Theorems on Efficient Argument Reductions

Ren-Cang Li*
Department of Mathematics
University of Kentucky
Lexington, KY 40506
Email: rcli@ms.uky.edu

Sylvie Boldo, Marc Daumas
Laboratoire de l'Informatique du Parallélisme
UMR 5668 CNRS - ENS de Lyon - INRIA
Email: Sylvie.Boldo@ens-lyon.fr
Marc.Daumas@ens-lyon.fr

Abstract

A commonly used argument reduction technique in elementary function computations begins with two positive floating point numbers α and γ that approximate (usually irrational but not necessarily) numbers $1/C$ and C, e.g., $C = 2\pi$ for trigonometric functions and $\ln 2$ for e^x. Given an argument to the function of interest it extracts z as defined by $x\alpha = z + \varsigma$ with $z = k2^{-N}$ and $|\varsigma| \leq 2^{-N-1}$, where k, N are integers and $N \geq 0$ is preselected, and then computes $u = x - z\gamma$. Usually $z\gamma$ takes more bits than the working precision provides for storing its significand, and thus exact $x - z\gamma$ may not be represented exactly by a floating point number of the same precision. This will cause performance penalty when the working precision is the highest available on the underlying hardware and thus considerable extra work is needed to get all the bits of $x - z\gamma$ right. This paper presents theorems that show under mild conditions that can be easily met on today's computer hardware and still allow $\alpha \approx 1/C$ and $\gamma \approx C$ to almost the full working precision, $x - z\gamma$ is a floating point number of the same precision. An algorithmic procedure based on the theorems is obtained. The results will enhance performance, in particular on machines that has hardware support for fused-multiply-add (`fma`*) instruction(s).*

1 Introduction

Table-based methods to compute elementary functions rely on efficient argument reduction techniques. The idea is to reduce an argument x to u that falls into a tiny interval to allow efficient polynomial approximations (see [5, 10, 11, 12, 13, 15, 17, 18, 19, 20] and references therein).

*This work was supported in part by the National Science Foundation under Grant No. ACI-9721388 and by the National Science Foundation CAREER award under Grant No. CCR-9875201. Part of this work was done while this author was on leave at Hewlett-Packard Company.

By default in this paper, all floating point numbers (FPNs), unless otherwise explicitly stated, are binary and of the same type and with p bits in the significand, hidden bits (if any) included, and thus the machine roundoff is

$$\epsilon_{\mathsf{m}} = 2^{-p}.$$

Also we shall assume the default rounding mode is *round-to-nearest* or to even in the case of a tie [1, 6, 14] unless otherwise explicitly stated differently. The underlying machine hardware conforms to the IEEE floating point standard [1].

A commonly used argument reduction technique begins with two positive FPNs α and γ that approximate (usually irrational but not necessarily) numbers $1/C$ and $C > 0$, and thus $\alpha\gamma \approx 1$. Examples include $C = \pi/2$ or π or 2π for trigonometric functions $\sin x$ and $\cos x$, and $C = \ln 2$ for exponential function e^x. Let x be a given argument, a FPN of course. The argument reduction starts by extracting z as defined by

$$x \cdot \frac{1}{C} \approx x\alpha = z + \varsigma = \boxed{k2^{-N}}\boxed{\varsigma}, \quad (1.1)$$

where k is an integer, and $|\varsigma| \leq 2^{-N-1}$, where $N \geq 0$ is an integer. Then it computes a reduced argument

$$u = x - z\gamma. \quad (1.2)$$

For IEEE single precision elementary functions, this u is often good enough, provided $\alpha \approx 1/C$ and $\gamma \approx C$ are IEEE double precision approximations carefully chosen and IEEE double precision arithmetic is used. But sometimes better approximations to C than γ may be necessary for accuracy considerations, e.g., when creating IEEE double precision elementary functions on any of today's RISC (Reduced Instruction Set Computers) machines. If this is the case, often another FPN γ_{L}, roughly containing the next p bits in the significand of C so that the unevaluated $\gamma + \gamma_{\mathsf{L}} \approx C$ to about $2p$ bits in the significand, is made available to overwrite the u in (1.2) by

$$u - z\gamma_{\mathsf{L}} \quad (1.3)$$

IEEE COMPUTER SOCIETY

Whether the u by (1.2) or this updated one is accurate enough for computing the elementary function in question is subject to further error analysis on function-by-function basis. But this is out of the scope of this paper.

On machines that have hardware support for the fused-multiply-add (`fma`) instructions, such as machines with HP/Intel Itanium Microprocessors [11] and IBM PowerPC Microprocessors. The computation of z can be done efficiently as

$$\{x\alpha + \sigma\}_{\mathsf{fma}} - \sigma,$$

where σ is a pre-chosen constant[1]. Given the trend of getting `fma` as a callable function (inlinable by compilers at certain optimization level) to the language standards such as the C99 standard [2] and the new FORTRAN standard currently under development, this technique is available to users who program only in high level languages.

Notice that if k is an ℓ bit integer, it takes up to $p + \ell$ bits to store the significand of $z\gamma$. It is conceivable that some bits of x and $z\gamma$ will cancel each other, but it is not clear how many of them will and under what condition(s), and consequently if accuracy calls for $x - z\gamma$ to be calculated exactly (or to more than p bits in the significand), how do we get these bits efficiently? This question is especially critical if the working precision is the highest available on the underlying computing platform. In this paper, we will show with mild conditions that can be easily met, $x - z\gamma$ can be represented exactly by a FPN, and thus it can be computed by an instruction of the `fma` type without error. While this does not exclude the possibility of any further updating as in (1.3) if deemed necessary, it does eliminate any expensive procedure[2] to compute correctly all the bits of $x - z\gamma$ had we not known that it were a FPN. Our results will enhance performance, in particular on machines that have hardware support for `fma` instructions.

The phenomena of $x - z\gamma$ being a FPN was mentioned in [5], but no further detail was provided there.

Throughout this paper, all FPNs in question are normalized. This is not as restrictive an assumption as it seems. Because those α and γ from elementary function computations are far from subnormal FPNs, and when x is subnormal, it is so tiny that no argument reduction is ever needed. Even if subnormal x is, say, passed to $\{x\alpha + \sigma\}_{\mathsf{fma}} - \sigma$, z will be computed to 0 and thus $u = x - z\gamma = x$ as we would like it to.

The rest of this paper is organized as follows. Section 2 presents a theorem on the number of cancelled bits of two close FPNs that will be used repeatedly in the next section. The theorem, which is of interest in its own right, is an extension of the well-known theorem due to Sterbenz [16]. Our main result is given in Section 3. In Section 4, we analyze how to satisfy the conditions of the theorem in Section 3. Combining the results of Sections 3 and 4, Section 5 presents an algorithm for α and γ, given C and $p(\geq 3)$, and its applications to $C = \ln 2$ for the exponential function $\exp(x)$ and $C = 2\pi$ for the trigonometric functions. Section 6 concludes the work of this paper.

Notation. Throughout, $\oplus, \ominus, \otimes, \oslash$ denote the floating point addition, substraction, multiplication, and division, respectively. $\{\mathcal{X}\}_{\mathsf{fma}}$ denotes the result by an instruction of the fused-multiply-add type, i.e., the exact $\mathcal{X}$ after only one rounding, where $\mathcal{X}$ is one of $\pm a \pm bc$ and $\pm a \mp bc$. ":=" defines the left-hand side to be the right-hand side. $\lfloor a \rfloor$ is the biggest integer that is no greater than a. $\mathsf{round_to_nearest}(a)$ is the FPN obtained from rounding a in the round-to-nearest mode, and $\mathsf{round_up}(a)$ is the smallest FPN that is no smaller than a, and $\mathsf{ulp}(b) := 2^{m-p+1}$ is the unit in the last place of a FPN $b = \pm 2^m \times 1.b_1 b_2 \cdots b_{p-1}$.

2 Exact Subtraction Theorems

Throughout this section a and b are assumed nonnegative. But minor modifications can make all results valid for non-positive a and b, too.

A well-known property [4, 6, 16] of the floating point subtraction is the following.

Theorem 2.1 (Sterbenz) *Let a and b be two FPNs. If $b/2 \leq a \leq 2b$, then $a \ominus b = a - b$, i.e., $a \ominus b$ is exact.*

We now extend this theorem to

Theorem 2.2 *Let a and b be two FPNs with $p + \ell$ bits in the significand, where integer $\ell \geq 0$. If*

$$b/2 \leq (1 + 2^{-\ell})^{-1} b \leq a \leq (1 + 2^{-\ell}) b \leq 2b, \qquad (2.1)$$

then $a - b$ is a (default) FPN, i.e., $a - b$ can be represented exactly by a FPN with p bits in the significand.

Remark 2.1 Like Theorem 2.1 of Sterbenz, Theorem 2.2 can be shown to hold for radix other than 2. An automatic machine proof in Coq for this is available from the second author upon request or by visiting

[1] This idea appeared in Markstein [11, Chap. 10] who told the first author that he got it from Clemens Roothaan.

[2] Without knowing $x - z\gamma$ is a FPN, a typical procedure to compute it exactly may look like this piece of code:

$$\begin{aligned} &a_{\mathsf{H}} = z \otimes \gamma; \\ &v = x \ominus a_{\mathsf{H}}; \quad a_{\mathsf{L}} = \{z\gamma - a_{\mathsf{H}}\}_{\mathsf{fma}}; \\ &u_{\mathsf{H}} = v \ominus a_{\mathsf{L}}; \\ &b = v \ominus u_{\mathsf{H}}; \\ &u_{\mathsf{L}} = b \ominus a_{\mathsf{L}}; \end{aligned}$$

This is at least five times as slow as by $\{x - z\gamma\}_{\mathsf{fma}}$ had we known that it were a FPN. Here we assume that there must be some cancellations in the leading bits of x and a_{H} and thus $|v| \geq |a_{\mathsf{L}}|$. The return value is $u_{\mathsf{H}} + u_{\mathsf{L}}$ unevaluated, and $u_{\mathsf{H}} + u_{\mathsf{L}} = x - z\gamma$ exactly.

IEEE COMPUTER SOCIETY

`http://www.ens-lyon.fr/~sboldo/coq/FArgReduct.html`. The interested reader is referred to [3, 7] for more detail about Coq and machine proving.

Proof of Theorem 2.2: $(1+2^{-\ell})^{-1}b \le a \le (1+2^{-\ell})b$ implies that $a = 0 \Leftrightarrow b = 0$. Without loss of generality we may assume $a, b > 0$. Write

$$a = 2^{m_a} 1.a_1 \cdots a_{p+\ell-1}, \quad b = 2^{m_b} 1.b_1 \cdots b_{p+\ell-1},$$

where $a_i, b_i \in \{0, 1\}$. We claim that $|m_a - m_b| \le 1$; Otherwise if $m_a - m_b \ge 2$, then

$$\frac{a}{b} = 2^{m_a - m_b} \frac{1.a_1 \cdots a_{p+\ell-1}}{1.b_1 \cdots b_{p+\ell-1}} > 2^2 \frac{1}{2} = 2,$$

a contradiction; Similarly if $m_a - m_b \le -2$, we have $b/a > 2$, a contradiction as well. Also without loss of generality (scale by $2^{-\max\{m_a, m_b\}}$), we may assume $\max\{m_a, m_b\} = 0$, and thus $a - b$ takes this form

$$a - b = \pm d_0.d_1 d_2 \cdots d_{p+\ell}$$

and $d_{p+\ell} = 0$ if $m_a = m_b = 0$.

- If $b \le a$, then $b \le a \le (1+2^{-\ell})b$ which implies $0 \le a - b \le 2^{-\ell} b$. Now if also $m_a = m_b = 0$, then $d_0 = d_1 = \cdots = d_{\ell-1} = 0 = d_{p+\ell}$, and thus $a - b = 2^{-\ell} d_\ell.d_{\ell+1} \cdots d_{p+\ell-1}$, representable exactly by the default FPN system; On the other hand if $m_a = 0 > m_b = -1$, then $d_0 = d_1 = \cdots = d_\ell = 0$, and thus $a - b = 2^{-(\ell+1)} d_{\ell+1}.d_{\ell+2} \cdots d_{p+\ell}$, also exactly representable.
- If $b > a$, then $b > a \ge (1+2^{-\ell})^{-1} b$ which implies $0 < b - a \le 2^{-\ell} a$. The rest of the proof is the same as for the case $b \le a$.

This completes the proof. ∎

Equivalently, (2.1) can be restated as

$$\frac{1}{1+2^{-\ell}} \le \frac{a}{b} \le 1 + 2^{-\ell} \tag{2.2}$$

unless $a = b = 0$.

3 Main Result

We now present the conditions under which $x - z\gamma$ can be represented exactly by a FPN, and thus it can be computed by $\{x - z\gamma\}_{\mathsf{fma}}$ without error. As in Section 1, $\alpha \approx 1/C$ and $\gamma \approx C > 0$. For the rest of this paper set

$$\delta := \alpha\gamma - 1, \tag{3.1}$$

and suppose

the last q consecutive significant bits of γ are zeros. (3.2)

q is allowed to be zero in which case the last significant bit of γ is 1. Let z be as defined by (1.1) with the conditions on z and ς given there. Assume for the moment that $k \ne 0$ and thus $z \ne 0$. We have

$$\begin{aligned} \frac{x}{z\gamma} &= \frac{x\alpha}{z\alpha\gamma} \\ &= \frac{z+\varsigma}{z\alpha\gamma} \\ &= \left(1 + \frac{\varsigma}{z}\right) \frac{1}{1+\delta}. \end{aligned}$$

Noticing that $|\varsigma/z| \le 1/(2k)$, we get

$$\left(1 - \frac{1}{2k}\right) \frac{1}{1+\delta} \le \frac{x}{z\gamma} \le \left(1 + \frac{1}{2k}\right) \frac{1}{1+\delta}. \tag{3.3}$$

Theorem 3.1 *$x - z\gamma$ is a FPN if the following conditions are met:*

$$-1/4 \le \delta \le 1/2, \tag{3.4}$$

$$\gamma \le \mathsf{round_up}(1/\alpha), \tag{3.5}$$

$$2^{\lfloor \log_2 |k| \rfloor + 1} \le \begin{cases} \frac{(2^q-1)+(2+2^q)\delta+\sqrt{\Delta_-}}{-2\delta}, & \text{if } \delta < 0, \\ \frac{2^q-1-2\delta+\sqrt{\Delta_+}}{2\delta}, & \text{if } \delta > 0. \end{cases} \tag{3.6}$$

where Δ_- and Δ_+ are defined by

$$\Delta_- := (2^q-2)^2\delta^2 + 2[(2^q)^2 - 3\cdot 2^q - 2]\delta + (2^q-1)^2, \tag{3.7}$$

$$\Delta_+ := 4\delta^2 + 4\delta + (2^q-1)^2. \tag{3.8}$$

One implication of this theorem is that the `fma` makes explicitly storing the extra bits in $z\gamma$ and then subtracting it carefully from x unnecessary.

Remark 3.1 Conditions (3.4) — (3.6) essentially restrict the selection of α and γ, as approximations to $1/C$ and C. It is easy for (3.4) to hold, and the range of feasible k is tied up with δ. Therefore the major hurdle is to satisfy (3.5), while making $\alpha \approx 1/C$ and $\gamma \approx C$ as accurately as possible. Section 4 will presents a detailed analysis in this regard.

Remark 3.2 Notice that for $k > 0$

$$k = 2^{\log_2 k} \le 2^{\lfloor \log_2 k \rfloor + 1} \le 2^{\log_2 k + 1} = 2k.$$

Thus (3.6) holds if

$$|k| \le \begin{cases} \frac{(2^q-1)+(2+2^q)\delta+\sqrt{\Delta_-}}{-4\delta}, & \text{if } \delta < 0, \\ \frac{2^q-1-2\delta+\sqrt{\Delta_+}}{4\delta}, & \text{if } \delta > 0. \end{cases} \tag{3.9}$$

(3.6) and (3.9) leave a bound on $|k|$ undefined if $\delta = 0$ for which case, there is no constraint on k. The case $\delta = 0$ happens only when both α and γ are powers of two, a case that is not very interesting for elementary function computations.

IEEE COMPUTER SOCIETY

Remark 3.3 Let us examine asymptotically in δ how big the bound by (3.9) can be because δ is very tiny in the interesting cases. To do so, what we essentially need is to expand $\sqrt{\Delta_-}$ and $\sqrt{\Delta_+}$ at $\delta = 0$. Both Δ_- and Δ_+ are quadratic in δ with the constant terms vanish at $q = 0$. Therefore the expansions should be done depending on whether $q = 0$ or not. We have

- When $q = 0$,

$$|k| \le \begin{cases} \frac{1}{\sqrt{-2\delta}} - \frac{3}{4} + \mathcal{O}(\sqrt{-\delta}), & \text{if } \delta < 0, \\ \frac{1}{2\sqrt{\delta}} - \frac{1}{2} + \mathcal{O}(\sqrt{\delta}), & \text{if } \delta > 0. \end{cases} \tag{3.10}$$

- When $q \ge 1$,

$$|k| \le \begin{cases} \frac{2^q-1}{-2\delta} - \frac{[2^q]^2-2^q-2}{2(2^q-1)} + \mathcal{O}(\delta), & \text{if } \delta < 0, \\ \frac{2^q-1}{2\delta} - \frac{2^q-2}{2(2^q-1)} + \mathcal{O}(\delta), & \text{if } \delta > 0. \end{cases} \tag{3.11}$$

If $\delta = \mathcal{O}(\epsilon_{\mathsf{m}}) = \mathcal{O}(2^{-p})$, these bounds say that $|k|$ can grow as big as of $\mathcal{O}(2^{p/2})$ if $q = 0$, and of $\mathcal{O}(2^p)$ if $q \ge 1$. Later in Algorithm 5.1 and Remark 5.1 of Section 5, we shall show that with a slight modification to the last significant bit of γ such that $\gamma \approx C$ with error less than 1.5 ulp, q can be made $q \ge 1$.

Proof of Theorem 3.1: No proof is needed if $k = 0$ and thus $z = 0$. Assume that $|k| \ge 1$. Then x and k have the same sign, and

$$|k|2^{-N} - 2^{-N-1} \le |x\alpha| = |k2^{-N} + \varsigma| \le |k|2^{-N} + 2^{-N-1}.$$

From now on we consider the case $x > 0$ only, and the other case $x < 0$ can be handled in a similar way.

Suppose $k = 1$. Then $z\gamma$ is a FPN, and

$$2^{-N-1} \le x\alpha \le 3 \cdot 2^{-N-1}$$

which implies

$$2^{-N-1}/\alpha \le x \le 3 \cdot 2^{-N-1}/\alpha.$$

Let $x_{\min}$ be the smallest FPN such that $x_{\min} \ge 2^{-N-1}/\alpha$, i.e.,

$$x_{\min} = 2^{-N-1}\mathsf{round_up}(1/\alpha).$$

Thus $\gamma \le 2^{N+1}x_{\min}$ by (3.5). Now $z = 2^{-N}$, and thus

$$\frac{x}{z\gamma} \ge \frac{x_{\min}}{z\gamma} = \frac{x_{\min}}{2^{-N}\gamma} \ge \frac{1}{2}.$$

On the other hand,

$$\frac{x}{z\gamma} \le \frac{3 \cdot 2^{-N-1}/\alpha}{2^{-N}\gamma} = \frac{3}{2}\frac{1}{1+\delta} \le 2,$$

if $\delta \ge -1/4$ which holds by (3.4). Thus $x - z\gamma$ is a FPN by Theorem 2.1.

Suppose $k \ge 2$. Inequality (3.3) yields

$$\frac{3}{4}\frac{1}{1+\delta} \le \frac{x}{z\gamma} \le \frac{5}{4}\frac{1}{1+\delta}.$$

Therefore

$$\frac{1}{2} \le \frac{x}{z\gamma} \le 2 \quad \text{if } -3/8 \le \delta \le 1/2 \tag{3.12}$$

which is guaranteed by (3.4). Let

$$\ell := \lfloor \log_2 k \rfloor + 1$$

which is the number of bits to store k exactly. Then

$$2^{\ell-1} \le k < 2^\ell - 1.$$

Now if $k = 2^{\ell-1}$, then $z\gamma$ is a FPN. So $x - z\gamma$ is a FPN by (3.12) and Theorem 2.1. Notice that $z\gamma$ is a FPN with at most $p - q + \ell$ significant bits. If $q \ge \ell$, $z\gamma$ is also a FPN (in the default format). By (3.12) and Theorem 2.1, $x - z\gamma$ is a FPN. Assume now that $q < \ell$ and $k \ge 2^{\ell-1} + 1$. Then (3.3) implies

$$\left(1 - \frac{1}{2^\ell + 2}\right)\frac{1}{1+\delta} \le \frac{x}{z\gamma} \le \left(1 + \frac{1}{2^\ell + 2}\right)\frac{1}{1+\delta}.$$

We claim that under the condition of the theorem

$$\left(1 + \frac{1}{2^\ell + 2}\right)\frac{1}{1+\delta} \le 1 + \frac{2^q}{2^\ell}, \tag{3.13}$$

$$\left(1 - \frac{1}{2^\ell + 2}\right)\frac{1}{1+\delta} \ge \left(1 + \frac{2^q}{2^\ell}\right)^{-1}. \tag{3.14}$$

Therefore we have

$$\left(1 + 2^{-\ell+q}\right)^{-1}(z\gamma) \le x \le \left(1 + 2^{-\ell+q}\right)(z\gamma).$$

Since $z\gamma$ is a FPN with no more than $p - q + \ell$ bits in the significand, by Theorem 2.2, $x - z\gamma$ is a FPN (in the default format), as expected.

We have to prove (3.13) and (3.14).

Inequality (3.13) is equivalent to

$$-\left[2^\ell\right]^2\delta - [(2^q - 1) + (2 + 2^q)\delta]2^\ell - 2 \cdot 2^q(1+\delta) \le 0. \tag{3.15}$$

This inequality holds if $\delta \ge 0$. Assume $\delta < 0$, and notice that Δ_- is the resultant of the quadratic polynomial in 2^ℓ on the left-hand side of (3.15)

$$\Delta_- = [(2^q - 1) + (2 + 2^q)\delta]^2 - 4 \times \delta \times 2 \cdot 2^q(1+\delta).$$

$\Delta_- \ge 0$ for all $q \ge 0$ and all $\delta < 0$. This can be checked directly for $q = 0$ and $q = 1$ for which Δ_- is $\delta^2 - 8\delta$ and

IEEE COMPUTER SOCIETY

$1 - 8\delta$ respectively. For $q \geq 2$, Δ_- as a polynomial in δ never vanishes because its resultant

$$4[(2^q)^2 - 32^q - 2]^2 - 4 \times (2^q - 2)^2 \times (2^q - 1)^2 = 32 \cdot 2^q(3 - 2^q) < 0.$$

This combining with the fact that $\Delta_- > 0$ at $\delta = 0$ implies that $\Delta_- > 0$ always. It can now be seen that (3.15) is guaranteed by (3.6).

Inequality (3.14) is equivalent to

$$-\left[2^\ell\right]^2 \delta + [(2^q - 1) - 2\delta]2^\ell + 2^q \geq 0. \tag{3.16}$$

This inequality holds if $\delta \leq 0$. Assume $\delta > 0$, and notice that Δ_+ is the resultant of the quadratic polynomial in 2^ℓ on the left-hand side of (3.16)

$$\Delta_+ = [(2^q - 1) - 2\delta]^2 - 4 \times (-\delta) \times 2^q.$$

$\Delta_+ \geq 0$ for all $q \geq 0$ and all $\delta > 0$. It can now be seen that (3.16) is guaranteed by (3.6).

The proof is now completed. ∎

4 Analysis of Constraints Between α and γ

In using Theorem 3.1 to come up with α and γ for argument reductions, we essentially need to consider making α and γ to satisfy (3.5) and, if necessary, forcing the last bit of γ to be 0 because for modest p, $|\delta|$ is easily made to be much less than $1/4$ and because the constraints on k are results of the two. In fact, it is easy to make δ as tiny as ϵ_{m}. For functions like exponentials, k cannot be much bigger before overflow or underflow takes over and thus the range imposed on k by Theorem 3.1 is sufficient, even for $q = 0$; While for others, the range imposed on k by Theorem 3.1 for $q \geq 1$ is also sufficient for reasons as follows. It is conceivable that (1.1) simulates extracting in exact arithmetic the certain number of leading significant bits of x/C, while (1.2) simulates $x - zC$. However $x\alpha$ if represented exactly has up to $2p$ significant bits with only about p leading bits trustworthy as an approximation to x/C because in general $\alpha \approx 1/C$ with relative error about ϵ_{m}. Therefore in order for (1.1) and (1.2) to mimic what they are intended, the number of extracted bits in z in (1.1) should be made no bigger than p, or equivalently $|k| \leq 2^{p-1} = 2^{-1}\epsilon_{\mathsf{m}}^{-1}$.

In what follows, we shall concentrate on how to make

$$\alpha \approx 1/C \text{ and } \gamma \approx C$$

as accurate as possible while not violating the assumptions of Theorem3.1. Naturally best possible α and γ are

$$\alpha = \mathsf{round_to_nearest}(1/C) \text{ and } \gamma = \mathsf{round_to_nearest}(C), \tag{4.1}$$

but that cannot always be done as will be shown by Example 4.1 below. Our best choice is to make γ approximates its target as accurately as possible while α approximates $1/\gamma$ in the best way, i.e.,

$$\alpha = 1 \oslash \gamma. \tag{4.2}$$

Lemma 4.1 *Let $a > 0$ be a FPN. Then*

$$\mathsf{round_up}\left(\frac{1}{1 \oslash a}\right) \geq a.$$

Proof: Without loss of generality, we scale a such that $1 \leq a < 2$. Write $1 \oslash a = 1/a + \eta$ with $|\eta| \leq 2^{-p-1}$. No proof is needed if $a = 1$. For $1 < a < 2$, it suffices to show that

$$\frac{1}{1 \oslash a} > a - 2^{-p+1}. \tag{4.3}$$

If $\eta \leq 0$, then $1 \oslash a \leq 1/a$, and thus (4.3) holds. Assume $\eta > 0$. We have

$$\frac{1}{1 \oslash a} = \frac{a}{1 + a\eta} = a[1 - (a\eta) + (a\eta)^2 - \cdots] = a - a[(a\eta) - (a\eta)^2 + \cdots].$$

Notice that $(a\eta) - (a\eta)^2 + \cdots$ is an alternating series and thus it is bounded strictly by its first term, i.e.,

$$0 < a[(a\eta) - (a\eta)^2 + \cdots] < a(a\eta) < 4 \times 2^{-p-1} = 2^{-p+1}$$

which yields (4.3). ∎

A related result but only for those a which can be scaled by a power of two to fall between 1 and $\sqrt{2}$

$$1 \oslash (1 \oslash a) = a$$

is due to W. Kahan [9, Exercise 27 of §4.22 and its solution].

Theorem 4.1 (3.5) *holds if $\alpha\gamma \leq 1$ which is true if*

$$\textit{either } \alpha = \mathsf{round_down}(1/\gamma) \textit{ or } \gamma = \mathsf{round_down}(1/\alpha). \tag{4.4}$$

Proof: $\alpha\gamma \leq 1$ implies $\gamma \leq 1/\alpha$ and thus (3.5). Note that (4.4) implies $\alpha \leq (1/\gamma)(1 - \epsilon)$ for some $0 \leq \epsilon \leq 2\epsilon_{\mathsf{m}}$, and thus $\alpha\gamma \leq 1$. ∎

Theorem 4.2 (3.5) *holds if* (4.1) *such that either $\alpha \leq 1/C$ or $\gamma \leq C$ or C can be scaled by a power of two to fall in $[1, \sqrt{2})$.*

The restriction that the scaled C to fall in $[1, \sqrt{2})$ is quite unpleasant but necessary as the following example shows. It is invoked in the later proof when both $\alpha \leq 1/C$ and $\gamma \leq C$ are violated.

IEEE COMPUTER SOCIETY

Example 4.1 $C = 2\pi$ for which it can be verified that with (4.1), (3.4) and (3.5) hold for $3 \le p \le 197$ but (3.5) fails for $p = 198$.

Proof of Theorem 4.2: Without loss of generality, we may scale C by a power of 2 such that $1 \le C < 2$. No proof is needed if $C = 1$. Assume $1 < C < 2$. Then

$$\alpha = 1/C + \delta_1, \quad \gamma = C + \delta_2$$

and $|\delta_1| \le 2^{-p-1}$, $|\delta_2| \le 2^{-p}$. First if $\delta_1 \le 0$, then

$$\alpha \le 1/C \Rightarrow C \le 1/\alpha,$$

and thus

$$\begin{aligned} \gamma &= \mathsf{round_to_nearest}(C) \\ &\le \mathsf{round_up}(C) \\ &\le \mathsf{round_up}(1/\alpha), \end{aligned}$$

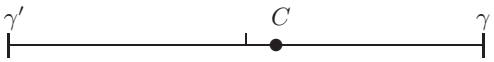
as expected. Next if $\delta_2 \le 0$, then

$$\begin{aligned} \gamma \le C &\Rightarrow 1/C \le 1/\gamma \\ &\Rightarrow \alpha \le 1 \oslash \gamma \\ &\Rightarrow \frac{1}{\alpha} \ge \frac{1}{1 \oslash \gamma}, \end{aligned}$$

and thus by Lemma 4.1

$$\mathsf{round_up}\left(\frac{1}{\alpha}\right) \ge \mathsf{round_up}\left(\frac{1}{1 \oslash \gamma}\right) \ge \gamma.$$

It is remained to prove the claim for the case $\delta_1 > 0$ and $\delta_2 > 0$. This is the situation where we need $C < \sqrt{2}$. Notice that $\gamma = C + \delta_2 > C > 1$. Let γ' be the biggest FPN that is smaller than γ, i.e., $\gamma' = \gamma - 2^{-p+1}$. $\delta_2 > 0$ implies that C is above or at the middle point between γ' and γ as show below.



It suffices to prove $1/\alpha > \gamma'$ for (3.5) to hold. Notice

$$\begin{aligned} \frac{1}{\alpha} = \frac{C}{1 + C\delta_1} &= C[1 - (C\delta_1) + (C\delta_1)^2 - \cdots] \\ &= C - C[(C\delta_1) - (C\delta_1)^2 + \cdots]. \end{aligned}$$

$C < \sqrt{2}$ implies

$$0 < C[(C\delta_1) - (C\delta_1)^2 + \cdots] < C^2\delta_1 < 2 \times 2^{-p-1} = 2^{-p}$$

and consequently

$$\frac{1}{\alpha} > C - 2^{-p} \ge \gamma',$$

as expected. ∎

Theorem 4.3 (3.5) *holds if* (4.2).

Proof: It is a consequence of Theorem 4.2 by taking $C = \gamma$, and thus $\gamma \le C$ holds, and α and γ are defined as in (4.1). ∎

5 An Algorithm for α and γ

Thanks to the results of Sections 3 and 4, we suggest the following algorithm for picking $\alpha \approx 1/C$ and $\gamma \approx C$ for all p that is not too small, say $p \ge 3$. (This is to make (3.4) always satisfied.)

Algorithm 5.1 *Given C, the following steps produce α and γ such that $x - z\gamma$ is a FPN.*

1. *Compute α and γ as in* (4.1)*;*
2. *Verify* (3.5)*. This is automatically satisfied if we know beforehand that C can be scaled by a power of two to fall in $[1, \sqrt{2})$. Otherwise either verify directly* (3.5)*, or check if one of the following conditions is satisfied:*

   $$\alpha\gamma \le 1, \text{ or } \alpha \le 1/C, \text{ or } \gamma \le C$$

   according to Theorems 4.1 and 4.2.
3. *Compute an upper bound on all applicable k as given by (3.9). If, however, the bound by (3.9) is too small for the computation of the elementary function in question (This may happen when $q = 0$), we can either add 1 ulp to or subtract 1 ulp from γ, and then take*

   $$\alpha = 1 \oslash \gamma$$

   as in (4.2). Doing so makes (3.5) *automatically satisfied by Theorem 4.3.*

Remark 5.1 In the 3rd step of Algorithm 5.1, adding 1 ulp or subtracting 1 ulp, if done carefully, can make $q \ge 2$. In fact, this can be accomplished by adding 1 ulp if the last two bits of γ are 11_2 or subtracting 1 ulp if the last two bits of γ are 01_2.

Next we shall present two examples: $C = \ln 2$ from the computation of $\exp(x)$, and $C = 2\pi$ from the computation of radian trigonometric functions. When it comes to write a library of the elementary mathematical functions, we often use the floating point arithmetic of the highest precision available on any given hardware. This means to use the IEEE double precision arithmetic on the existing RISC machines, and Intel double-extended precision arithmetic (64 bits in the significand) on machines equipped with Intel processors. Therefore the parameters α and γ are either of IEEE double precision or of Intel double-extended precision. In what follows, however, we do give IEEE single precision α and γ just to show the applicability of our theorems and algorithm.

Example 5.1 Consider the computation of $\exp(x)$ based on [11]

$$\exp(x) = 2^{x \log_2 e} = 2^{x/\ln 2},$$

IEEE COMPUTER SOCIETY

where $e = \exp(1)$ the natural number. Here $C = \ln 2$. Because $2 \ln 2 \approx 1.386$, C can be scaled by a power of two to fall in $[1, \sqrt{2})$. Thus (3.5) holds with (4.1). For $\exp(x)$, $k2^{-N}$ is extracted to serve two purposes: the exponent M of $\exp(x)$ and a table lookup index m

$$k2^{-N} = M + m2^{-N}$$

where M and m are integers, where $0 \le m \le 2^N - 1$. Then

$$\exp(x) \approx 2^M \times 2^{m2^{-N}} \times \exp(t),$$

where $t = x - z\gamma$ (or $t \approx x - z \ln 2$ if some γ_l is used so that unevaluated $\gamma + \gamma_l \approx \ln 2$ to about $2p$ bits). Typically $2^{m2^{-N}}$ is obtained through table lookup. In order not to use too big a table, N may be chosen, say, no bigger than 10. On the other hand, 2^M quickly overflows or underflows as $|M|$ gets bigger, e.g., for IEEE double precision, it overflows if $M \ge 1023$ and underflows to zero if $M < -1022 - 53$. For this reason, interesting $|k|$ is no bigger than about 2^{11+N}. Therefore the constraints imposed on k by Theorem 3.1 for $q = 0$ or $q \ge 1$ are acceptable. In what follows, a string with the subscript 16 denotes a hexadecimal number, and that without the subscript is a decimal number of the usual radix 10.

a) IEEE single precision:

$$\begin{aligned} \alpha &= 2^{-3} \times B.8AA3B_{16}, \\ \gamma &= 2^{-4} \times B.17218_{16}, \\ \delta &\approx -1.06 \times 10^{-08}, \\ |k| &\le 13AD5D94_{16}. \end{aligned}$$

Here $q = 3$.

b) IEEE double precision:

$$\begin{aligned} \alpha &= 2^{0} \times 1.71547652B82FE_{16}, \\ \gamma &= 2^{-1} \times 1.62E42FEFA39EF_{16}, \\ \delta &\approx -4.76 \times 10^{-17}, \\ |k| &\le 61C6EC2_{16}. \end{aligned}$$

Here $q = 0$. Thus the largest possible $|k|$ is about $\epsilon_m^{-1/2}$. If we add 1 ulp to γ and take $\alpha = 1 \oslash \gamma$ as suggested by Step 3) of Algorithm 5.1, it will make $q = 4$. Doing so, we get

$$\begin{aligned} \alpha &= 2^{0} \times 1.71547652B82FD_{16}, \\ \gamma &= 2^{-1} \times 1.62E42FEFA39F0_{16}, \\ \delta &\approx -4.13 \times 10^{-17}, \\ |k| &\le 2851984E2E90048_{16}. \end{aligned}$$

c) Intel double-extended precision: (64 bits in the significand)

$$\begin{aligned} \alpha &= 2^{-3} \times B.8AA3B295C17F0BC_{16}, \\ \gamma &= 2^{-4} \times B.17217F7D1CF79AC_{16}, \\ \delta &\approx 3.57 \times 10^{-20}, \\ |k| &\le 2464972759AF9B334_{16}. \end{aligned}$$

Example 5.2 Consider the computation of $\sin(x)$ and $\cos(x)$. They are two of the most difficult elementary functions to compute. Here $C = 2\pi$ which is their period. Therefore only the fractional part of $x/(2\pi)$ is interesting. But getting enough correct fractional bits of $x/(2\pi)$ can be tricky and costly for x of huge magnitude or extremely close to an integral multiple of $\pi/2$. Presumably arguments x as such are rare in any given application. In order not to slow down the speed for most common arguments, often a fast reduction just as outlined at the beginning of this paper is performed and then some quick checking is done to see if a more careful reduction procedure is needed and if that is the case, the code is branched to perform a careful argument reduction which is rare and slow. The interested reader is referred to [8, 11, 12, 13] and references therein for more detail. In this paper, however, we are only interested in speeding up the fast reduction part.

a) IEEE single precision:

$$\begin{aligned} \alpha &= 2^{-6} \times A.2F983_{16}, \\ \gamma &= 2^{-1} \times C.90FDB_{16}, \\ \delta &\approx -1.25 \times 10^{-08}, \\ |k| &\le 18B0_{16}. \end{aligned}$$

Here $q = 0$. Subtracting 1 ulp from γ and taking $\alpha = 1 \oslash \gamma$ yield $q = 2$ and

$$\begin{aligned} \alpha &= 2^{-6} \times A.2F982_{16}, \\ \gamma &= 2^{-1} \times C.90FDC_{16}, \\ \delta &\approx -3.03 \times 10^{-08}, \\ |k| &\le 2F4A062_{16}. \end{aligned}$$

b) IEEE double precision:

$$\begin{aligned} \alpha &= 2^{-3} \times 1.45F306DC9C883_{16}, \\ \gamma &= 2^{2} \times 1.921FB54442D18_{16}, \\ \delta &\approx 2.28 \times 10^{-17}, \\ |k| &\le 22066D471BD6D2D_{16}. \end{aligned}$$

Here $q = 3$.

c) Intel double-extended precision:

$$\begin{aligned} \alpha &= 2^{-6} \times A.2F9836E4E44152A_{16}, \\ \gamma &= 2^{-1} \times C.90FDAA22168C235_{16}, \\ \delta &\approx 1.72 \times 10^{-20}, \\ |k| &\le E2ED4431_{16}. \end{aligned}$$

Here $q = 0$. Subtracting 1 ulp from γ and taking $\alpha = 1 \oslash \gamma$ yield $q = 2$ and

$$\begin{aligned} \alpha &= 2^{-6} \times A.2F9836E4E44152B_{16}, \\ \gamma &= 2^{-1} \times C.90FDAA22168C234_{16}, \\ \delta &\approx 3.34 \times 10^{-20}, \\ |k| &\le 26FA94EFA25DF2177_{16}. \end{aligned}$$

IEEE COMPUTER SOCIETY

6 Conclusions

We have presented theorems that prove the correctness and effectiveness of the commonly used argument reduction technique in elementary function computations, especially on machines that have hardware support for fused-multiply-add instructions. The conditions of these theorems are easily met as our analysis indicates. While we showed it is not always possible to use the best possible parameters as defined by (4.1) under all circumstances, an almost best possible selection as in (4.2) can be used at all times. On case-by-case basis, however, it is possible to use (4.1) by verifying individually the conditions in our main theorem in Section 3 while none of the theorems of Section 4 apply, e.g., $C = 2\pi$ and $3 \leq p \leq 197$. Based on our results in Sections 3 and 4, a 3-step algorithm is presented to derive argument reduction parameters α and γ.

While Theorem 3.1 as of now is sufficient in the sense that effective parameters for efficient argument reductions can be obtained without any difficulty, it would be interesting to know if some of the conditions of Theorem 3.1 are necessary, i.e., $x - z\gamma$ is not a FPN if one or more of the conditions fails. But we could not either prove it or find a counterexample at this point. We shall work on this in the future.

Finally we comment that the results in this paper are extensible to floating point number systems with radix other than 2 (see Remark 2.1). But we omit details here.

Acknowledgment

The authors are indebted to referees' constructive suggestions which improve the presentation considerably. One of the referees read the early draft so carefully that he compiled a long list of inappropriate English used, along with many valuable technical comments. They are especially grateful for his/her diligent readership.

Part of this work was done while the first author was on leave at Hewlett-Packard Company. He is grateful for help received from Jim Thomas, John Okada, and Peter Markstein of HP Itanium floating point and elementary math library team at Cupertino, California. Working with these people has been a pleasure.

References

[1] American National Standards Institute and Institute of Electrical and Electronic Engineers. IEEE standard for binary floating-point arithmetic. *ANSI/IEEE Standard, Std 754-1985,* New York, 1985.

[2] ANSI/ISO/IEC 9899:1999. *Programming Languages – C*. 1999.

[3] M. Daumas, L. Rideau, and L. Théry. A generic library of floating-point numbers and its application to exact computing. In *14th International Conference on Theorem Proving in Higher Order Logics*, pages 169–184, Edinburgh, Scotland, 2001.

[4] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–47, Mar. 1991.

[5] J. Harrison, T. Kubaska, S. Story, and P. T. P. Tang. The computation of transcendental functions on the IA-64 architecture. *Intel Technology Journal*, (Q4):1–7, November 1999.

[6] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadephia, 1996.

[7] G. Huet, G. Kahn, and C. Paulin-Mohring. The Coq proof assistant: a tutorial: version 7.2. Technical Report 256, Institut National de Recherche en Informatique et en Automatique, Le Chesnay, France, 2002. Available at `ftp://ftp.inria.fr/INRIA/publication/publi-pdf/RT/RT-0256.pdf`.

[8] W. Kahan. Minimizing q*m-n. At the beginning of the file "nearpi.c", available electronically at `http://http.cs.berkeley.edu/~wkahan/testpi`, 1983.

[9] D. Knuth. *The Art of Computer Programming*, volume 2. Addison Wesley, Reading, MA, 1981.

[10] R.-C. Li. Always Chebyshev interpolation in elementary function computations. submitted for publication, June 2001.

[11] P. Markstein. *IA-64 and Elementary Functions: Speed and Precision*. Prentice Hall, New Jersey, 2000.

[12] J.-M. Muller. *Elementary Functions: Algorithms and Implementation*. Birkhåuser, Boston•Basel•Berlin, 1997.

[13] K. C. Ng. Argument reduction for huge arguments: Good to the last bit. Technical report, SunPro, 1992. available electronically at `http://www.validgh.com/`.

[14] M. L. Overton. *Numerical Computing with IEEE Floating Point Arithmetic*. SIAM, Philadelphia, 2001.

[15] M. J. D. Powell. On the maximum errors of polynomial approximations defined by interpolation and by least squares criteria. *The Computer Journal*, 9(4):404 – 407, February 1967.

[16] P. H. Sterbenz. *Floating-Point Computation*. Prentice-Hall series in automatic computation. Prentice-Hall, Englewood Cliffs, NJ, USA, 1974.

[17] P. T. P. Tang. Table-driven implementation of the exponential function in IEEE floating-point arithmetic. *ACM Transactions on Mathematical Software*, 15(2):144–157, June 1989.

[18] P. T. P. Tang. Table-driven implementation of the logarithm function in IEEE floating-point arithmetic. *ACM Transactions on Mathematical Software*, 16(4):378–400, Dec. 1990.

[19] P. T. P. Tang. Table lookup algorithms for elementary functions and their error analysis. In P. Kornerup and D. W. Matula, editors, *Proceedings of the 10th IEEE Symposium on Computer Arithmetic*, pages 232–236, Grenoble, France, June 1991. IEEE Computer Society Press, Los Alamitos, CA.

[20] P. T. P. Tang. Table-driven implementation of the `expm1` function in IEEE floating-point arithmetic. *ACM Transactions on Mathematical Software*, 18(2):211–222, June 1992.

IEEE COMPUTER SOCIETY