On Ziv's rounding test

FLORENT DE DINECHIN, École Normale Supérieure de Lyon
CHRISTOPH LAUTER, Université Pierre et Marie Curie Paris VI
JEAN-MICHEL MULLER, CNRS
SERGE TORRES, École Normale Supérieure de Lyon

A very simple test, introduced by Ziv, allows one to determine if an approximation to the value $f(x)$ of an elementary function at a given point x suffices to return the floating-point number nearest $f(x)$. The same test may be used when implementing floating-point operations with input and output operands of different formats, using arithmetic operators tailored for manipulating operands of the same format. That test depends on a "magic constant" e. We show how to choose that constant e to make the test reliable and efficient. Various cases are considered, depending on the availability of an fma instruction, and on the range of $f(x)$.

Categories and Subject Descriptors: G.1.0 [**Numerical analysis**]: Computer arithmetic

Additional Key Words and Phrases: Floating-Point arithmetic, correct rounding, elementary functions

ACM Reference Format:
de Dinechin, F., Lauter, C., Muller, J.-M., and Torres, S. 2012. On Ziv's rounding test. *ACM Trans. Math. Softw.* 0, 0, Article 0 (February 2013), 19 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

The result of an accurate numerical calculation is frequently available as the unevaluated sum of two floating-point numbers: a "main term" y_h and a "correcting term" y_ℓ, such that if y is the exact result for which we are trying to find an approximation, $y_h + y_\ell$ is very close to y (say, within some relative error bound ϵ). Examples occur in: evaluation of elementary functions, compensated summation algorithms, and (as we will see below) implementation of "heterogeneous" floating-point operations. The problem we face here is to check whether we can easily deduce the floating-point number nearest y from y_h, y_ℓ and ϵ. Before going further, let us recall some elementary notions of floating-point arithmetic and define some notation.

1.1. Floating-point numbers

A binary, precision-p, floating-point number is a number of the form

$$x = X \cdot 2^{k_x - p + 1}, \tag{1}$$

This work is partly supported by the french Agence Nationale de la Recherche, under grant ANR 2010 BLAN 0203 01 (TaMaDi project).
Author's addresses: F. de Dinechin and J.-M. Muller and S. Torres, Laboratoire LIP, ENS Lyon, 46 allée d'Italie, 69364 Lyon Cedex 07, France; C. Lauter, Université Pierre et Marie Curie Laboratoire d'Informatique de Paris 6, Équipe PEQUAN, Boîte Courrier 169, 4, place Jussieu 75252 Paris Cedex 05, France.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2013 ACM 0098-3500/2013/02-ART0 $15.00
DOI: http://dx.doi.org/10.1145/0000000.0000000

where X is an integer, $|X| \leq 2^p - 1$, and k_x is an integer. In practical implementations of floating-point arithmetic, it is required that k_x should be between two extremal exponents, $k_{\min}$ and $k_{\max}$. The floating-point representation (1) of x is said *normal* if $|X|$ is the largest possible that still satisfies $|X| \leq 2^p - 1$. When (1) is the normal representation of X, X is called the *integral significand* of x, $X \cdot 2^{-p+1}$ is called the *significand* of x, and k_x is called the *exponent* of x. A *normal* number has absolute value larger than or equal to $2^{k_{\min}}$. Its integral significand is larger than or equal to 2^{p-1}. A *subnormal* number has absolute value less than $2^{k_{\min}}$.

Assuming a given precision p, we call *midpoint* a number that is exactly halfway between two consecutive floating-point numbers, and we define a function *ulp* (which is an acronym for *unit in the last place* [Kahan 2004]) as follows:

— if $|x| \in [2^k, 2^{k+1})$, where k is an integer larger than or equal to $k_{\min}$, then $\mathrm{ulp}(x) = 2^{k-p+1}$;
— if x is subnormal then $\mathrm{ulp}(x) = 2^{k_{\min}-p+1}$.

Roughly speaking, $\mathrm{ulp}(x)$ is the distance between two consecutive floating-point numbers in the neighborhood of x.

We define $\mathrm{RN}_p(t)$ as the precision-p floating-point number that is closest to t (when t falls exactly halfway between two consecutive floating-point numbers, $\mathrm{RN}_p(t)$ will be the one of these two numbers whose integral significand is even: this is the so-called *round to nearest ties to even* rule of the IEEE 754 standard [IEEE Computer Society 2008]), and $\mathrm{RU}_p(t)$ as the smallest floating-point number that is larger than or equal to t. When there is no ambiguity on the value of p, we just write "$\mathrm{RN}(t)$" or "$\mathrm{RU}(t)$". With a correctly rounded arithmetic, assuming that round to nearest ties to even is the chosen rounding rule, the number `x` obtained after executing the C-instruction `x = y + z` is RN(`y + z`).

We will also need the following algorithm, due to Dekker:

ALGORITHM 1: Fast2Sum(a, b)

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

It satisfies:

THEOREM 1.1 (DEKKER'S FAST2SUM ALGORITHM). *(for a proof, see [Dekker 1971; Knuth 1998; Muller et al. 2010])*

Assume a binary floating-point system.

Let a and b be floating-point numbers, and assume that the exponent of a is larger than or equal to that of b. Algorithm 1 computes two floating-point numbers s and t that satisfy:

— $s + t = a + b$ *exactly;*
— *s is the floating-point number that is closest to $a + b$.*

In the following, we assume we use a binary floating-point arithmetic that is compliant with the IEEE 754 standard, which implies that the arithmetic operations are correctly rounded, and we also assume that the "round-to-nearest ties-to-even" rounding direction is selected (or that none is selected, since that one is the default).

1.2. Instances of the "main+correcting terms" problem

1.2.1. Evaluation of functions. Assume one wishes to write a program that evaluates a real-valued function f with *correct rounding*, that is, for any input floating-point number x, the program must return $\mathrm{RN}(f(x))$.[1] A strategy for doing this, first suggested and implemented by Ziv [Ziv 1991] in his library of correctly-rounded transcendentals is the following one:

— First, compute an approximation, say y_1, to $f(x) = y$ with relative error less than ϵ_1. From that approximation and the error bound, one can deduce an interval I_1 such that $y \in I_1$.
— If all elements of I_1 round to the same floating-point number $\hat{y}$ (in other words, if I_1 does not contain a *midpoint*), then, necessarily, $\mathrm{RN}(f(x)) = \hat{y}$.
— Otherwise, we need to perform another step, i.e., to compute a more accurate approximation y_2, with relative error bound $\epsilon_2 < \epsilon_1$, and so on: we may need an arbitrarily long sequence of approximations y_1, y_2, y_3, etc., that become more and more costly to compute.

This is illustrated by Fig. 1. This process can be improved if we know in advance what is the smallest nonzero relative distance between $f(z)$, where z is a floating-point number, and a midpoint, provided that this smallest distance is not too small. When we know that smallest distance, we can implement an efficient program with a small (in practice, 2 or 3) number of steps. Knowing that smallest distance requires solving a problem called the *Table Maker's Dilemma* [Kahan 2004; Muller 2006].

The important fact is the following: since the cost of computing an approximation increases as the error bound ϵ_i decreases, it is very important to make sure that we hardly ever need to use the second, third, etc. steps. Hence, the test that decides whether we need to perform a second step must satisfy the following properties:

— of course, it must be *reliable*: if (y_1, ϵ_1) does not suffice to deduce the value of $\mathrm{RN}(y)$, the test must say so;
— the number of "wrong alarms" (i.e., of cases for which (y_1, ϵ_1) would have sufficed to deduce the value of $\mathrm{RN}(y)$, and yet step 2 is performed) must be as small as possible;
— the test itself must be very "cheap", i.e., cost a very few arithmetic operations and comparisons only.

In the following, we assume the availability of an algorithm that makes it possible to approximate $y = f(x)$ by the unevaluated sum of two floating-point numbers: a "main term" y_h and a "correcting term" y_ℓ, with relative error bounded by some ϵ, that is,

$$y_h + y_\ell = y \cdot (1 + \alpha), \qquad \text{with} \quad |\alpha| \leq \epsilon. \tag{2}$$

Such algorithms do exist for evaluating the usual transcendental functions (in general, through some judicious combination of table look-up and polynomial approximation). We can even assume that y_ℓ is small enough in front of y_h, so that the floating-point number nearest $y_h + y_\ell$ is y_h: otherwise, it suffices to use the Fast2Sum algorithm, (Algorithm 1, given above), to get into this situation.

We wish to find some way of very quickly checking whether $y_h = \mathrm{RN}(y)$ or not. We will call that problem the "main+correcting terms" problem (see below for a more accurate definition).

[1] We assume that $f(x)$ is not exactly halfway between two consecutive floating-point numbers. This case can be shown to never happen in nontrivial cases with the most usual transcendental functions such as $\sin$, $\exp$, $\ln$, ..., and concerning the most frequent algebraic functions, this problem is partly dealt with in [Jeannerod et al. 2011].

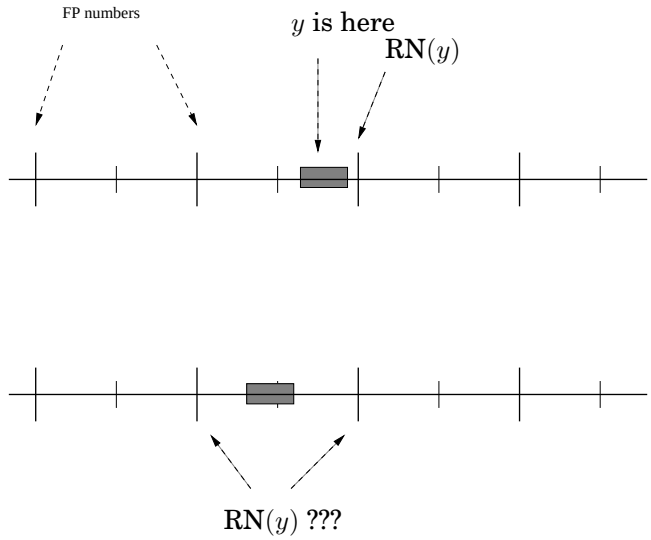

Fig. 1. Given an approximation y_1 to $y = f(x)$ and a bound ϵ_1 on the approximation error, we know that y lies in an interval I_1. In the first case, all elements of I_1 round to the same floating-point number, so that we can return RN(y). In the second case, a new, more accurate, approximation y_2 must be computed.

1.2.2. Implementation of heterogeneous operations. A similar situation also occurs when we want to implement "heterogeneous" floating-point operations (that is, operations whose operands have different floating-point formats), assuming we only have hardware operators for "homogeneous" operations. Notice that such heterogeneous operations are required by the IEEE 754-2008 Standard for Floating-Point Arithmetic [IEEE Computer Society 2008]. Suppose we have two different formats available on the same system: a format of precision p, and another of precision q, where $p < q$ (typical real-life examples are with $p = 53$ and $q = 64$, which correspond to the binary64/double precision and the "Intel double-extended" formats, and with $p = 24$ and $q = 53$, which correspond to the binary32/single precision and binary64/double precision formats). Also suppose we only have addition/subtraction operators for adding two numbers of precision p and returning a result in precision p; and for adding two numbers of precision q and returning a result in precision q (we consider here the case of addition, but the very same applies to multiplication, division, and square root). We want to add two numbers of precision q, and get the result in precision p, rounded to nearest. That is, given $a^{(q)}$ and $b^{(q)}$ of precision q, we wish to compute

$$y^{(p)} = \mathrm{RN}_p(a^{(q)} + b^{(q)}).$$

To do so, the first idea that springs in mind is to first compute, using the precision-q floating-point addition instruction:

$$y^{(q)} = \mathrm{RN}_q(a^{(q)} + b^{(q)}),$$

which satisfies[2] $y^{(q)} = (a^{(q)} + b^{(q)})(1 + \mu)$, with $|\mu| \leq 2^{-q}$, and then to round $y^{(q)}$ to the nearest precision-p floating-point number, that is to compute

$$\hat{y}^{(p)} = \mathrm{RN}_p(y^{(q)}).$$

[2]Unless underflow occurs, but for the usual floating-point formats, an underflow in precision q will correspond to a number of absolute value under the smallest nonzero floating-point number in precision p.

Unfortunately, although $\hat{y}^{(p)}$ will frequently be equal to $y^{(p)} = \mathrm{RN}_p(a^{(q)} + b^{(q)})$, this will not always be the case: this is a typical issue of the *double rounding* problem [Monniaux 2008; Boldo and Melquiond 2008]. An example with $p = 53$ and $q = 64$ is $a^{(q)} = 2^{63} + 2^{10}$ and $b^{(q)} = 2^{-2}$, for which one easily finds that $\hat{y}^{(p)} = 2^{63}$ whereas $y^{(p)} = 2^{63} + 2^{11}$.

Once $\hat{y}^{(p)}$ is computed, one may compute

$$y_\ell^* = y^{(q)} - \hat{y}^{(p)}.$$

Since $\hat{y}^{(p)}$ is a precision-p floating-point number and $p < q$, $\hat{y}^{(p)}$ is also a precision-q floating-point number. Therefore, y_ℓ^* can be computed using the precision-q floating-point addition instruction.

One easily shows that y_ℓ^* is exactly representable in precision q. Hence, y_ℓ^* is exactly computed. We now consider two cases:

— if $q \leq 2p + 1$ then y_ℓ^* is exactly representable in precision p. Hence, after having (exactly) computed y_ℓ^* using precision-q addition, one can convert it to the precision-p format without any loss of accuracy. By choosing $y_h = \hat{y}^{(p)}$, $y_\ell = y_\ell^*$, and $\alpha = \mu$,we find

$$y_h + y_\ell = (a^{(q)} + b^{(q)})(1 + \alpha),$$

with $|\alpha| \leq 2^{-q}$ and $\mathrm{RN}(y_h + y_\ell) = y_h$, which is just another instance of (2);

— if $q \geq 2p + 1$, then define $y_\ell = \mathrm{RN}_p(y_\ell^*)$. We have

$$|y_\ell - y_\ell^*| \leq 2^{-p}|y_\ell^*| \leq 2^{-2p}|y^{(q)}| \leq 2^{-2p}(1 + \mu)|a^{(q)} + b^{(q)}|,$$

Hence, by choosing $y_h = \hat{y}^{(p)}$, we find

$$y_h + y_\ell = \left(\hat{y}^{(p)} + y_\ell^*\right) + (y_\ell - y_\ell^*) = \left(a^{(q)} + b^{(q)}\right) \cdot (1 + \alpha),$$

with $|\alpha| \leq 2^{-q} + 2^{-2p} + 2^{-2p-q}$ and $\mathrm{RN}(y_h + y_\ell) = y_h$,[3] which is just another instance of (2).

1.2.3. Summation algorithms. There is a large literature on summation algorithms that first approximate a sum $x_1 + x_2 + \cdots + x_m$ by a main term and a correcting term and then return the floating-point sum of these two terms (see for instance [Pichat 1972; Neumaier 1974; Ogita et al. 2005]). Being able to quickly check whether we finally obtain the floating-point number closest to the exact sum is, of course, of interest.

1.3. Statement of the "main+correcting terms" problem

Let us now state the "main+correcting terms" problem more formally.

PROBLEM 1 (MAIN+CORRECTING TERMS PROBLEM). *Assume that an exact, real number y, is approximated by two precision-p binary floating-point numbers y_h and y_ℓ, that satisfy*

$$|(y_h + y_\ell) - y| < \epsilon \cdot |y|, \tag{3}$$

with $\epsilon < 2^{-p-1}$, and

$$y_h = \mathrm{RN}(y_h + y_\ell). \tag{4}$$

[3]Unless $y_\ell = (1/2)\mathrm{ulp}(y_h)$ exactly, but this case corresponds to the case when $a^{(q)} + b^{(q)}$ is extremely close to a precision-p midpoint: in such a case, the test will correctly reply that we are unable to conclude whether y_h equals $\mathrm{RN}(a^{(q)} + b^{(q)})$ or not.

We assume that y_h is a normal number (otherwise, the only floating-point value y_ℓ that satisfies (4) is $y_\ell = 0$). The problem we address here is the following one: find some simple test that makes it possible to quickly determine whether y_h is the floating-point number closest to y or not.

1.4. Ziv's rounding technique

In Ziv's Accurate Portable Mathlib (or `libultim`) library of elementary functions [Ziv 1991], the test being performed to solve Problem 1 is the following one:

$$\textit{Is } y_h \textit{ equal to } \mathrm{RN}\left(y_h + \mathrm{RN}(y_\ell \cdot e)\right)?$$

where e is some "magic constant" that must be adequately chosen (this is the main purpose of this paper). A similar test has also been used, later on, in the CRLibm library [de Dinechin et al. 2005]. Notice that $\mathrm{RN}\left(y_h + \mathrm{RN}(y_\ell \cdot e)\right)$ is what we get[4] by evaluating, in floating-point arithmetic, the `C` expression:

```
yh + (yl * e).
```

We wish to find that "magic constant" e. It will be a nonnegative floating-point number, as small as possible (due to Observation 1, below), such that

$$y_h = \mathrm{RN}\left(y_h + \mathrm{RN}(y_\ell \cdot e)\right) \Rightarrow y_h = \mathrm{RN}(y). \tag{5}$$

Notice that (3) and (4) imply that y_h and y have the same sign. Without loss of generality, we assume that they are positive. In the following, we will say that e *allows for a correct Ziv rounding test* if (5) holds as soon as (3) and (4) hold. We will say that (y, y_h, y_ℓ) is a *false negative* if (3) and (4) hold, $y_h \neq \mathrm{RN}\left(y_h + \mathrm{RN}(y_\ell \cdot e)\right)$ and yet $y_h = \mathrm{RN}(y)$. Our goal is to make sure that e should allow for a correct Ziv rounding test, and that false negatives should be as infrequent as possible.

Let us now raise two observations:

OBSERVATION 1. *If (y, y_h, y_ℓ) is a false negative for the Ziv rounding test with constant e, then it will be a false negative with any constant $e' > e$. In other words, the number of false negatives will increase with e.*

OBSERVATION 2. *If constant e does not allow for a correct Ziv rounding test, then no constant $e' < e$ will allow for a correct Ziv rounding test.*

These two observations are an almost immediate consequence of the fact that the function $t \mapsto \mathrm{RN}(t)$ is monotonic. From these observations, we conclude that we should try, given ϵ and p, to find a value of e as small as possible that allows for a correct Ziv rounding test. This is the purpose of the next section.

2. MAIN RESULT

2.1. Preliminary properties

PROPERTY 1. *If $(y - y_h)$ and y_ℓ have different signs then $y_h = \mathrm{RN}(y)$.*

PROOF. If $(y - y_h)$ and y_ℓ have different signs then either $y \leq y_h \leq y_h + y_\ell$ or $y_h + y_\ell \leq y_h \leq y$. In any case, y is closer to y_h than to $y_h + y_\ell$, so that (3) implies $|y_h - y| < \epsilon \cdot |y|$. Since $|\epsilon| < 2^{-p-1}$, this implies $y_h = \mathrm{RN}(y)$. This is illustrated by Figure 2. □

[4]Assuming *round-to-nearest, ties-to-even* rounding direction attribute, which is the default.

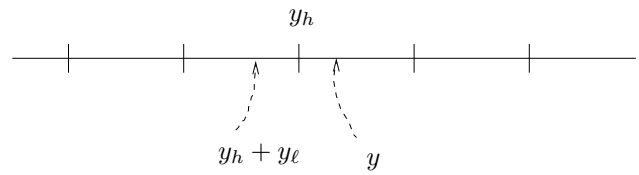

Fig. 2. Illustration of Property 1: when $(y - y_h)$ and y_ℓ have different signs, y is closer to y_h than to $y_h + y_\ell$, which implies $y_h = \mathrm{RN}(y)$.

Now, notice that (3) implies $|y_h - y| < \epsilon \cdot |y| + |y_\ell|$, and $y < (y_h + y_\ell)/(1-\epsilon)$, from which we deduce

$$|y_h - y| < \frac{\epsilon}{1-\epsilon} \cdot (y_h + y_\ell) + |y_\ell|. \tag{6}$$

2.1.1. If y_h is not a power of two

PROPERTY 2. *Assume that y_h is not a power of two. If $|y_\ell| \leq (1/2)\mathrm{ulp}(y_h) \cdot (1-\epsilon) - y_h\epsilon$ then $y_h = \mathrm{RN}(y)$.*

PROOF. If $|y_\ell| \leq (1/2)\mathrm{ulp}(y_h) \cdot (1-\epsilon) - y_h\epsilon$, then

$$\frac{1}{2}\mathrm{ulp}(y_h) - y_h \cdot \frac{\epsilon}{1-\epsilon} \geq |y_\ell| \left(1 + \frac{\epsilon}{1-\epsilon}\right),$$

which implies

$$\frac{\epsilon}{1-\epsilon} \cdot (y_h + y_\ell) + |y_\ell| \leq \frac{1}{2}\mathrm{ulp}(y_h).$$

Therefore, using (6), we find $|y_h - y| < (1/2)\mathrm{ulp}(y_h)$, which implies $y_h = \mathrm{RN}(y)$. □

PROPERTY 3. *Assume that y_h is not a power of two. Let e be a nonnegative floating-point number. If $(1/2)\mathrm{ulp}(y_h)$ is in the normal range, and $y_h = \mathrm{RN}\left(y_h + \mathrm{RN}(y_\ell \cdot e)\right)$, then $|y_\ell| \leq (1 + 2^{-p}) \cdot \mathrm{ulp}(y_h)/(2e)$.*

PROOF. From $y_h = \mathrm{RN}\left(y_h + \mathrm{RN}(y_\ell \cdot e)\right)$, and since y_h is not a power of 2, we deduce

$$y_h - \frac{1}{2}\mathrm{ulp}(y_h) \leq y_h + \mathrm{RN}(y_\ell \cdot e) \leq y_h + \frac{1}{2}\mathrm{ulp}(y_h),$$

which implies

$$|\mathrm{RN}(y_\ell \cdot e)| \leq \frac{1}{2}\mathrm{ulp}(y_h). \tag{7}$$

Since $(1/2)\mathrm{ulp}(y_h)$ is in the normal range, (7) implies $|y_\ell \cdot e| \leq (1 + 2^{-p}) \cdot (1/2)\mathrm{ulp}(y_h)$, q.e.d. □

Combining Properties 2 and 3, we find that if

$$\frac{(1 + 2^{-p}) \cdot \mathrm{ulp}(y_h)}{2e} \leq \frac{1}{2}\mathrm{ulp}(y_h) \cdot (1-\epsilon) - y_h\epsilon, \tag{8}$$

then $y_h = \mathrm{RN}\left(y_h + \mathrm{RN}(y_\ell \cdot e)\right)$ implies $y_h = \mathrm{RN}(y)$. Now, let us find a value of e, as small as possible, such that (8) holds for all floating-point numbers y_h that are not in the subnormal range. Condition (8) is equivalent to

$$e \geq \frac{(1 + 2^{-p}) \cdot \mathrm{ulp}(y_h)}{(1-\epsilon) \cdot \mathrm{ulp}(y_h) - 2y_h\epsilon}.$$

If y_h is in the normal range, then one can write $\text{ulp}(y_h) = \xi \cdot y_h \cdot 2^{-p}$, with $1 < \xi \leq 2$. We therefore have

$$\frac{(1+2^{-p}) \cdot \text{ulp}(y_h)}{(1-\epsilon) \cdot \text{ulp}(y_h) - 2y_h\epsilon} = \frac{(1+2^{-p}) \cdot \xi}{(1-\epsilon) \cdot \xi - 2^{p+1}\epsilon}.$$

Elementary calculus shows that if $\epsilon < 1/(2^{p+1}+1)$ then the largest value of

$$\frac{(1+2^{-p}) \cdot \xi}{(1-\epsilon) \cdot \xi - 2^{p+1}\epsilon}$$

for $1 \leq \xi \leq 2$ is attained for $\xi = 1$. From this, we immediately deduce

PROPERTY 4. *Assume that y_h is not a power of 2, that $\frac{1}{2}\text{ulp}(y_h)$ is in the normal range and that $\epsilon < 1/(2^{p+1}+1)$. If*

$$e \geq \frac{1+2^{-p}}{1-\epsilon-2^{p+1}\epsilon} \tag{9}$$

then $y_h = \text{RN}\left(y_h + \text{RN}(y_\ell \cdot e)\right)$ implies $y_h = \text{RN}(y)$.

2.1.2. If y_h is a power of 2. Reminder: we still assume that y and y_h are positive.

(1) first, if $y \geq y_h$ and $y_\ell \geq 0$ (i.e., $y_h + y_\ell \geq y_h$), then we have

$$y - y_h < \frac{1}{2}\text{ulp}(y_h) \Rightarrow y = \text{RN}(y_h),$$

and

$$(y_h = \text{RN}(y_h + \text{RN}(y_\ell \cdot e)) \Rightarrow y_h \leq y_h + \text{RN}(y_\ell \cdot e) \leq y_h + \frac{1}{2}\text{ulp}(y_h),$$

so that the proofs of properties 2, 3, and 4 remain valid: Property 4 still holds;

(2) second, if ($y \leq y_h$ and $y_\ell \geq 0$) or ($y \geq y_h$ and $y_\ell \leq 0$) then, from Property 1, $y = \text{RN}(y_h)$;

(3) the remaining case is $y \leq y_h$ and $y_\ell \leq 0$. In that case, since y_h is a power of 2, if y_h is in the normal range, then $\text{ulp}(y_h) = 2^{-p+1}y_h$, and we have the following properties.

PROPERTY 5. *Assume that y_h is a power of 2, $y \leq y_h$ and $y_\ell \leq 0$. If*

$$|y_\ell| \leq y_h \cdot \frac{(1-\epsilon) \cdot 2^{-p-1} - \epsilon}{1-2\epsilon} \tag{10}$$

then $y_h = \text{RN}(y)$.

Notice that Condition (10) cannot be satisfied if $2^{-p-1} - \epsilon/(1-\epsilon) \leq 0$, that is, if $\epsilon \geq 1/(1+2^{p+1})$.

PROOF. If (10) is satisfied, then

$$\frac{\epsilon}{1-\epsilon}(y_h + y_\ell) + |y_\ell| \leq 2^{-p-1}y_h,$$

which implies, using (6), that $0 \leq y_h - y \leq 2^{-p-1}y_h = (1/4)\text{ulp}(y_h)$, from which we deduce $y_h = \text{RN}(yh)$. □

Notice that, since y_h is a power of 2, the usual condition $|y_h - y| < \frac{1}{2}\text{ulp}(y_h)$ would not have sufficed to show that $y_h = \text{RN}(y)$. This is illustrated by Fig. 3

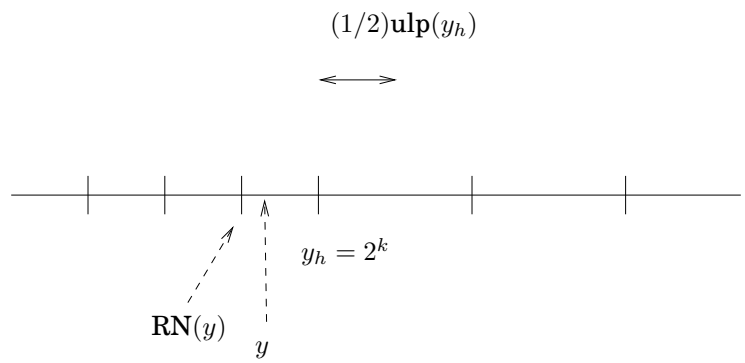

Fig. 3. When y_h is a power of 2, $|y_h - y| < \frac{1}{2}\mathrm{ulp}(y_h)$ does not necessarily imply that $y_h = \mathrm{RN}(y)$.

PROPERTY 6. *Assume that y_h is a power of 2, $y \leq y_h$, that $y_\ell \leq 0$, and that $2^{-p-1}y_h = (1/4)\mathrm{ulp}(y_h)$ is a normal number. If*

$$y_h = \mathrm{RN}(y_h + \mathrm{RN}(y_\ell \cdot e)), \tag{11}$$

then $|y_\ell| \leq ((1 + 2^{-p}) \cdot 2^{-p-1} \cdot y_h)/e$.

PROOF. Since y_h is a power of 2, (11) implies $y_h - (1/4)\mathrm{ulp}(y_h) \leq y_h + \mathrm{RN}(y_\ell \cdot e) \leq y_h$, which implies (since $\mathrm{ulp}(y_h) = 2^{-p+1}y_h$),

$$-2^{-p-1}y_h \leq \mathrm{RN}(y_\ell \cdot e) \leq 0,$$

which implies (since $2^{-p-1}y_h$ is a normal number), $|y_\ell \cdot e| \leq (1 + 2^{-p}) \cdot 2^{-p-1}y_h$. q.e.d □

Finally, from Properties 5, 6, and the previous observations, we deduce

PROPERTY 7. *Assume that y_h is a power of 2, that $\frac{1}{4}\mathrm{ulp}(y_h)$ is a normal number, and that $\epsilon < 1/(1 + 2^{p+1})$. If*

$$e \geq \frac{(1 + 2^{-p}) \cdot (1 - 2\epsilon)}{1 - \epsilon - 2^{p+1}\epsilon} \tag{12}$$

then $y_h = \mathrm{RN}\left(y_h + \mathrm{RN}(y_\ell \cdot e)\right)$ implies $y_h = \mathrm{RN}(y)$.

PROOF. As discussed above, we only have to consider the case where $y \leq y_h$ and $y_\ell \leq 0$. From properties 5 and 6 we immediately see that if

$$\frac{(1 + 2^{-p}) \cdot 2^{-p-1} \cdot y_h}{e} \leq y_h \cdot \frac{2^{-p-1} - \frac{\epsilon}{1-\epsilon}}{1 - \frac{\epsilon}{1-\epsilon}}$$

then $y_h = \mathrm{RN}\left(y_h + \mathrm{RN}(y_\ell \cdot e)\right)$ will imply $y_h = \mathrm{RN}(y)$. This immediately gives Condition (12). □

2.2. On the values of e that allow for a correct Ziv rounding test

From Properties 4 and 7, we easily deduce

THEOREM 2.1. *Assume that y_h is a floating-point number such that $\frac{1}{4}\mathrm{ulp}(y_h)$ is in the normal range, and that ϵ is less than $1/(2^{p+1}+1)$. Also assume that $y_h = \mathrm{RN}(y_h + y_\ell)$ and $|(y_h + y_\ell) - y| < \epsilon \cdot |y|$, with $\epsilon < 2^{-p-1}$. If*

$$e \geq \frac{1 + 2^{-p}}{1 - \epsilon - 2^{p+1}\epsilon}$$

then $y_h = \mathrm{RN}\left(y_h + \mathrm{RN}(y_\ell \cdot e)\right)$ *implies* $y_h = \mathrm{RN}(y)$.

Theorem 2.1 means that in practice, we will perform the rounding test using

$$e = \mathrm{RU}\left(\frac{1+2^{-p}}{1-\epsilon-2^{p+1}\epsilon}\right),$$

where $\mathrm{RU}(t)$ is the smallest floating-point number larger than or equal to t. Notice that if a fused multiply-add (fma) instruction is available, one may compute $\mathrm{RN}(y_h + y_\ell \cdot e)$ instead of $\mathrm{RN}\left(y_h + \mathrm{RN}(y_\ell \cdot e)\right)$, and get a very slightly better value of e. This will be dealt with in Section 3.2.

2.3. A result on near-optimality

Let us show that the value e given by Theorem 2.1 is close to the best possible one, at least for some range of values of ϵ. To do so, consider ϵ of the form $\epsilon = (2^{-p-k+1} + 2^{-3p})/(2 - 2^{-p} + 2^{-3p})$, for $k > 1$ and $k < p/2$. For that ϵ, the minimum value of e given by Theorem 2.1 is

$$\frac{1+2^{-p}}{1-\epsilon-2^{p+1}\epsilon} = \left(\frac{2^k}{2^k-2}\right) + \left(\frac{4^k}{(2^k-2)^2}\right)\cdot 2^{-p} - \left(\frac{-4^k+2^{1+k}-8^k}{(2^k-2)^3}\right)\cdot 2^{-2p} + \dots$$

which is within around 2^{2-2k} from $1+2^{1-k}$, and above it. One can easily check that for

$$\begin{cases} y_h &= 2-2^{-p+1} \\ y_\ell &= 2^{-p}-2^{-p-k+1} \\ y &= 2-2^{-p}+2^{-3p} \\ e &= 1+2^{1-k}, \end{cases}$$

we have $|y_h + y_\ell - y| < \epsilon \cdot |y|$, $y_h = \mathrm{RN}(y_h + y_\ell)$, and $y_h = \mathrm{RN}\left(y_h + \mathrm{RN}(y_\ell \cdot e)\right)$, and yet $y_h \neq \mathrm{RN}(y)$. This shows that for that value of e, that is slightly below the minimum value provided by Theorem 2.1, we do not always have a correct Ziv rounding.

This was a kind of "generic" example (for arbitrary values of p). For particular values of p we have even more convincing examples. Assume the floating-point format is the binary64 (formerly called "double precision") format of the IEEE 754 Standard. We have $p = 53$. Consider

$$y = \frac{14619832736129378743570969657 22}{7769347642300524093767136003 23},$$

and assume that we have

$$\begin{aligned} \epsilon &= 2^{-80}, \\ y_h &= \frac{2118642268759237}{1125899906842624}, \\ y_\ell &= \frac{9007199188662643}{811296384146066816957890051440 64}. \end{aligned}$$

One easily finds that $|(y_h + y_\ell) - y| < \epsilon \cdot |y|$, more precisely, we have

$$\frac{|(y_h + y_\ell) - y|}{\epsilon \cdot |y|} \approx 1 - 2.54811 \cdot 10^{-37}.$$

The bound on e given by Theorem 2.1 is

$$e^* = \frac{1+2^{-p}}{1-\epsilon-2^{p+1}\epsilon} = \frac{1208925819614629308923904}{1208925801600230665224191}.$$

One may easily check that $\mathrm{RN}(y_h + \mathrm{RN}(e^* \cdot y_\ell)) \neq y_h$. Now, if we choose

$$e = \frac{4503599649443365}{4503599627370496}$$

then we have

— $\mathrm{RN}(y_h + \mathrm{RN}(e^* \cdot y_\ell)) = y_h$, whereas

$$\mathrm{RN}(y) = \frac{8474569075036949}{4503599627370496} \neq y_h,$$

which shows that the chosen value of e does not allow for a valid Ziv rounding test;

— and yet, $e/e^* = 0.9999999900000000987640\ldots$: the chosen value of e is therefore extremely close to the bound given by Theorem 2.1.

This example shows that, at least in some cases, Theorem 2.1 gives a very sharp bound.

2.4. Simpler bounds for e

In Section 2.2, we give a bound on the value of e, such that if e is larger than or equal to that bound, we have a correct Ziv rounding test. Let us call e^* that bound, namely,

$$e^* = \frac{1 + 2^{-p}}{1 - \epsilon - 2^{p+1}\epsilon}.$$

Computing e^* (or, more precisely, computing $\mathrm{RU}(e^*)$, which is the smallest value of e that is a floating-point number and that satisfies the condition given by Theorem 2.1) cannot be easily done in precision-p floating-point arithmetic. In most cases this is not a problem: an elementary function library is designed once for all, and will be used many times so that it is worth spending time to develop it, possibly using a multiple-precision environment. This might be different if we consider the automatic design of ad-hoc programs for specific functions. Let us now suggest two simpler bounds that are easily computable. Assume that $1 - 2^{p+1}\epsilon$ is exactly representable in precision-p floating-point arithmetic (this is not too much a constraint in practice: ϵ will frequently be a power of 2, and larger than 2^{-2p}). The two simpler bounds will be floating-point approximations to $(1 + 2^{-p+1})/(1 - 2^{p+1}\epsilon)$, more precisely, we will consider

$$e^{\uparrow} = \mathrm{RU}\left(\frac{1 + 2^{-p+1}}{1 - 2^{p+1}\epsilon}\right), \quad \text{and} \quad e^{\mathcal{N}} = \mathrm{RN}\left(\frac{1 + 2^{-p+1}}{1 - 2^{p+1}\epsilon}\right),$$

Notice that since $1 + 2^{-p+1}$ is a floating-point number, and since we have assumed that $1 - 2^{p+1}\epsilon$ is a floating-point number too, $e^{\uparrow}$ and $e^{\mathcal{N}}$ are straightforwardly computed. As Property 8 will show, $e^{\uparrow}$ is always larger than or equal to e^* and $e^{\mathcal{N}}$. We could not show that $e^{\mathcal{N}}$ is always larger than e^*, yet this is true in some important practical cases (See Property 9). However, using $e^{\uparrow}$, one is able to show the following result.

PROPERTY 8. *If $\epsilon \leq 2^{-p-3}$ and $1 - 2^{p+1}\epsilon$ is a floating-point number (for instance, this will be satisfied if ϵ is a power of two larger than or equal to 2^{-2p-1}) then $1 \leq e^{\uparrow}/e^* < 1 + 4 \cdot 2^{-p}$.*

PROOF. Define $\rho = e^{\uparrow}/e^*$. We easily find

$$\frac{\dfrac{1 + 2^{-p+1}}{1 - 2^{p+1}\epsilon}}{\dfrac{1 + 2^{-p}}{1 - \epsilon - 2^{p+1}\epsilon}} \leq \rho \leq \frac{\dfrac{\left(1 + 2^{-p+1}\right) \cdot \left(1 + 2^{-p+1}\right)}{1 - 2^{p+1}\epsilon}}{\dfrac{1 + 2^{-p}}{1 - \epsilon - 2^{p+1}\epsilon}}.$$

Let us denote $\omega = 1 - 2^{p+1}\epsilon$. We have $1 - \epsilon - 2^{p+1}\epsilon = \omega\left(1 - \frac{\epsilon}{\omega}\right)$, so that

$$\frac{\dfrac{1+2^{-p+1}}{\omega}}{\dfrac{1+2^{-p}}{\omega\left(1-\dfrac{\epsilon}{\omega}\right)}} \le \rho \le \frac{\dfrac{\left(1+2^{-p+1}\right)^2}{\omega}}{\dfrac{1+2^{-p}}{\omega\left(1-\dfrac{\epsilon}{\omega}\right)}},$$

which implies

$$\frac{1+2^{-p+1}}{1+2^{-p}}\left(1-\frac{\epsilon}{\omega}\right) \le \rho \le \frac{\left(1+2^{-p+1}\right)^2}{1+2^{-p}}\left(1-\frac{\epsilon}{\omega}\right) \le \frac{\left(1+2^{-p+1}\right)^2}{1+2^{-p}}.$$

Elementary manipulation shows that for all $p \ge 1$, we have $(1+2^{-p+1})/(1+2^{-p}) \ge 1 + 2^{-p} - 2^{-2p}$, and $(\left(1+2^{-p+1}\right)^2)/(1+2^{-p}) \le 1 + 4 \cdot 2^{-p}$. Furthermore, $\epsilon \le 2^{-p-3}$ implies $\omega \ge 3/4$ and $1 - \epsilon/\omega \le 1 - 2^{-p-1}/3$, from which we get, since $\left(1 + 2^{-p} - 2^{-2p}\right) \cdot \left(1 - 2^{-p-1}/3\right) = 1 + \frac{5}{6}2^{-p} - \frac{7}{6}2^{-2p} + \frac{1}{6}2^{-3p}$ is larger than 1 as soon as $p \ge 1$,

$$1 \le \rho < 1 + 4 \cdot 2^{-p}.$$

□

Now, for special values of p (namely, those corresponding to the binary interchange formats of the IEEE Standard for Floating-Point Arithmetic and to the INTEL "double-extended precision" format), and special values of ϵ (those of the form $\gamma \cdot 2^{-p-k}$, with $\gamma = 1$, $3/4$, $5/8$, or $7/8$, and $3 \le k \le p+1$), one can get an even more interesting result:

PROPERTY 9. *If $p \in \{11, 24, 53, 64, 113\}$, and $\epsilon = \gamma \cdot 2^{-p-k}$, with $\gamma = 1$, $3/4$, $5/8$, or $7/8$, and $3 \le k \le p+1$, then $e^* \le e^{\mathcal{N}} < e^* + \mathrm{ulp}\,(e^*)$.*

To prove that property, we just enumerated all possible cases. When the conditions of Property 9 hold, $e^{\mathcal{N}}$ is equal to the smallest floating-point number larger than or equal to e^*: it is therefore not possible to deduce a better constant e from Theorem 2.1.

3. ADDITIONAL RESULTS

3.1. When the test fails, it gives us information anyway

Let us now see that, under mild conditions on ϵ and e, when we perform a correct Ziv rounding test and the test fails (i.e., we cannot be sure that $y_h = \mathrm{RN}(y)$), we get some information anyway. More precisely, we have the following property:

THEOREM 3.1. *If*

— *$p \ge 2$;*
— *$\epsilon < 1/(2^{p+3} + 9)$;*
— *the chosen value of e is*

$$\mathrm{RU}\left(\frac{1+2^{-p}}{1-\epsilon-2^{p+1}\epsilon}\right)$$

(i.e., the smallest one for which Theorem 2.1 guarantees that we have a correct Ziv rounding test),

then when the test fails, i.e., when we have

$$y_h + y_\ell = y \cdot (1+\alpha), \qquad \text{with} \quad |\alpha| \le \epsilon, \tag{13}$$

$$y_h = \mathrm{RN}(y_h + y_\ell),$$

and

$$y_h \neq \mathrm{RN}(y_h + \mathrm{RN}(e \cdot y_\ell))$$

the two numbers y_h *and* $y_c = \mathrm{RN}(y_h + \mathrm{RN}(e \cdot y_\ell))$ *are consecutive floating-point numbers, and* y *is between them, which implies that* $\mathrm{RN}(y)$ *is either* y_h *or* y_c.

Theorem 3.1 is especially useful when implementing a locally monotonic elementary function f at point x. It essentially means that it suffices to approximate

$$f^{-1}\left(\frac{y_h + y_c}{2}\right)$$

with large enough accuracy to determine if the correct value to be returned is y_h or y_c. This is of interest when f^{-1} is simpler to approximate than f (e.g., when f is the arcsin function). Theorem 3.1 is also useful when implementing heterogeneous operations. Consider the problem presented in Section 1.2.2. When the test fails, assuming $q \geq p+1$, $(y_h + y_c)/2$ is a precision-q floating-point number, so that it can easily be compared to $a^{(q)} + b^{(q)}$.

PROOF. Without loss of generality, we assume that y_h and y are positive. Assume $p \geq 2$ and $\epsilon < 1/(2^{p+3} + 9)$. First, elementary manipulation shows that

$$\frac{1 + 2^{-p}}{1 - \epsilon - 2^{p+1}\epsilon} < 2,$$

which implies $e \leq 2$.
Now, from $y_h = \mathrm{RN}(y_h + y_\ell)$ and $e \leq 2$, we easily deduce that

— $|\mathrm{RN}(ey_\ell)| \leq \mathrm{ulp}(y_h)$ if y_h is not a power of 2 or y_ℓ has the same sign as y_h; and
— $|\mathrm{RN}(ey_\ell)| \leq \frac{1}{2}\mathrm{ulp}(y_h)$ if y_h is a power of 2 and y_ℓ has not the same sign as y_h.

A consequence of this is that $y_c = \mathrm{RN}(y_h + \mathrm{RN}(e \cdot y_\ell))$ *is either* y_h *or the floating-point predecessor or successor of* y_h.

Since $e \leq 2$, one easily deduces that if $|y_\ell| \leq \frac{1}{8}\mathrm{ulp}(y_h)$ then $|\mathrm{RN}(ey_\ell)| \leq \frac{1}{4}\mathrm{ulp}(y_h)$, so that $y_c = \mathrm{RN}(y_h + \mathrm{RN}(e \cdot y_\ell)) = y_h$ (this is straightforward if y_h is not a power of two, and this is a consequence of the round to nearest *even* rule if y_h is a power of two). As a consequence, if $y_c \neq y_h$ then, necessarily, $|y_\ell| > \frac{1}{8}\mathrm{ulp}(y_h) > 2^{-p-3}y_h$. Now, from (13) we have

$$y - y_h = y_\ell\left(1 - \alpha\frac{y}{y_\ell}\right) \tag{14}$$

with $|\alpha| \leq \epsilon$. We have just shown above that $y_c \neq y_h$ implies

$$\left|\frac{y_h}{y_\ell}\right| < 2^{p+3}. \tag{15}$$

Also, $\mathrm{RN}(y_h + y_\ell) = y_h$ implies

$$\left|\frac{y_h}{y_\ell}\right| \geq 2^p. \tag{16}$$

We have

$$y = \frac{y_h + y_\ell}{1 + \alpha},$$

therefore, using (16),

$$\left|\frac{y}{y_\ell}\right| \leq \frac{y_h + |y_\ell|}{|y_\ell|(1+\alpha)} \leq \frac{y_h(1+2^{-p})}{|y_\ell| \cdot (1-\epsilon)},$$

therefore, using (15),

$$\left|\frac{y}{y_\ell}\right| < 2^{p+3} \cdot \frac{1+2^{-p}}{1-\epsilon},$$

therefore

$$|\alpha| \cdot \left|\frac{y}{y_\ell}\right| < 2^{p+3} \cdot \left(1+2^{-p}\right) \cdot \frac{\alpha}{1-\epsilon} \leq 2^{p+3} \cdot \left(1+2^{-p}\right) \cdot \frac{\epsilon}{1-\epsilon}. \tag{17}$$

Elementary manipulation shows that the right-hand side of (17) will be less than 1 as soon as $\epsilon < 1/(2^{p+3}+9)$. Hence, using (14), as soon as $\epsilon < 1/(2^{p+3}+9)$, if $y_c \neq y_h$ then i) $y - y_h$ and y_ℓ have the same sign, and ii) y is less than or equal to the floating-point successor of y_h, and larger than or equal to its floating-point predecessor. Therefore:

— if $y_\ell > 0$ then $y > y_h$ and (straightforwardly), $y_c > y_h$;
— if $y_\ell < 0$ then $y < y_h$ and (straightforwardly), $y_c < y_h$.

Hence, when the test fails, y is between the two consecutive floating-point numbers y_c and y_h, which implies that $\mathrm{RN}(y)$ is either y_h or y_c. □

3.2. When an fma instruction is available

If an fma instruction is available, we can use it to change the test (5) into another one that is very slightly better in the general case and, more important, that works even if $(1/4)\mathrm{ulp}(y_h)$ is subnormal. The fma instruction evaluates expressions of the form$\mathrm{RN}(a+bc)$.It is available on processors such at the Intel Itanium, IBM PowerPC, AMD Bulldozer, and will be available on the Intel Haswell. It allows for faster and, in general, more accurate dot products, matrix multiplications, and polynomial evaluations. The fma instruction is included in the new IEEE 754-2008 standard for floating-point arithmetic. If an fma instruction is available, a natural idea is to replace the test

is y_h *equal to* $\mathrm{RN}(y_h + \mathrm{RN}(y_\ell \cdot e))$ *?*

by the test

is y_h *equal to* $\mathrm{RN}(y_h + y_\ell \cdot e)$ *?*

Analyzing that new test, that is, trying, given ϵ, to find a bound e^* on e such that if $e \geq e^*$ then

$$y_h = \mathrm{RN}\left(y_h + y_\ell \cdot e\right) \Rightarrow y_h = \mathrm{RN}(y), \tag{18}$$

is done in a way very similar to what we did in Section 2. Assuming y_h is in the normal range, Property 3 becomes:

Assume y_h *is not a power of* 2*. If* $y_h = \mathrm{RN}(y_h + y_\ell \cdot e)$ *then* $|y_\ell| \leq \mathrm{ulp}(y_h)/2e$.

That is, compared to Property 3, we get rid of the term $(1+2^{-p})$ and of the requirement that $(1/2)\mathrm{ulp}(y_h)$ should be in the normal range. A consequence is that we have a property analogous to Property 4, without that term and that requirement. Properties 6 and 7 are transformed in a similar way, and we deduce:

THEOREM 3.2. *Assume that* y_h *is a normal floating-point number and that* ϵ *is less than* $1/(2^{p+1}+1)$*. Also assume that* $y_h = \mathrm{RN}(y_h + y_\ell)$ *and* $|(y_h + y_\ell) - y| < \epsilon \cdot |y|$*. If* $e \geq 1/(1-\epsilon-2^{p+1}\epsilon)$ *then* $y_h = \mathrm{RN}\left(y_h + y_\ell \cdot e\right)$ *implies* $y_h = \mathrm{RN}(y)$.

3.3. If $(1/4)\mathrm{ulp}(y_h)$ is subnormal and there is no fma instruction

Now, if $(1/4)\mathrm{ulp}(y_h)$ is subnormal, then the relative error committed when rounding $y_\ell \cdot e$ may be very large. If we try to adapt the proof of Proposition 3, we easily find that from

$$|\mathrm{RN}(y_\ell \cdot e)| \leq \frac{1}{2}\mathrm{ulp}(y_h)$$

the only thing we can deduce is $|y_\ell \cdot e| \leq (3/4)\mathrm{ulp}(y_h)$. Therefore, the bound of Theorem 2.1 becomes

$$e \geq \frac{3/2}{1-\epsilon-2^{p+1}\epsilon} \tag{19}$$

which is quite bad. Unfortunately, at least in some cases, values of e slightly below that bound will fail. Consider for instance:

$$\begin{cases} y &= 2^{k_{\min}+1} + 2^{k_{\min}-p+1} + 2^{k_{\min}-2p} \\ y_h &= 2^{k_{\min}+1} \\ y_\ell &= 2^{k_{\min}-p+1} \\ \epsilon &= 2^{-2p-1}. \end{cases}$$

For which we have $y_h = \mathrm{RN}(y_h + y_\ell)$, and $|(y_h + y_\ell) - y| \leq \epsilon \cdot |y|$. The bound on e provided by (19) is

$$e^* = \frac{3/2}{1-2^{-p}-2^{-2p-1}} \approx \frac{3}{2} \cdot \left(1+2^{-p}\right).$$

If we choose the following value of e, which is only very slightly below e^*:

$$e = \frac{3}{2} - 2^{-p+1} = \underbrace{1.0111\cdots1}_{p \text{ bits}},$$

then we have $\mathrm{RN}(y_\ell \cdot e) = 2^{k_{\min}-p+1}$, so that $\mathrm{RN}(y_h + \mathrm{RN}(y_\ell \cdot e)) = y_h$, and yet,

$$\mathrm{RN}(y) = 2^{k_{\min}+1} + 2^{k_{\min}-p+2} \neq y_h.$$

This example shows that if $\mathrm{ulp}(y_h)/4$ can be subnormal, it is much preferable to perform Ziv's rounding test using an fma instruction. This is a real concern for the implementation of the heterogeneous operations: small values must be handled separately. However, if we aim at evaluating usual transcendental functions with correct rounding, this problem will almost never need to be considered. Assume we wish to compute $f(x)$, for an input floating-point number x such that $f(x)$ is very near zero:

— if x is very near zero too, for most usual functions, we can directly deduce the correctly rounded value of $f(x)$ from the Taylor series expansion of f at 0. Consider for instance the case of the sine function. Assume we wish to evaluate $\sin(x)$, where $x > 0$ is a tiny floating-point number. Let $x = m_x \cdot 2^{e_x}$, with $1 \leq m_x < 2$. Assume x is not a power of 2 (that case is easily handled separately) If $x < \sqrt{3} \cdot 2^{-p/2}$, then $m_x \cdot x^2 < 6 \cdot 2^{-p}$, so that $m_x^3 \cdot 2^{3e_x} < 6 \cdot 2^{e_x - p}$, which implies $x^3/6 < 2^{e_x-p} = (1/2)\mathrm{ulp}(x)$. As a consequence, all values between $x - x^3/6$ and x will round to x, which implies $\mathrm{RN}(\sin(x)) = x$. Similar reasonings can be done for various functions and, possibly, better bounds can be obtained for particular values of p. Table I shows how to handle many usual functions in the double-precision/binary64 format of IEEE 754.

— in practice, for larger arguments, when x is a floating-point number, $f(x)$ is always far from the subnormal range, even if, in theory, there is a root of f near x: for

Table I. Correctly rounding some functions for tiny arguments in the double-precision/binary64 format, assuming rounding to nearest (that table is extracted from [Muller et al. 2010]). If x is a real number, we let x^- denote the largest floating-point number strictly less than x, and x^+ denote the smallest floating-point number strictly larger than x.

This function	can be replaced by	when
$\exp(\epsilon) - 1$	ϵ	$\lvert\epsilon\rvert < \mathrm{RN}(\sqrt{2}) \times 2^{-53}$
$\exp(\epsilon) - 1, \epsilon \geq 0$	ϵ^+	$\mathrm{RN}(\sqrt{2}) \times 2^{-53} \leq \epsilon < \mathrm{RN}(\sqrt{3}) \times 2^{-52}$
$\ln(1+\epsilon)$	ϵ	$\lvert\epsilon\rvert < \mathrm{RN}(\sqrt{2}) \times 2^{-53}$
$\sin(\epsilon), \sinh(\epsilon), \sinh^{-1}(\epsilon)$	ϵ	$\lvert\epsilon\rvert \leq \alpha = \mathrm{RN}(3^{1/3}) \times 2^{-26}$
$\arcsin(\epsilon)$	ϵ	$\lvert\epsilon\rvert < \alpha = \mathrm{RN}(3^{1/3}) \times 2^{-26}$
$\tan(\epsilon), \tanh^{-1}(\epsilon)$	ϵ	$\lvert\epsilon\rvert < \eta = \mathrm{RN}(12^{1/3}) \times 2^{-27}$
$\tanh(\epsilon), \arctan(\epsilon)$	ϵ	$\lvert\epsilon\rvert \leq \eta$

instance, for the trigonometric functions, using a continued-fraction argument first presented by Kahan [Kahan 1983], one can show that the double-precision/binary64 number larger than 1 which is nearest an integer multiple of $\pi/2$ is

$$6381956970095103 \times 2^{+797}$$

whose cosine is around $-4.69 \times x^{-19}$: $1/4$ ulp of that value is far above the subnormal threshold.

4. EXPERIMENTS

4.1. General remarks

We have limited our tests to IEEE binary64 floating-point numbers ($p = 53$): both y_h and y_ℓ are of the type "double" of the C language. We (allegedly) tried to approximate an arbitrary precision number y, with relative error bounded by ϵ. We also have limited ourselves to peculiar values of y_ℓ, where it is positive (the negative case is symmetric), and close to $(1/2)\mathrm{ulp}(y_h)$ since, considering the small values of e that we obtain for reasonable values of ϵ, nothing special happens for values of y_ℓ notably smaller than $(1/2)\mathrm{ulp}(y_h)$. The value of y is chosen extremely close to the limit imposed by ϵ. When y_ℓ is very close to $(1/2)\mathrm{ulp}(y_h)$, y will frequently be larger than or equal to $y_h + (1/2)\mathrm{ulp}(y_h)$ unless ϵ is very small. Therefore our experiments can be viewed as mimicking a worst case situation, where y is always cornered close to the upper bound rather than having its value scattered all over the interval $((y_h + y_\ell) \times (1-\epsilon), (y_h + y_\ell) \times (1+\epsilon))$.

Every result of our tests falls in one of the following categories:

— **positive:** $y_h = \mathrm{RN}(y_h + RN(y_\ell \times e))$ and $y_h = \mathrm{RN}(y)$. In such a case, we are done: Ziv's rounding test indicates that y_h is the correct rounding of y;
— **false positive:** $y_h = \mathrm{RN}(y_h + \mathrm{RN}(y_\ell \times e))$ and $y_h \neq \mathrm{RN}(y)$. That case must never occur, since it means that the rounding test is not correct;
— **negative:** $y_h \neq \mathrm{RN}(y_h + \mathrm{RN}(y_\ell \times e))$ and $y_h \neq \mathrm{RN}(y)$. In such a case, a different strategy (e.g. a tighter approximation) should be used;
— **false negative:** $y_h \neq \mathrm{RN}(y_h + \mathrm{RN}(y_\ell \times e))$ and $y_h = \mathrm{RN}(y)$, which causes us to unduly switch to a more expensive strategy. This case must occur as scarcely as possible.

More precisely, the various parameters involved in our experiments satisfy:

— $y_h \in [1, 2)$;
— $y_\ell \in [15/32\mathrm{ulp}(y_h), 16/32\mathrm{ulp}(y_h))$;

— $\epsilon \in \left(2^{-81}, 2^{-55}\right]$;
— as a consequence, $e \in [1.0000000074505808, 2.0000000000000004]$.

4.2. Ziv rounding test and ϵ (for $p = 53$)

The results of our experiments ($100,000,000$ values ϵ chosen randomly for each interval $[2^{-k}, 2^{-k+1})$, for k from -81 to -56, one random y_h in $[1, 2)$ and one random y_ℓ in $[15/32\text{ulp}(y_h), 16/32\text{ulp}(y_h))$ for each ϵ) are presented in Table II. For the largest

Table II. Results of our experiments, for $\epsilon \in \left(2^{-81}, 2^{-55}\right]$

ϵ	Positives	False Positives	Negatives	False Negatives
$[2^{-55}, 2^{-56}[$	0	0	100 000 000	0
$[2^{-56}, 2^{-57}[$	0	0	99 997 603	2397
$[2^{-57}, 2^{-58}[$	23 284 922	0	76 116 019	599 059
$[2^{-58}, 2^{-59}[$	61 962 104	0	37 740 947	296 949
$[2^{-59}, 2^{-60}[$	80 926 366	0	18 924 027	149 607
$[2^{-60}, 2^{-61}[$	90 593 819	0	9 332 714	73 467
$[2^{-61}, 2^{-62}[$	95 167 993	0	4 794 367	37 640
$[2^{-62}, 2^{-63}[$	97 625 196	0	2 356 226	18 578
$[2^{-63}, 2^{-64}[$	98 776 533	0	1 213 736	9 731
$[2^{-64}, 2^{-65}[$	99 399 403	0	596 044	4 553
$[2^{-65}, 2^{-66}[$	99 703 144	0	294 513	2 343
$[2^{-66}, 2^{-67}[$	99 853 799	0	145113	1 088
$[2^{-67}, 2^{-68}[$	99 924 441	0	74 914	645
$[2^{-68}, 2^{-69}[$	99 962 425	0	37 261	314
$[2^{-69}, 2^{-70}[$	99 981 576	0	18286	138
$[2^{-70}, 2^{-71}[$	99 990 768	0	9 154	78
$[2^{-71}, 2^{-72}[$	99 995 282	0	4 677	41
$[2^{-72}, 2^{-73}[$	99 997 740	0	2 239	21
$[2^{-73}, 2^{-74}[$	99 998 842	0	1 148	10
$[2^{-74}, 2^{-75}[$	99 999 413	0	584	3
$[2^{-75}, 2^{-76}[$	99 999 715	0	284	1
$[2^{-76}, 2^{-77}[$	99 999 853	0	147	0
$[2^{-77}, 2^{-78}[$	99 999 928	0	70	2
$[2^{-78}, 2^{-79}[$	99 999 955	0	44	1
$[2^{-79}, 2^{-80}[$	99 999 981	0	19	0
$[2^{-80}, 2^{-81}[$	99 999 991	0	9	0

values of ϵ, the values of e are so large that test fails systematically (yielding a "negative" result) because y is certain to be larger than or equal to $y_h + \text{ulp}(y_h)$ and $RN(y_\ell \times e) \geq (1/2)\text{ulp}(y_h)$. As ϵ decreases, so does e and y has less latitude to wander beyond $y_h + \text{ulp}(y_h)$. As a consequence, the number of "negatives" decreases and so does the number of "false negatives". At some point, the "false negatives" seem to totally vanish: the probability of a false negative is too low for there to be one among the tested cases, and all we have left is a small and decreasing (by a factor 2 each time ϵ itself decreases by a factor 2) number of "negatives". The comforting result is that, at no point, do we have a "false positive".

4.3. Near optimality of the value of e given by Theorem 2.1

To minimize the number of "false negatives" e must be as small as possible (and yet large enough to allow for a correct Ziv testing). Hence, it is important to know how far from optimal is the value of e given by Theorem 2.1. The best constant that can be

deduced from Theorem 2.1 is $\mathrm{RU}(e^*) = \mathrm{RU}((1+2^{-p})/(1-\epsilon-2^{p+1}\epsilon))$. We want to know how much we can subtract from $\mathrm{RU}(e^*)$ before we start getting "false positive" values. In several experiments, we got our first false positive value at $\mathrm{RU}(e^*) - 2^{21} \times \mathrm{ulp}(e^*)$. "False positive" cases obtained in a specific run, with value $\mathrm{RU}(e^*) - 2^{21} \times \mathrm{ulp}(e^*)$, are given in Table III.

Table III. False positives for $e = \mathrm{RU}(e^*) - 2^{21} \times \mathrm{ulp}(e^*)$

Example 1		
$y_h = 1.9947587312896187$	$y_\ell = 1.1102229216237452e-16$	$\epsilon = 5.1757461373254229e-24$
$\mathrm{RN}(y) = 1.9947587312896189$	$y = 1.99475873128961878055$ $77597160974603068928590135$ $23728434792028406527975734$ 416324	$e = 1.000000092772301$
Example 2		
$y_h = 1.9883894880686108$	$y_\ell = 1.110222993284877e-16$	$\epsilon = 1.5927638864921235e-24$
$\mathrm{RN}(y) = 1.988389488068611$	$y = 1.98838948806861093299$ $14757256952656859493280554$ $75352754840860468630843325$ 121040	$e = 1.0000000282270229$
Example 3		
$y_h = 1.9802155978993843$	$y_\ell = 1.1102230085276806e-16$	$\epsilon = 8.2224179116152375e-25$
$\mathrm{RN}(y) = 1.9802155978993845$	$y = 1.98021559789938439433$ $14345963756318263894453847$ $32167637700306671773559863$ 566352	$e = 1.0000000143465304$

4.4. Simplified and accurate formulas

As noticed in Section 2.4, two simpler bounds, $e^{\mathcal{N}}$ and $e^{\uparrow}$, can be computed in precision-p floating-point arithmetic. Since $e^{\mathcal{N}}$ is always less than or equal to $e^{\uparrow}$, it is preferable to choose $e = e^{\mathcal{N}}$, whenever we are sure that this yields to a correct Ziv rounding test. Unfortunately, $e^{\mathcal{N}}$ is not always larger than or equal to the bound e^* of Theorem 2.1 (we could only show that for some specific values of ϵ, for the binary formats specified by the IEEE 754-2008 Standard, see Property 9). Below are the results of some tests done in the binary64 format, for approximation accuracy bounds ϵ varying from 2^{-56} to 2^{-90}. Notice that in all the previous developments, we have considered $\epsilon < 2^{-p-1}$. In our case that should give $\epsilon < 2^{-54}$. But when $\epsilon \in (2^{-56}, 2^{-55}]$ we have found many cases where $e^{\mathcal{N}} < e^*$. This is a limited annoyance since, as can be found up from section 4.2, in this approximation accuracy interval the test is far from pertinent (all our corner cases are "negative", but that could also be true for many of then in more "conventional" situations): in practice, much smaller values of ϵ are considered.

Table IV. Comparison between $e^{\mathcal{N}}$ and e^*, for tests done in the binary64 format, and error bounds ϵ varying from 2^{-56} to 2^{-90}.

max relative error $(e^{\mathcal{N}} - \mathrm{RU}(e^*))/\mathrm{RU}(e^*)$	2.2204460492339477e-16
$e^{\mathcal{N}} = \mathrm{RU}(e^*)$ cases	1 514 615 533
Total number of cases	3 500 000 000

5. CONCLUSION

We have given and proven conditions on the constant e of Ziv's rounding test that allow one to make sure that the test is always reliable (i.e., we never have "false positives"). Furthermore, these conditions are very tight, and for values of the approximation error ϵ that make sense in practice, the probability of observing a "false negative" (i.e., the test makes us believe that a more accurate computation is required, whereas y_h was already the right answer) is very low. This makes Ziv's rounding test an excellent solution for checking whether we can return a correctly-rounded result or not. There are several already-existing applications of that test for implementing correctly-rounded elementary functions, and there are interesting potential applications for implementing "heterogeneous" arithmetic operations.

REFERENCES

BOLDO, S. AND MELQUIOND, G. 2008. Emulation of FMA and correctly rounded sums: proved algorithms using rounding to odd. *IEEE Transactions on Computers 57,* 4, 462–471.

DE DINECHIN, F., ERSHOV, A. V., AND GAST, N. 2005. Towards the post-ultimate libm. In *Proceedings of the 17th IEEE Symposium on Computer Arithmetic*. ARITH '05. IEEE Computer Society, Washington, DC, USA, 288–295.

DEKKER, T. J. 1971. A floating-point technique for extending the available precision. *Numerische Mathematik 18,* 3, 224–242.

IEEE COMPUTER SOCIETY. 2008. *IEEE Standard for Floating-Point Arithmetic*. IEEE Standard 754-2008. available at http://ieeexplore.ieee.org/servlet/opac?punumber=4610933.

JEANNEROD, C.-P., LOUVET, N., MULLER, J.-M., AND PANHALEUX, A. 2011. Midpoints and exact points of some algebraic functions in floating-point arithmetic. *IEEE Transactions on Computers 60,* 2.

KAHAN, W. 1983. Minimizing q*m-n. Text accessible electronically at http://http.cs.berkeley.edu/~wkahan/. At the beginning of the file "nearpi.c".

KAHAN, W. 2004. A logarithm too clever by half. Available at http://http.cs.berkeley.edu/~wkahan/LOG10HAF.TXT.

KNUTH, D. 1998. *The Art of Computer Programming* 3rd Ed. Vol. 2. Addison-Wesley, Reading, MA.

MONNIAUX, D. 2008. The pitfalls of verifying floating-point computations. *ACM TOPLAS 30,* 3, 1–41. A preliminary version is available at http://hal.archives-ouvertes.fr/hal-00128124.

MULLER, J.-M. 2006. *Elementary Functions, Algorithms and Implementation* 2nd Ed. Birkhäuser Boston, MA.

MULLER, J.-M., BRISEBARRE, N., DE DINECHIN, F., JEANNEROD, C.-P., LEFÈVRE, V., MELQUIOND, G., REVOL, N., STEHLÉ, D., AND TORRES, S. 2010. *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston. ACM G.1.0; G.1.2; G.4; B.2.0; B.2.4; F.2.1., ISBN 978-0-8176-4704-9.

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

OGITA, T., RUMP, S. M., AND OISHI, S. 2005. Accurate sum and dot product. *SIAM Journal on Scientific Computing 26,* 6, 1955–1988.

PICHAT, M. 1972. Correction d'une somme en arithmétique à virgule flottante. *Numerische Mathematik 19*, 400–406. In French.

ZIV, A. 1991. Fast evaluation of elementary mathematical functions with correctly rounded last bit. *ACM Transactions on Mathematical Software 17,* 3, 410–423.