriori* if the result is faithfully rounded.

4. Conclusion

In this paper, we provided an accurate algorithm for computing product of floating point numbers. We gave some sufficient conditions to obtain a faithfully rounded result as well as validated error bounds.

References

Dekker, T. J.: 1971, 'A floating-point technique for extending the available precision'. *Numer. Math.* **18**, 224–242.

Graillat, S., N. Louvet, and P. Langlois: 2005, 'Compensated Horner Scheme'. Research Report 04, Équipe de recherche DALI, Laboratoire LP2A, Université de Perpignan Via Domitia, France, 52 avenue Paul Alduy, 66860 Perpignan cedex, France.

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

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

Knuth, D. E.: 1998, *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*. Reading, MA, USA: Addison-Wesley, third edition.

Kornerup, P., V. Lefevre, and J.-M. Muller: 2007, 'Computing Integer Powers in Floating-Point Arithmetic'. arXiv:0705.4369v1 [cs.NA].

Langlois, P. and N. Louvet: 2007, 'How to Ensure a Faithful Polynomial Evaluation with the Compensated Horner Algorithm'. In: *Proceedings of the 18th IEEE Symposium on Computer Arithmetic (ARITH '07), Montpellier, France*. pp. 141–149, IEEE Computer Society, Los Alamitos, CA, USA.

Ogita, T., S. M. Rump, and S. Oishi: 2005a, 'Accurate Sum And Dot Product'. *SIAM J. Sci. Comput.* **26**(6), 1955–1988.

Ogita, T., S. M. Rump, and S. Oishi: 2005b, 'Verified solution of linear systems without directed rounding'. Technical Report No. 2005-04, Advanced Research Institute for Science and Engineering, Waseda University.

Rump, S. M., T. Ogita, and S. Oishi: 2005, 'Accurate Floating-Point Summation'. Technical Report 05.12, Faculty for Information and Communication Sciences, Hamburg University of Technology.

Sterbenz, P. H.: 1974, *Floating-point computation*. Englewood Cliffs, N.J.: Prentice-Hall Inc. Prentice-Hall Series in Automatic Computation.