On the robustness of the 2Sum and Fast2Sum algorithms

Sylvie Boldo, Stef Graillat, Jean-Michel Muller

On the robustness of the 2Sum and Fast2Sum algorithms

Sylvie Boldo, Inria, Université Paris-Saclay, France
Stef Graillat, Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, Paris
Jean-Michel Muller, CNRS, LIP, Université de Lyon, France

The 2Sum and Fast2Sum algorithms are important building blocks in numerical computing. They are used (implicitely or explicitely) in many *compensated* algorithms (such as compensated summation or compensated polynomial evaluation). They are also used for manipulating floating-point *expansions*. We show that these algorithms are much more robust than it is usually believed: the returned result makes sense even when the rounding function is not round-to-nearest, and they are almost immune to overflow.

CCS Concepts: •**Mathematics of computing** → **Numerical analysis;** •**Software and its engineering** → **Correctness;** *Software verification and validation;*

Additional Key Words and Phrases: floating-point, error-free transformation, rounding errors, faithful rounding, 2Sum, Fast2Sum

ACM Reference Format:
Sylvie Boldo, Stef Graillat, and Jean-Michel Muller, 2016. On the robustness of the 2Sum and Fast2Sum algorithms. *ACM Trans. Math. Softw.* V, N, Article A (January YYYY), 14 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. MOTIVATIONS

One easily shows that, provided that no overflow occurred, the error of a rounded-to-nearest floating-point addition or subtraction is exactly representable by a floating-point number. The 2Sum [Knuth 1998] and Fast2Sum [Dekker 1971] algorithms make it possible to compute that error, under some conditions that will be reminded below. That error can therefore be used later on in a calculation: this is the underlying idea behind *compensated* algorithms. This has allowed for the development of various techniques, such as very accurate (compensated) summation [Kahan 1965; Neumaier 1974; Rump et al. 2008a; Rump et al. 2008b; Demmel and Nguyen 2013] or dot products, accurate polynomial evaluation [Graillat et al. 2009], efficient manipulation of floating-point expansions [Priest 1991; Shewchuk 1997; Hida et al. 2001] (floating-point expansions represent real numbers as unevaluated sums of floating-point numbers. Algorithms for adding and multiplying these expansions make much use of 2Sum and Fast2Sum), etc. However, these techniques suffer from some limitations:

— as noticed, among others, by Boldo and Daumas [Boldo and Daumas 2003], when the rounding function differs from round-to-nearest, the error of floating-point addi-

This work was supported by the FastRelax (ANR-14-CE25-0018-01) project of the French National Agency for Research (ANR).
Sylvie Boldo, Inria, LRI, CNRS & Univ. Paris-Sud, Université Paris-Saclay, bâtiment 650, Université Paris-Sud, F-91405 Orsay Cedex, France.
Stef Graillat, Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, 4 place Jussieu, F-75252 Paris Cedex 05, France
Jean-Michel Muller, CNRS, Laboratoire LIP, École Normale Supérieure de Lyon, 46 allée d'Italie, 69364 Lyon Cédex 07, France.

DOI: http://dx.doi.org/10.1145/0000000.0000000

tion/subtraction may not be exactly representable—although this is often the case, see Lemma 2.4 below. And yet, rounding functions such as round towards $\pm\infty$ and round towards zero (called "directed roundings") are very useful. They allow one to get certain lower and/or upper bounds on the exact result of a computation, and to easily implement *interval arithmetic* or *stochastic arithmetic*. Also, on many processors, changing the rounding mode is rather costly (it may require flushing the arithmetic pipe-line): hence someone who required directed roundings in previous parts of one's program may be reluctant to switch to round-to-nearest before using Algorithms 2Sum and Fast2Sum. Hence, with directed roundings, even if we cannot always obtain the "exact" error of floating-point addition, it would still be useful to obtain a value close to that error. This problem was partly dealt with by Demmel and Nguyen [Demmel and Nguyen 2013], and later on by Graillat, Jézéquel, and Picot [Graillat et al. 2015] for the Fast2Sum algorithm, and by Martin-Dorel et al. [Martin-Dorel et al. 2013] in the case of "double roundings". We aim at tackling this issue in a more general context, and we wish to study the behaviour of 2Sum and Fast2Sum just assuming "general" rounding functions (see definition 2.1 below). There already exist some kind of error-free transformations for summation with faithful rounding (see Priest [Priest 1992]). But these algorithms are costly and our work shows that they are not necessary to achieve a good accuracy in many compensated algorithms: the simple and well-known Fast2Sum and 2Sum algorithms will frequently suffice.

— in the literature, these algorithms are usually considered as returning a correct result provided that no underflow or overflow occurs. The case of underflow can be handled fairly intuitively, using a property mentioned by Hauser [Hauser 1996]—see below. The case of overflow is more problematic: the central question is: *can we have a "spurious" overflow?*, i.e., can we have situations where the initial addition does not overflow, and yet one of the arithmetic operations performed in the 2Sum or Fast2Sum algorithm overflows? We will see in the following that such a situation almost never arises: this gives more confidence on many compensated algorithms, and on algorithms that use floating-point expansions for "middle-precision" (e.g., around 100 digits) computations.

The rest of the paper is organized as follows. In Section 2, we introduce some notation, definitions and preliminary remarks used in the sequel. The accuracy of Fast2Sum with no overflow is analyzed in Section 3 while the accuracy of 2Sum is dealt with in Section 4. Section 5 is devoted to show that Fast2Sum is immune to overflow and Section 6 to show that 2Sum is almost immune to overflow.

2. NOTATION, DEFINITIONS, PRELIMINARY REMARKS

Throughout this paper, we assume a radix-2, precision-p, floating-point (FP) arithmetic, of extremal exponents $e_{\min}$ and $e_{\max}$. We also assume that subnormal numbers are available. We denote by Ω the largest representable floating-point number:

$$\Omega = (2 - 2^{1-p}) \cdot 2^{e_{\max}}.$$

The floating-point predecessor of a FP number x will be noted $\mathrm{pred}(x)$. Its successor will be noted $\mathrm{succ}(x)$. If x is a real number, satisfying $2^k \leq |x| < 2^{k+1}$, where k is an integer, we define $\mathrm{ulp}(x)$ as follows:

$$\mathrm{ulp}(x) = 2^{\max(k, e_{\min})-p+1}.$$

When an arithmetic operation τ is performed, with input FP operands a and b, what is actually computed is $\circ(a \tau b)$, where $\circ$ is a *rounding function*. The IEEE-754 Standard for Floating-point arithmetic defines 5 rounding functions (round towards $+\infty$—denoted RU below—, round towards $-\infty$—denoted RD below—, round towards zero, round to nearest ties to even, and round to nearest ties to infinity). The two round-to-nearest functions will be de-

noted RN in the following (the choice of the tie-breaking rule is not important here). We say that the FP number y is a *faithful rounding* of the real number x if $y \in \{\mathrm{RD}(x), \mathrm{RU}(x)\}$.

The rounding functions considered in this paper satisfy the following definition (introduced by Kulisch [Kulisch 1971] under the name of *optimal rounding*).

Definition 2.1 (*Rounding function—"optimal rounding" in [Kulisch 1971]*). Let F_p be the set of the precision-p binary floating-point numbers. Function $\circ$ from $\mathbb{R}$ to F_p is a *rounding function* if

— $\forall x \in F_p, \circ(x) = x$;
— $\forall (x, y) \in \mathbb{R}^2, x \leq y \Rightarrow \circ(x) \leq \circ(y)$.

Remark 2.2. If $\circ$ is a rounding function, then for any x, $\circ(x) \in \{\mathrm{RD}(x), \mathrm{RU}(x)\}$, where RD and RU are the rounds-towards $-\infty$ and round-towards $+\infty$ rounding functions.

The Fast2Sum algorithm was first introduced by Dekker [Dekker 1971]. It allows one to compute the error of a (rounded to nearest) floating-point addition. That algorithm is

ALGORITHM 1: Conventional Fast2Sum Algorithm.

$s \leftarrow \mathrm{RN}(a + b)$
$z \leftarrow \mathrm{RN}(s - a)$
$t \leftarrow \mathrm{RN}(b - z)$

The conventional 2Sum algorithm, due to Knuth [Knuth 1998] and Møller [Møller 1965], is

ALGORITHM 2: Conventional 2Sum algorithm.

(1) $s \leftarrow \mathrm{RN}(a + b)$
(2) $a' \leftarrow \mathrm{RN}(s - b)$
(3) $b' \leftarrow \mathrm{RN}(s - a')$
(4) $\delta_a \leftarrow \mathrm{RN}(a - a')$
(5) $\delta_b \leftarrow \mathrm{RN}(b - b')$
(6) $t \leftarrow \mathrm{RN}(\delta_a + \delta_b)$

We know that, in the absence of overflow, if the radix β of the floating-point system being used is less than or equal to 3, and if the floating-point exponents e_a and e_b of a and b satisfy $e_a \geq e_b$, then the values s and t returned by Algorithm 1 satisfy $s + t = a + b$, i.e., t is the error of the floating-point addition $s \leftarrow \mathrm{RN}(a + b)$. Depending on the environment, testing the exponents of a and b may prove difficult. However if $|a| \geq |b|$ then $e_a \geq e_b$. Algorithm 2 gives the same results as Algorithm 1, but without any requirement on β or on the exponents of a and b: it works in all cases provided that no overflow occurs. Due to the large penalty of a wrong branch prediction on modern architectures, if we do not have preliminary information on the respective orders of magnitude of a and b, calling the 6-operation 2Sum algorithm (Algorithm 2) is, in general, more efficient than comparing $|a|$ and $|b|$, swapping them if needed, and calling the 3-operation algorithm Fast2Sum (Algorithm 1).

Algorithms 1 and 2 allow one to compute the error of a floating-point addition, provided that this addition was performed using a round-to-nearest rounding function. The computed error can be re-injected later on in a calculation to compensate for it. This makes these "error-free transformations" very useful. However, when a rounding function different from round-to-nearest is used, the error of a floating-point addition is not always equal to a floating-point number. For instance [Muller et al. 2010], in a radix-2 and precision-p

arithmetic, assuming rounding toward $-\infty$, if $a = 1$ and $b = -2^{-3p}$, then

$$\begin{aligned} s &= \mathrm{RD}(a+b) = 0.\underbrace{111111\cdots11}_{p} \\ &= 1 - 2^{-p}, \end{aligned}$$

and

$$a + b - s = \underbrace{1.1111111111\cdots11}_{2p} \times 2^{-p-1},$$

which cannot be exactly represented with precision p (it would require precision $2p$).

Therefore, with rounding functions different from RN, it is important to know what Algorithms 1 and 2 (or, rather, a modified version, with different rounding functions, of these algorithms) will return, to know if they are still of any use.

This issue was already dealt with by Martin-Dorel, Melquiond, and Muller [Martin-Dorel et al. 2013] in the restricted case where the rounding function is round to nearest with a possible "double rounding".[1] Demmel and Nguyen show that if $4\mathrm{ulp}(a) \leq |b| \leq a$ then Algorithm 1 returns the error of the floating-point addition of a and b when directed rounding functions are used.

Graillat, Jézéquel, and Picot [Graillat et al. 2015] give an error bound on the value returned by Algorithm 1 when directed rounding functions are used. We will improve on their bound, showing that the algorithm always returns the best possible result, namely a floating-point number t closer to the error of the floating-point addition of a and b than any other floating-point number. We will perform a similar analysis with the 2Sum algorithm.

There is another issue with these two algorithms. One can rather easily convince oneself that they are immune to underflow. The main reason for this is that, as shown by Hauser [Hauser 1996], if the sum $a + b$ of two floating-point numbers is below the underflow threshold, then that sum is a floating-point number, which implies that it is computed exactly, with any rounding function (it can be viewed as a consequence of Lemma 2.4 below). It is, however, much more difficult to know if these algorithms are, at least for some restricted input domain, immune to *overflow*. More precisely, if the first operation (namely the floating-point addition of a and b) does not overflow, can one of the following operations overflow?

The goal of this paper is to deal with these two issues, and to show that Fast2Sum and 2Sum (Algorithms 1 and 2) are much more robust than it is in general believed: for any combination of rounding functions (we can even have a different rounding function at each step of the algorithm) they are immune to overflow (except for a very limited number of "extreme" cases that are easy to detect), and they always produce a very accurate estimate of the error of the floating-point addition $a+b$. The algorithms that we will analyze are the following:

ALGORITHM 3: Fast2Sum with faithful roundings: $\circ_1, \circ_2, \circ_3$ are rounding functions (see Definition 2.1).

$s \leftarrow \circ_1(a+b)$
$z \leftarrow \circ_2(s-a)$
$t \leftarrow \circ_3(b-z)$

[1] This happens when the result of an operation is first rounded to a wider floating-point format, before being rounded to the destination format.

ALGORITHM 4: 2Sum with faithful roundings: $\circ_i$, for $i = 1, \ldots, 6$, are rounding functions (see Definition 2.1).

(1) $s \leftarrow \circ_1(a + b)$
(2) $a' \leftarrow \circ_2(s - b)$
(3) $b' \leftarrow \circ_3(s - a')$
(4) $\delta_a \leftarrow \circ_4(a - a')$
(5) $\delta_b \leftarrow \circ_5(b - b')$
(6) $t \leftarrow \circ_6(\delta_a + \delta_b)$

We will make much use of the following result, due to Sterbenz [Sterbenz 1974] (see for instance [Hauser 1996] or [Muller et al. 2010] for a proof).

LEMMA 2.3 (STERBENZ). *In a radix-β floating-point system with subnormal numbers available, if x and y are finite floating-point numbers such that*

$$\frac{y}{2} \leq x \leq 2y,$$

then $x - y$ is a floating-point number.

Lemma 2.4 below is common computer arithmetic folklore. We give a proof of it for the sake of completeness.

LEMMA 2.4. *Let a and b be two binary FP numbers of respective exponents e_a and e_b. Let $s \in \{\mathrm{RD}(a + b), \mathrm{RU}(a + b)\}$. If the exponent e_s of s is less than or equal to $\min(e_a, e_b)$ then $s = a + b$ exactly.*

PROOF. First a and b are multiple of $2^{e_a - p + 1}$ and $2^{e_b - p + 1}$, respectively. Since $e_s \leq \min(e_a, e_b)$, the number $a + b$ is an integer multiple of $2^{e_s - p + 1}$. Hence:

— the largest multiple of $2^{e_s - p + 1}$ less than or equal to $a + b$ is $a + b$ itself, and
— the smallest multiple of $2^{e_s - p + 1}$ larger than or equal to $a + b$ is $a + b$ itself.

Therefore $\mathrm{RD}(a + b) = a + b$, and $\mathrm{RU}(a + b) = a + b$. Hence $s = a + b$. □

The following lemma allows one to understand the behavior of the first two lines of Fast2Sum.

LEMMA 2.5. *Let a and b be two binary FP numbers, with $e_a \geq e_b$. Let $s \in \{\mathrm{RD}(a+b), \mathrm{RU}(a+b)\}$. The number $s - a$ is a floating-point number (which implies that it will be computed exactly, with any rounding function).*

Notice that Lemma 2.5 only holds in radix 2. With floating-point systems of higher radices, we can build counter-examples. For instance, in radix 3 with $p = 4$ and $\circ = \mathrm{RU}$, if $a = 1002_3 = 29_{10}$ and $b = 2222_3 = 80_{10}$, then $s = \mathrm{RU}(a + b) = 11010_3 = 111_{10}$, so that $s - a = 10001_3 = 82_{10}$ is not exactly representable with precision 4.

PROOF. We have $a = M_a \cdot 2^{e_a - p + 1}$ and $b = M_b \cdot 2^{e_b - p + 1}$, with $|M_a|, |M_b| \leq 2^p - 1$. Without loss of generality, we assume $M_a \geq 0$. Let M_s and e_s be the integral significand and the exponent of s, respectively. Since $|s| \leq 2 \max\{|a|, |b|\}$, we have $e_s \leq e_a + 1$.

(1) if $e_s = e_a + 1$, then

$$M_s \in \left\{ \left\lfloor \frac{M_a}{2} + \frac{M_b}{2^{1+(e_a - e_b)}} \right\rfloor, \left\lceil \frac{M_a}{2} + \frac{M_b}{2^{1+(e_a - e_b)}} \right\rceil \right\}. \tag{1}$$

Defining $\mu = 2M_s - M_a$, from (1), we obtain

$$\frac{M_b}{2^{e_a - e_b}} - 2 < \mu < \frac{M_b}{2^{e_a - e_b}} + 2,$$

which implies $|\mu| \leq |M_b|+1 \leq 2^p$. An immediate consequence is that $s-a = \mu \cdot 2^{e_a-p+1}$ is a floating-point number.

(2) if $e_s \leq e_a$ then first notice that if $e_s \leq e_b$ then $s = a + b$ exactly by Lemma 2.4 so that $s - a = b$ is a floating-point number. Therefore we need only to focus on the case $e_s > e_b$. In that case

$$s \in \left\{ \left\lfloor 2^{e_a-e_s} M_a + 2^{e_b-e_s} M_b \right\rfloor \cdot 2^{e_s-p+1}, \left\lceil 2^{e_a-e_s} M_a + 2^{e_b-e_s} M_b \right\rceil \cdot 2^{e_s-p+1} \right\};$$

so that

$$\left(2^{e_b-e_s} M_b - 1\right) \cdot 2^{e_s-p+1} < s - a < \left(2^{e_b-e_s} M_b + 1\right) \cdot 2^{e_s-p+1}.$$

Hence $|s - a|$ is of the form $K \cdot 2^{e_s-p+1}$, with

$$|K| \leq \frac{|M_b|}{2} + 1 < 2^p,$$

which implies that it is a floating-point number.

□

The following lemma shows that even when the rounding function is not round-to-nearest, the error of a floating-point addition will very frequently be exactly representable by a floating-point number.

LEMMA 2.6. *Let a and b be binary, precision-p, floating-point numbers. Let $s \in \{\mathrm{RD}(a + b), \mathrm{RU}(a + b)\}$. If the difference $|e_a - e_b|$ of the exponents of a and b does not exceed $p - 1$, then $s - (a + b)$ is a binary, precision-p, floating-point number.*

PROOF. Without loss of generality, we assume $|a| \geq |b|$, and $e_a - e_b \leq p-1$. The numbers a and b are multiple of 2^{e_b-p+1}, therefore $a+b$ and s are multiple of 2^{e_b-p+1} too. Therefore, there exists an integer Z such that

$$(a + b) - s = Z \cdot 2^{e_b-p+1}. \tag{2}$$

Let e_s be the FP exponent of s. Since $|s - (a + b)| < \mathrm{ulp}(s)$, we have $|(a + b) - s| < 2^{e_s-p+1}$. Since $|b| \leq |a|$, we have $|s| \leq 2|a|$, which implies $e_s \leq e_a + 1$. Therefore

$$|(a + b) - s| < 2^{e_a-p+2} \leq 2^{e_b+1}. \tag{3}$$

By combining (2) and (3) we deduce that $|Z| \leq 2^p - 1$, therefore $(a+b)-s$ is a FP number. □

3. ACCURACY OF FAST2SUM IN THE ABSENCE OF OVERFLOW

Let us first deal with Algorithm Fast2Sum with arbitrary rounding functions (Algorithm 3).

THEOREM 3.1. *If no overflow occurs, and $e_a \geq e_b$ then the values s and t returned by Algorithm 3 satisfy*

$$t = \circ_3((a + b) - s),$$

i.e., t is a faithful rounding of the error of the FP addition $s \leftarrow \circ_1(a + b)$.

Notice that if we combine this theorem with Lemma 2.6, we deduce that if the difference of the exponents of a and b does not exceed $p - 1$ (which will occur in many practical cases), then t is exactly $(a + b) - s$.

PROOF. Lemma 2.5 above implies that $s-a$ is a floating-point number. Hence, $z = s-a$, so that

$$t = \circ_3(b - z) = \circ_3((a + b) - s).$$

□

4. ACCURACY OF 2SUM IN THE ABSENCE OF OVERFLOW

We now consider Algorithm 2Sum with arbitrary rounding functions (Algorithm 4). Contrarily to what happens in the previous section with Algorithm 3, we do not always obtain a final value t equal to a faithful rounding of $(a+b)-s$. Consider the following example, in binary32/single-precision arithmetic ($p = 24$):

— $a = 3076485 \cdot 2^{-21}$, $b = -6130317 \cdot 2^{-49}$;
— $\circ_1 = \circ_2 = \circ_5 = \mathrm{RU}$, $\circ_3 = \circ_4 = \circ_6 = \mathrm{RD}$.

We successively obtain:

$$\begin{aligned} s &= a = 3076485 \cdot 2^{-21}; \\ a' &= 12305941 \cdot 2^{-23}; \\ b' &= -2^{-23}; \\ \delta_a &= -2^{-23}; \\ \delta_b &= 15244637 \cdot 2^{-47}; \\ t &= -1532579 \cdot 2^{-47}. \end{aligned}$$

and since $(a+b)-s = b$ is a floating-point number, with any rounding function $\circ$, $\circ((a+b)-s) = b$ is different from t. However, $(a+b-s)-t = -2^{-49}$, so that t remains a very good approximation to $(a+b)-s$. As we are going to see, this is always true. More precisely, we prove together the following two results. The first one (Theorem 4.1) is the main result of this section. The second one (Lemma 4.2) is needed in the proof of Theorem 6.2.

THEOREM 4.1. *If $p \geq 4$ and no overflow occurs, then the values s and t returned by Algorithm 4 satisfy*

$$t = (a+b) - s + \alpha,$$

with $|\alpha| < 2^{-p+1} \cdot \mathrm{ulp}(a+b) \leq 2^{-p+1} \cdot \mathrm{ulp}(s)$. Furthermore, if the floating-point exponents e_s and e_b of s and b satisfy $e_s - e_b \leq p-1$ then t is a faithful rounding of $(a+b)-s$.

LEMMA 4.2. *If $p \geq 4$ and no overflow occurs in lines (1) to (5) of Algorithm 4, then the variables δ_a and δ_b computed at lines (4) and (5) satisfy*

$$|\delta_a + \delta_b| \leq \mathrm{ulp}(a+b).$$

PROOF. We prove together Theorem 4.1 and Lemma 4.2. This means the case split and intermediate results are the same, but they do not rely one on another. Without loss of generality, we assume $a \geq 0$. Figure 4 below illustrates the various cases that are discussed in the proof. Case 1 ($|b| \geq a$) just uses Theorem 3.1 (we can exhibit Algorithm 3, hidden in the lines of Algorithm 4). Case 2 ($|b| < a$ and $|s| \leq |b|$) is a simple application of Sterbenz' Lemma (Lemma 2.3), and Case 3 ($|b| < a$ and $|s| > |b|$) requires more effort. Notice that in Cases 1 and 2, t will always be a faithful rounding of $a+b-s$. Hence, the part in the theorem that is specific to the case $e_s - e_b \leq p-1$ does not need to be addressed in these first two cases.

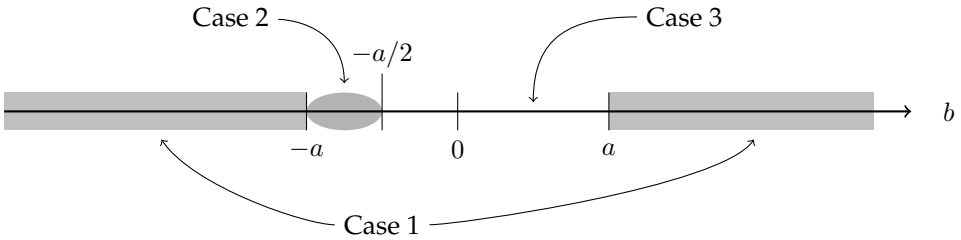

Fig. 1. Various cases discussed in the proof.

Case 1 – If $|\mathbf{b}| \geq \mathbf{a}$ then lines (1), (2), and (4) of Algorithm 4 are equivalent to Fast2Sum(b,a). Therefore, from Theorem 3.1, we have $a' = s - b$ and $\delta_a = \circ_4(a + b - s)$, hence $|\delta_a| \leq \text{ulp}(a + b)$. An immediate consequence of $a' = s - b$ is $b' = b$ and $\delta_b = 0$. From this, we find

$$t = \circ_4(a + b - s),$$

which proves the result of Theorem 4.1; and $|\delta_a + \delta_b| \leq \text{ulp}(a + b)$, so that the result of Lemma 4.2 holds.

Case 2 – If $|\mathbf{b}| < \mathbf{a}$ and $|\mathbf{s}| \leq |\mathbf{b}|$ (which is equivalent to saying that $-a < b \leq -a/2$) then by Sterbenz Lemma, $s = a + b$. An immediate consequence is $a' = a$, $b' = b$, $\delta_a = \delta_b = 0$ (so that, obviously, the result of Lemma 4.2 holds) , $t = 0$. Hence $t = (a + b) - s$ (so that the result of Theorem 4.1 holds).

Case 3 – If $|\mathbf{b}| < \mathbf{a}$ and $|\mathbf{s}| > |\mathbf{b}|$ (which is equivalent to saying that $-a/2 < b < a$), notice that we have $s > 0$. Let $u = 2^{1-p}$ (i.e., u is the rounding unit for directed roundings). We have

$$\begin{array}{rcl} s & = & (a + b) \cdot (1 + \epsilon_1); \text{ with } |\epsilon_1| \leq u; \\ a' & = & (s - b) \cdot (1 + \epsilon_2); \text{ with } |\epsilon_2| \leq u. \end{array}$$

Thus $a' = (a + a\epsilon_1 + b\epsilon_1) \cdot (1 + \epsilon_2)$. Since $|b| < a$, $a\epsilon_1 + b\epsilon_1$ can be written $2a\epsilon_3$, with $|\epsilon_3| \leq u$. Therefore

$$a' = a \cdot (1 + \eta),$$

with $|\eta| \leq 3u + 2u^2$. As soon as $p \geq 4$, we have $|\eta| < 1/2$, so that $a' \geq 0$ and $a/2 \leq a' \leq 2a$. Therefore, Sterbenz Lemma applies to line (4) of Algorithm 4, and

$$\delta_a = a - a'. \tag{4}$$

Also, since $s > |b|$, Lines (2) and (3) of Algorithm 4 are equivalent to the first two lines of Fast2Sum$(s, -b)$, so that

$$b' = s - a', \tag{5}$$

and

$$\delta_b = \circ_5(a' - (s - b)). \tag{6}$$

Notice that, from Lemma 2.6, as soon as the exponents e_s and e_b of s and b satisfy $e_s - e_b \leq p - 1$, (6) implies $\delta_b = a' - (s - b)$, which, combined with (4), gives $t = \circ_6(a + b - s)$, so that Theorem 4.1 holds. Also, in that case, $\delta_a + \delta_b = (a + b) - s$, so that Lemma 4.2 holds. Hence, *let us now assume that* $e_s - e_b \geq p$. Notice that this implies

$$|b| < 2^{e_b+1} \leq 2^{e_s-p+1} = \text{ulp}(s).$$

Hence,

$$a' \in \{\text{succ}(s), s, \text{pred}(s), \text{pred}(\text{pred}(s))\}.$$

Notice that the case $a' = \text{pred}(\text{pred}(s))$ can occur only when s is a power of 2.

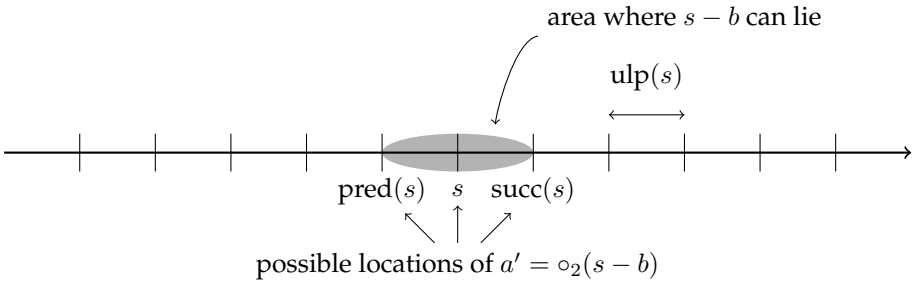

Fig. 2. General case: s is not a power of 2

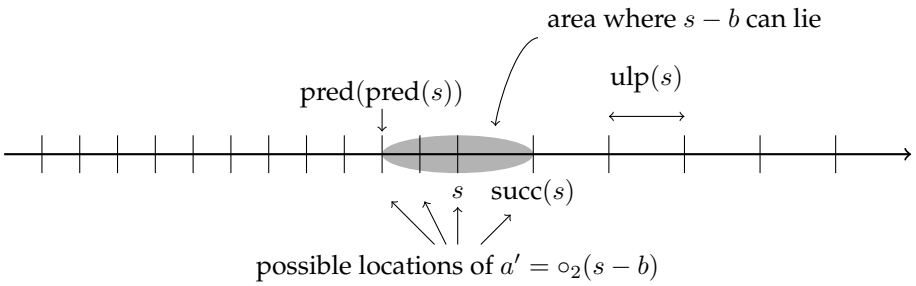

Fig. 3. Special case: s is a power of 2

(1) **If $a' = s$ then** $b' = 0$. It follows that $\delta_b = b$ and $\delta_a = a - s$, for which we deduce $t = \circ_6(\delta_a + \delta_b) = \circ_6(a + b - s)$, so that Theorem 4.1 holds, and $|\delta_a + \delta_b| = |a + b - s| <$ ulp$(a + b)$, so that Lemma 4.2 holds.

(2) **If $a' \neq s$ then**

$$a' = s - \sigma \cdot \text{ulp}(s), \text{with } \sigma \in \{-1, 1/2, 1\},$$

(the case $\sigma = 1/2$ can occur only when s is a power of 2), and we have

$$\begin{aligned} b' &= \sigma \cdot \text{ulp}(s) \\ \delta_a &= a - s + \sigma \cdot \text{ulp}(s) \\ \delta_b &= \circ_5(b - \sigma \cdot \text{ulp}(s)). \end{aligned}$$

We know that $|b| <$ ulp(s). Furthermore, b has the same sign as σ. Therefore

— either $|b| \geq |\sigma|/2 \cdot$ ulp(s), in which case Sterbenz Lemma implies $\delta_b = b - \sigma \cdot$ ulp(s), so that $\delta_a + \delta_b = a + b - s$. This has two consequences: firstly, $|\delta_a + \delta_b| = |a + b - s| <$ ulp$(a + b)$, so that Lemma 4.2 holds; and secondly, $t = \circ_6(a + b - s)$, therefore Theorem 4.1 holds;

— or $|b| < |\sigma|/2 \cdot$ ulp(s), in which case

$$|b - \sigma \cdot \text{ulp}(s)| < |\sigma| \cdot \text{ulp}(s)$$

(unless $b = 0$ but that case is straightforwardly handled), so that (since $|\sigma| \cdot$ ulp(s) is a power of 2)

$$|\delta_b - (b - \sigma \cdot \text{ulp}(s))| < \frac{|\sigma|}{2}\text{ulp}(\text{ulp}(s)) = |\sigma| \cdot 2^{-p}\text{ulp}(s)$$

(since ulp(s) is a power of 2). An immediate consequence is

$$|(\delta_a + \delta_b) - (a + b - s)| < |\sigma| \cdot 2^{-p}\text{ulp}(s). \tag{7}$$

Since we already know that $|(a+b)-s| < \text{ulp}(a+b)$, we deduce

$$|\delta_a + \delta_b| < \text{ulp}(a+b) + |\sigma| \cdot 2^{-p}\text{ulp}(s). \tag{8}$$

Let us try to slightly improve on the bound (8). First, from $|b| \leq \text{ulp}(s)$ one easily deduces $a > s/2$ (otherwise, we would have $a+b \leq s/2 + \text{ulp}(s)$, which would imply $s = \circ_1(a+b) \leq \circ_1(s/2 + \text{ulp}(s)) = s/2 + \text{ulp}(s)$). Hence δ_a is a multiple of $\frac{1}{2}\text{ulp}(s)$. Also, $\text{ulp}(a+b)$ is equal to $\text{ulp}(s)$ or $\frac{1}{2}\text{ulp}(s)$.
Finally, $|b| < |\sigma|/2 \cdot \text{ulp}(s)$ and $b' = \sigma \cdot \text{ulp}(s)$ imply $|b - b'| > \frac{|\sigma|}{2}\text{ulp}(s)$, so that $|\delta_b| \geq \frac{|\sigma|}{2}\text{ulp}(s)$, which implies that δ_b is a multiple of $|\sigma| \cdot 2^{-p}\text{ulp}(s)$. All this implies that $\delta_a + \delta_b$ is a multiple of $|\sigma| \cdot 2^{-p}\text{ulp}(s)$. Hence, from (8), we deduce

$$|\delta_a + \delta_b| \leq \text{ulp}(a+b).$$

First, this proves Lemma 4.2. Furthermore, since $\text{ulp}(a+b)$ is a power of 2, we obtain

$$|\circ_6(\delta_a + \delta_b) - (\delta_a + \delta_b)| \leq \frac{1}{2}\text{ulp}\left(\text{ulp}(a+b)\right) = 2^{-p}\text{ulp}(a+b).$$

Combined with (7), this gives

$$|t - (a+b-s)| < 2^{-p} \cdot \left(\text{ulp}(a+b) + |\sigma| \cdot \text{ulp}(s)\right). \tag{9}$$

This already gives $|t-(a+b-s)| < 2^{-p+1} \cdot \text{ulp}(s)$. Let us now try to express a bound on $|t-(a+b-s)|$ as a function of $\text{ulp}(a+b)$ only. We have four cases to consider

(a) if s is not a power of 2, or if $a+b \geq s$, then $\text{ulp}(a+b) = \text{ulp}(s)$, which gives $|t-(a+b-s)| < 2^{-p+1} \cdot \text{ulp}(a+b)$, so that Theorem 4.1 holds;
(b) if s is a power of 2 and $a+b < s$ and $\sigma = 1/2$, then $\text{ulp}(a+b) = 1/2 \cdot \text{ulp}(s)$, and (9) implies $|t-(a+b-s)| < 2^{-p+1} \cdot \text{ulp}(a+b)$, so that Theorem 4.1 holds;
(c) the case when s is a power of 2, $a+b < s$, and $\sigma = 1$ is impossible: we assumed $|b| < |\sigma|/2 \cdot \text{ulp}(s) = 1/2 \cdot \text{ulp}(s)$, which implies $s - b \geq s - 1/2 \cdot \text{ulp}(s) = \text{pred}(s)$, which implies $a' = \circ_2(s-b) \geq \text{pred}(s)$, which is not compatible with the assumption $\sigma = 1$, since $a' = s - \sigma\text{ulp}(s)$;
(d) if s is a power of 2, $a+b < s$, and $\sigma = -1$, we have the following relations (see Fig. 4): $a' = \text{succ}(s) = s + \text{ulp}(s)$, $b' = -\text{ulp}(s)$, and $-1/2 \cdot \text{ulp}(s) < b < 0$. We deduce that $a > \text{pred}(s) = s - \frac{1}{2}\text{ulp}(s)$ (otherwise we would have $a+b < \text{pred}(s)$, which would imply $s = \circ_1(a+b) \leq \text{pred}(s)$). Similarly, we have $a < \text{succ}(s) = s + \text{ulp}(s)$ (otherwise, we would have $a+b \geq \text{succ}(s) - \frac{1}{2}\text{ulp}(s) > s$). Therefore $a = s$, from which we immediately deduce $\delta_a = -\text{ulp}(s)$ and $\delta_b = \circ_5(b + \text{ulp}(s))$. Now, δ_a and δ_b have opposite signs, and

$$\frac{1}{2}\text{ulp}(s) < b + \text{ulp}(s) < \text{ulp}(s),$$

(notice that since $\text{ulp}(s)$ is a power of 2, this implies $\text{ulp}(b+\text{ulp}(s)) \leq 2^{-p}\text{ulp}(s)$) from which we deduce

$$\frac{|\delta_a|}{2} = \frac{1}{2}\text{ulp}(s) \leq \circ_5(b + \text{ulp}(s)) = \delta_b \leq |\delta_a| = \text{ulp}(s),$$

hence we can apply Sterbenz lemma to the addition of δ_a and δ_b, which gives

$$\begin{aligned} t = \circ_6(\delta_a + \delta_b) &= \delta_a + \delta_b \\ &= -\text{ulp}(s) + \circ_5(b + \text{ulp}(s)) \\ &= b + \eta \\ &= a + b - s + \eta, \end{aligned}$$

(since $a = s$), with $|\eta| < \text{ulp}(b + \text{ulp}(s)) \leq 2^{-p} \cdot \text{ulp}(s) = 2^{-p+1}\text{ulp}(a+b)$, hence Theorem 4.1 holds.

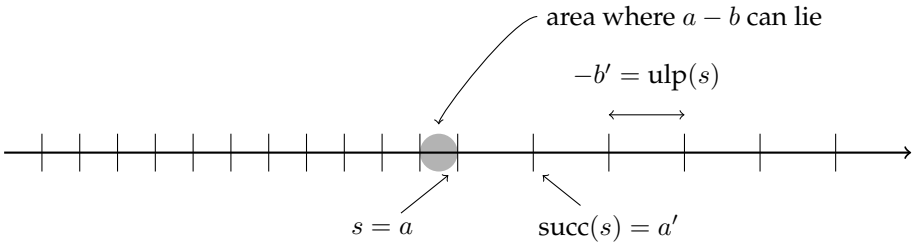

Fig. 4. Last case: s is a power of 2, $a + b < s$, and $\sigma = -1$

□

5. FAST2SUM IS IMMUNE TO OVERFLOW

Let us now consider Algorithm 3, with $e_a \geq e_b$, where e_a and e_b are the floating-point exponents of a and b, and let us assume that no overflow occurred in the first line ($s \leftarrow \circ_1(a+b)$). Without loss of generality, we can assume $a > 0$. Let us first deal with the second line of the algorithm ($z \leftarrow \circ_2(s-a)$).

We have $s = a + b + \epsilon$, with $|\epsilon| < \mathrm{ulp}(a+b) \leq 2\mathrm{ulp}(a)$. Hence $s - a = b + \epsilon$. Therefore, if the computation of $z = \circ_2(s-a)$ overflows, it means that either $b < -\Omega + 2\mathrm{ulp}(a)$ or $b > \Omega - 2\mathrm{ulp}(a)$.

The second case is impossible: if $b > \Omega - 2\mathrm{ulp}(a) \geq \Omega - 2\mathrm{ulp}(\Omega)$, then (since $e_a \geq e_b$, which here implies $e_a = e_b = e_{\max}$) then $a+b \geq \Omega - 2\mathrm{ulp}(\Omega) + 2^{e_{\max}} = 3\cdot 2^{e_{\max}} - 3\cdot 2^{e_{\max}-p+1}$, which implies that $a + b$ overflows. Let us consider the first case. In that case, we have $b < -\Omega + 2\mathrm{ulp}(a) \leq -\Omega + 2\mathrm{ulp}(\Omega)$ and (since $e_a \geq e_b$ which implies here $e_a = e_{\max}$), $\Omega/2 < 2^{e_{\max}} \leq a \leq \Omega$), in the first operation we are in the conditions of Sterbenz Lemma, so that $s = a + b$, which implies $z = b$: in that case the computation of z does not overflow.

Hence, in all cases, the second line of Algorithm 3 cannot overflow. Let us now deal with the last line ($t \leftarrow \circ_3(b-z)$). We know from Lemma 2.5 that $z = s - a$, so that $b - z = a + b - s$. The computation of t can overflow only if $|b - z| > \Omega$, but this is impossible since

$$|b - z| = |(a+b) - s| < \mathrm{ulp}(s) < |s|.$$

We immediately deduce

THEOREM 5.1. *Assume that we perform Algorithm 3 with input values a and b whose exponents satisfy $e_a \geq e_b$. If the computation of s (first line of the algorithm) does not overflow, then the other lines of the algorithm cannot overflow.*

6. 2SUM IS ALMOST IMMUNE TO OVERFLOW

The overflow analysis of Algorithm 4 will be significantly more difficult. Our main result is Theorem 6.2 below. To make its proof simpler, we first prove the following result.

LEMMA 6.1. *If there are no overflows at lines (1) to (5) of Algorithm 4, there cannot be an overflow at line (6).*

PROOF. From Lemma 4.2, we know that $|\delta_a + \delta_b| \leq \mathrm{ulp}(a+b)$. Since no overflow occurs at line (1), $a + b$ is in the representable range, so that $\mathrm{ulp}(a+b) \leq 2^{-p+1}|a+b|$ is much below the overflow threshold. Hence line (6) of Algorithm 4 (namely, $t \leftarrow \circ_6(\delta_a + \delta_b)$) cannot overflow. □

THEOREM 6.2. *If the first input value a of Algorithm 4 satisfies $|a| < \Omega$ and if there is no overflow at line (1) of the algorithm, then there will be no overflow at lines (2) to (6).*

PROOF. Without loss of generality, we assume $a > 0$ and $b \neq 0$. Assume that no overflow occurred in the first line ($s \leftarrow \circ_1(a+b)$).

(1) **If $\mathbf{b > 0}$**
The monotonicity of the rounding functions implies: i) $s \geq b$, so that $a' \geq \circ_2(0) = 0$; and ii) $a' \leq \circ_2(s) = s$. Therefore

$$0 \leq a' \leq s, \tag{10}$$

which implies that there is no overflow at line (2) of the algorithm. Now, (10) implies $0 \leq s - a' \leq s$, so that

$$0 \leq b' \leq s. \tag{11}$$

As a consequence, there is no overflow at line (3) of the algorithm.
Now, since $a > 0$ and $a' \geq 0$, we deduce $|a - a'| \leq \max\{a, a'\}$, hence line (4) cannot overflow.
Similarly, since $b > 0$ and $b' \geq 0$, we obtain $|b - b'| \leq \max\{b, b'\}$, hence line (5) cannot overflow.
Lemma 6.1 implies that line (6) cannot overflow.

(2) **If $\mathbf{b < 0}$**
Notice that there cannot be an overflow at line (1): $|a+b|$ (hence $|s|$) is less than or equal to $\max\{|a|, |b|\}$.
(a) **if $-\mathbf{b} < \mathbf{a}$**, then $a + b - \mathrm{ulp}(a+b) < s < a + b + \mathrm{ulp}(a+b)$, so that

$$a + b - \mathrm{ulp}(a) < s < a + b + \mathrm{ulp}(a) \tag{12}$$

(since $|a+b| < a$, which implies $\mathrm{ulp}(a+b) \leq \mathrm{ulp}(a)$). We therefore deduce

$$a - \mathrm{ulp}(a) < s - b < a + \mathrm{ulp}(a).$$

therefore, unless $a = \Omega$, there will be no overflow at line (2) of the algorithm, and $a' = \circ_2(s-b)$ will satisfy

$$a - \mathrm{ulp}(a) \leq a' \leq a + \mathrm{ulp}(a). \tag{13}$$

(this is deduced using the monotonicity of the rounding function $\circ_2$ and the fact that $a - \mathrm{ulp}(a)$ and $a + \mathrm{ulp}(a)$ are floating-point numbers). We now assume $a \neq \Omega$ (which, with our assumption $-b < a$, implies $-b < \mathrm{pred}(\Omega)$), i.e., since b is a floating-point number,

$$|b| = -b \leq \mathrm{pred}(\mathrm{pred}(\Omega)). \tag{14}$$

From (12) and (13), we find

$$b - 2\mathrm{ulp}(a) < s - a' < b + 2\mathrm{ulp}(a), \tag{15}$$

This, along with (14) and $\mathrm{ulp}(a) \leq \mathrm{ulp}(\Omega)$ implies that line (3) of the algorithm cannot overflow. Notice that $0 < -b < a$ implies

$$|b \pm 2\mathrm{ulp}(a)| < a + 2\mathrm{ulp}(a). \tag{16}$$

It also implies that a cannot be the smallest nonzero subnormal floating-point number $2^{e_{\min}-p+1}$. Hence $a \geq 2^{e_{\min}-p+2}$, so that $a \geq 2\mathrm{ulp}(a)$. This and (16) give $|b \pm 2\mathrm{ulp}(a)| \leq 2a$ so that $\mathrm{ulp}(b \pm 2\mathrm{ulp}(a)) \leq 2\mathrm{ulp}(a)$. Combined with (15), this gives

$$b - 4\mathrm{ulp}(a) \leq b' \leq b + 4\mathrm{ulp}(a). \tag{17}$$

Now, from (13) and (17), we deduce $|a - a'| \leq \mathrm{ulp}(a)$ and $|b - b'| \leq 4\mathrm{ulp}(a)$, to that lines (4) and (5) of the algorithm cannot overflow. Lemma 6.1 implies that line (6) cannot overflow.

(b) **if $-\mathbf{b} \geq \mathbf{a}$.** First, notice that the case $a \geq -b/2$ is easily handled, since Sterbenz Lemma applied to line (1) of Algorithm 4implies $s = a + b$, so that $a = a'$, $b = b'$, and $\delta_a = \delta_b = t = 0$. Hence we only need to focus on the case $\mathbf{a < -b/2}$.
From $0 \leq a < -b/2$ we deduce $b \leq a + b < b/2$, which implies[2]

$$b \leq s \leq b/2. \tag{18}$$

The consequence of (18) is twofold. First, we immediately deduce $0 \leq s - b \leq -b/2$, so that Line (2) of Algorithm 4 cannot overflow, and second, Sterbenz Lemma can be applied to line (2) of Algorithm 4, so that $a' = s - b$. It follows that $b' = b$ and Line (3) cannot overflow. Therefore $a - a' = a + b - s$, so that $|a - a'| < \mathrm{ulp}(a + b)$, hence Line (4) cannot overflow. We finally have $\delta_b = b - b' = 0$ and $t = \delta_a$: lines (5) and (6) cannot overflow.

□

Notice that condition $|a| < \Omega$ is necessary. Assume all rounding functions are RN (with ties-to-even tie-breaking rule). The choice $a = \Omega$ and $b = -(3/2) \cdot \mathrm{ulp}(\Omega)$ will give no overflow at line (1), and an overflow at line (2).

Conclusion

We have shown that, in binary floating-point arithmetic, the 2Sum and Fast2Sum algorithms are more "robust" than it is usually believed: even when the error of the initial floating-point addition is not exactly representable, they return a very good approximation to that error. Also, they are almost totally immune to overflow: the only case where a "spurious" overflow may occur is with 2Sum, when the absolute value of operand a is equal to the largest floating-point number.

Acknowledgement

We are very grateful to the anonymous referees for the detailed and very helpful review of our paper.

REFERENCES

S. Boldo and M. Daumas. 2003. Representable correcting terms for possibly underflowing floating point operations. In *Proceedings of the 16th Symposium on Computer Arithmetic*, J.-C. Bajard and M. Schulte (Eds.). IEEE Computer Society Press, Los Alamitos, CA, 79–86. http://perso.ens-lyon.fr/marc.daumas/SoftArith/BolDau03a.pdf

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

J. Demmel and H. D. Nguyen. 2013. Fast Reproducible Floating-Point Summation. In *21st IEEE Symposium on Computer Arithmetic, Austin, TX, USA, April 7-10*. 163–172.

S. Graillat, F. Jézéquel, and R. Picot. 2015. Numerical Validation of Compensated Summation Algorithms with Stochastic Arithmetic. *Electronic Notes in Theoretical Computer Science* 317 (2015), 55 – 69. DOI:http://dx.doi.org/10.1016/j.entcs.2015.10.007 The Seventh and Eighth International Workshops on Numerical Software Verification (NSV).

S. Graillat, P. Langlois, and N. Louvet. 2009. Algorithms for accurate, validated and fast computations with polynomials. *Japan Journal of Industrial and Applied Mathematics* 26, 2 (2009), 215–231.

J. R. Hauser. 1996. Handling floating-point exceptions in numeric programs. *ACM Trans. Program. Lang. Syst.* 18, 2 (1996), 139–174. DOI:http://dx.doi.org/10.1145/227699.227701

Y. Hida, X. S. Li, and D. H. Bailey. 2001. Algorithms for Quad-Double Precision Floating-Point Arithmetic. In *Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH-16)*, N. Burgess and L. Ciminiera (Eds.). Vail, CO, 155–162. DOI:http://dx.doi.org/10.1109/ARITH.2001.930115

[2]Unless $b/2$ is not a floating-point number: this can happen only of b is subnormal, and in that case, with $0 \leq a < -b/2$, overflow is of course impossible.

W. Kahan. 1965. Pracniques: further remarks on reducing truncation errors. *Commun. ACM* 8, 1 (1965), 40. DOI:http://dx.doi.org/10.1145/363707.363723

D. Knuth. 1998. *The Art of Computer Programming* (3rd ed.). Vol. 2. Addison-Wesley, Reading, MA.

U. W. Kulisch. 1971. An axiomatic approach to rounded computations. *Numer. Math.* 19 (1971), 1–17.

É. Martin-Dorel, G. Melquiond, and J.-M.s Muller. 2013. Some issues related to double rounding. *BIT Numerical Mathematics* 53, 4 (2013), 897–924. DOI:http://dx.doi.org/10.1007/s10543-013-0436-2

O. Møller. 1965. Quasi Double-Precision in Floating-Point Addition. *BIT* 5 (1965), 37–50.

J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres. 2010. *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston. 572 pages.

A. Neumaier. 1974. Rundungsfehleranalyse einiger Verfahren zur Summation endlicher Summen. *ZAMM* 54 (1974), 39–51. In German.

D. M. Priest. 1991. Algorithms for arbitrary precision floating point arithmetic. In *Proceedings of the 10th IEEE Symposium on Computer Arithmetic (Arith-10)*, P. Kornerup and D. W. Matula (Eds.). IEEE Computer Society Press, Los Alamitos, CA, 132–144.

D. M. Priest. 1992. *On Properties of Floating-Point Arithmetics: Numerical Stability and the Cost of Accurate Computations*. Ph.D. Dissertation. University of California at Berkeley.

S. M. Rump, T. Ogita, and S. Oishi. 2008a. Accurate Floating-Point Summation Part I: Faithful Rounding. *SIAM J. Sci. Comput.* 31, 1 (2008), 189–224. DOI:http://dx.doi.org/10.1137/050645671

S. M. Rump, T. Ogita, and S. Oishi. 2008b. Accurate Floating-Point Summation Part II: Sign, K-Fold Faithful and Rounding to Nearest. *SIAM J. Sci. Comput.* 31, 2 (2008), 1269–1302. DOI:http://dx.doi.org/10.1137/07068816X

J. R. Shewchuk. 1997. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discrete Computational Geometry* 18 (1997), 305–363. http://link.springer.de/link/service/journals/00454/papers97/18n3p305.pdf

P. H. Sterbenz. 1974. *Floating-Point Computation*. Prentice-Hall, Englewood Cliffs, NJ.