IEEE-754 precision-p base-β arithmetic implemented in binary

SIEGFRIED M. RUMP, Institute for Reliable Computing, Hamburg University of Technology, and Visiting Professor at Waseda University, Faculty of Science and Engineering

We show how an IEEE-754 conformant precision-p base-β arithmetic can be implemented based on some binary floating-point and/or integer arithmetic. This includes the four basic operations and square root subject to the five IEEE-754 rounding modes, namely the nearest roundings with roundTiesToEven and roundTiesToAway, the directed roundings downwards and upwards, as well as rounding towards zero. Exceptional values like ∞ of NaN are covered according to the IEEE-754 arithmetic standard.

The results of the precision-p base-β operations are computed using some underlying precision-q binary arithmetic. We distinguish two cases. When using a precision-q binary integer arithmetic, the base-β precision p is limited for all operations by $\beta^{2p} \leq 2^q$, whereas using a precision-q binary floating-point arithmetic imposes stronger limits on the base-β precision, namely $\beta^{2p} \leq 2^q$ for addition and multiplication, $\beta^{2p} \leq 2^{q-1}$ for division and $\beta^{2p} \leq 2^{q-3}$ for the square root. Those limitations cannot be improved.

The algorithms are implemented in a Matlab/Octave flbeta-toolbox with the choice of using `uint64` or `binary64` as underlying arithmetic. The former allows larger precisions, the latter is advantageous for the square root, whereas computing times are similar. The flbeta-toolbox offers precision-p base-β scalar, vector and matrix operations including sparse matrices as well as corresponding interval operations. The base β can be chosen in the range $\beta \in [2, 64]$. The flbeta-toolbox will be part of Version 13 of INTLAB [Rump 1999], the Matlab/Octave toolbox for reliable computing.

Categories and Subject Descriptors: G.1 [**Numerical Algorithms**]: Numerical Algorithms

General Terms: Computer arithmetic, precision-p base-β IEEE-754 arithmetic, interval arithmetic, Matlab

Additional Key Words and Phrases: Floating-point arithmetic, precision-p, base-β, IEEE-754, double rounding, interval arithmetic, INTLAB

1. OVERVIEW

Suppose a binary integer or floating-point arithmetic following the IEEE-754 standard with precision q is available. This note aims to emulate an IEEE-754 precision-p base-β arithmetic for $\beta \geq 2$ with specifiable exponent range $(E_{\min}, E_{\max})$. The arithmetic covers the four basic operations and the square root including rounding to nearest in roundTiesToEven and roundTiesToAway, directed roundings, rounding towards zero, exceptions and gradual underflow, i.e., subnormal numbers.

All results in the precision-p base-β arithmetic shall be IEEE-754 conformant including overflow, underflow and exceptional values such as ∞ and NaN.

Previous work includes flap [Stewart 2009], a Matlab toolbox for decimal arithmetic with rounding to nearest. According to Pete Stewart [Stewart 2014] he tested his tool-

Siegfried M. Rump, Institute for Reliable Computing, Hamburg University of Technology, Am Schwarzenberg-Campus 3, Hamburg 21073, Germany, and Waseda University, Faculty of Science and Engineering, 3–4–1 Okubo, Shinjuku-ku, Tokyo 169–8555, Japan; email `rump@tuhh.de` .

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

box thoroughly, however, without any claim of rigor. Indeed, at least for precision larger than 8 decimals, examples of incorrect rounding can be found[1].

There are several publications on the emulation of a lower precision binary arithmetic in binary with correct rounding, for example [Moler 2019; Higham and Pranesh 2019; Flegar et al. 2019; Fasi and Mikaitis 2020; Meurant 2020]. Much more powerful is the MPFR library [1] implementing a multiple precision binary arithmetic with correct rounding.

The emulation of a lower precision in the same base β with correct rounding according to IEEE-754 is discussed in [Rump 2017]. As an example for binary arithmetic, the fl-toolbox emulating precision-p in `binary64` (double precision) for $p \leq 26$ is implemented in INTLAB [Rump 1999], the Matlab/Octave toolbox for reliable computing.

A base-β arithmetic for $\beta \neq 2$ is not easily available. Therefore we describe in the present paper the emulation of a precision-p and general base-β arithmetic in binary arithmetic, for example in `binary64`. A main problem is that, for example, a correct result $x := \sqrt{a}$ is approximated by some α computed in binary arithmetic, but the precision-p base-β rounding of x and α need not to coincide. That effect [Muller et al. 2018] is called "double rounding", see Section 5. Double rounding cannot occur for $\beta = 2$ and large enough working precision, i.e., simulating a lower precision binary arithmetic in binary [Roux 2014; Rump 2017]. For odd base the situation is more involved, see Section 3.1.

We store precision-p base-β numbers as a pair (m, e) of `binary64` numbers comprising signed mantissa and exponent. For internal computations we use a q-bit binary arithmetic with the choice of `uint64` or `binary64`. Since the square of a mantissa should be effectively computable without error, that naturally limits the precision p by $\beta^{2p} \leq 2^q$. No matter which format is used for the underlying arithmetic, always $q \leq 64$ so that the mantissa of precision-p base-β numbers can be stored in 32 bits, i.e., in both internal formats without error. Only internal operations are performed in `binary64` or `uint64`, the pairs (m, e) are stored in `binary64`.

When using `binary64` for the underlying arithmetic, then rounding to nearest in roundTiesToEven is involved, and the possibility of double roundings reduces the maximal precision of base-β numbers by $\beta^{2p} \leq 2^q = 2^{53}$ for addition, subtraction and multiplication, by $\beta^{2p} \leq 2^{q-1} = 2^{52}$ for division, and by $\beta^{2p} \leq 2^{q-3} = 2^{50}$ for the square root. As we will see, those limits cannot be improved.

The underlying format `uint64` offers a larger precision for base-β numbers, namely $\beta^{2p} \leq 2^{64}$ suffices for all five operations. However, special care for the square root is necessary because it is not available in `uint64`. There is not much difference in computing time between `binary64` and `uint64` as underlying format, see Section 8.

All algorithms are implemented in the Matlab/Octave flbeta-toolbox with the choice of using `uint64` or `binary64` as underlying format. The former allows larger precisions, the latter is advantageous for the square root. The toolbox offers precision-p base-β scalar, vector and matrix operations including sparse matrices as well as a precision-p base-β interval arithmetic. The base β can be chosen in the range $\beta \in [2, 64]$, where the limit $\beta \leq 64$ is due to the limited number of characters for output. The flbeta-toolbox will be part of Version 13 of INTLAB [Rump 1999], the Matlab/Octave toolbox for reliable computing.

One of my motivations for writing this paper and the flbeta-toolbox is to have a versatile and easy-to-handle tool for testing hypotheses. For example, let $\mathrm{fl}(\cdot)$ be the rounding into some precision-p base-β arithmetic, and set $\tilde{t} := \mathrm{fl}(t)$ for $t \in \mathbb{R}$. Then

[1]For example, define $x = 10^{17} + d \cdot 10^9$ for $0 \leq d \leq 28448869$ and $e = 5 \cdot 10^8 - 1$. Then $x + e$ rounded to nearest in 9 decimal digits precision equals x, but flap yields $x + 10^9$, the successor of x. Note that all x and e are exactly representable in 9 decimal digits

the relative rounding error unit is $\mathbf{u} := \frac{1}{2}\beta^{1-p}$, and the error according to the first and second standard model is defined by $E_1(t) := |t-\tilde{t}|/|t|$ and $E_2(t) := |t-\tilde{t}|/|\tilde{t}|$, respectively [Higham 2002]. In [Jeannerod and Rump 2017] we proved that for a precision-p base-β number x the maximum errors of $t := \sqrt{x}$ according to the standard models are $E_1(t) = 1 - 1/\sqrt{1+2\mathbf{u}}$ and $E_2(t) = \sqrt{1+2\mathbf{u}} - 1$, respectively, that both bounds are sharp and are achieved if, and only if, $x = (1+2\mathbf{u})\beta^{2e}$ with $e \in \mathbb{Z}$. As an example, executable INTLAB-code to verify this for precision-6 base-5 arithmetic is

```
p = 6; beta = 5; u = beta^(1-p)/2; flbetainit(p,beta,100);
x = flbetasequence(1,beta^2); s = sqrt(double(x)); S = double(sqrt(x));
E1 = abs((S-s)./s); E2 = abs((S-s)./S);
[ max(E1) max(E2) ; 1-1/sqrt(1+2*u) sqrt(1+2*u)-1 ]
```

After initialization the vector `x` comprises all precision-p base-β numbers in the interval $[1, \beta^2]$, the next statements are standard Matlab code. The output is

```
ans =
1.0e-03 *
0.159961610237117   0.159987202047573
0.159961610237236   0.159987202047684
```

with results coinciding up to rounding errors of the `binary64` square roots. It is straightforward to compute the maximum error for different roundings.

The note is organized as follows. After introducing notations and definitions, the rationale of our approach and the arithmetical operations are presented in Section 3 together with proofs of correctness. In Sections 4 and 5 implementation details for the internal `uint64` and `binary64` computations are given, and the maximum precision p for an emulated base-β arithmetic is derived together with proof of optimality. Following a final normalization of the computed result and the conversion from `binary64` into precision-p base-β arithmetic is briefly discussed. The note finishes with some computational results and comparison with other packages.

2. NOTATION

Let $1 \le p \in \mathbb{N}$ and a pair $E := (E_{\min}, E_{\max})$ with $E_{\min}, E_{\max} \in \mathbb{Z}$ and $E_{\min} \le 0 \le E_{\max}$ be given. Denote by[2]

$$\mathfrak{F}_{p,\beta,E} := \{m\beta^e : \quad m, e \in \mathbb{Z},\ |m| < \beta^p,\ E_{\min} \le e \le E_{\max}\} \tag{1}$$

a set of precision-p base-β floating-point numbers, and set

$$\mathbb{F}_{p,\beta,E} := \mathfrak{F}_{p,\beta,E} \cup \{\pm\infty\} \quad \text{and} \quad \mathbb{F}^*_{p,\beta,E} := \mathbb{F}_{p,\beta,E} \cup \{\text{NaN}\}. \tag{2}$$

For example, $\mathfrak{F}_{p,\beta,E}$ with $E := (0,0)$ is the set of integers being less than β^p in absolute value.

We will use an internal representation according to Table I. The internal computations in the underlying q-bit arithmetic use `binary64` or unsigned integers `uint64`, where in the latter case we take care that in all computations the mantissa of intermediate results is nonnegative. Mantissas and exponents assume always integer values and are stored in binary64. That requires the technical assumption

$$2(E_{\max} - E_{\min} + p) < 2^{53}, \tag{3}$$

[2]For the sake of better exposition of the following results and proofs we choose an integer mantissa because directed rounding is natural to understand and rounding to nearest follows by the "0.5"-rule. It is equivalent to the often used "$m_1.m_2 \ldots m_p$" format by shifting the exponents by $p-1$.

Table I. Representation of precision-p base-β numbers $m\beta^e$.

Quantity	Condition on m	Condition on e
normalized number	$\beta^{p-1} \le \|m\| < \beta^p$	$E_{\min} \le e \le E_{\max}$
denormalized number	$1 \le \|m\| < \beta^{p-1}$	$e = E_{\min}$
realmax	$\beta^p - 1$	$e = E_{\max}$
realmin	β^{p-1}	$e = E_{\min}$
subrealmin	1	$e = E_{\min}$
± 0	± 0	
$\pm\infty$	$\pm\infty$	
NaN	NaN	

which means that exponents are limited by $E_{\max} - E_{\min} \lesssim 10^{15}$.

The exceptional cases $\pm 0, \pm\infty$ and NaN are represented by m according to Table I without condition on e, otherwise always $\beta^{p-1} \le |m| < \beta^p$ for normalized and $1 \le |m| < \beta^{p-1}$ for denormalized numbers.

The arithmetic operations shall follow the IEEE-754 floating-point arithmetic standard [IEEE 1987; 2008; 2019]. The set of nonzero floating-point numbers with $|m| < \beta^{p-1}$ and $e = E_{\min}$ represents the underflow range. Exceptional values like $\pm\infty$ and NaN are defined and treated as in IEEE-754. Note that always $\mathbb{F}_{p,\beta,E} = -\mathbb{F}_{p,\beta,E}$. For example, in the exceptional case $p = 1, \beta = 2$ the set $\mathbb{F}_{1,2,E}$ consists only of powers of 2, namely $\mathbb{F}_{1,2,E} = \{\pm 2^e : E_{\min} \le e \le E_{\max}\} \cup \{\pm 0, \pm\infty\}$, and there is no underflow range.

Very thorough and readable introductions to all aspects of floating-point arithmetic are [Muller et al. 2018] or [Brent and Zimmermann 2010].

The largest normalized, smallest normalized and smallest denormalized positive floating-point numbers realmax, realmin and subrealmin, respectively, in $\mathbb{F}$ are defined according to Table I. Hence

$$0 \ne f \in \mathfrak{F} \quad \text{normalized} \quad \Leftrightarrow \quad |f| \ge \text{realmin} \,.$$

We consider the five rounding functions

$$\mathrm{fl}_\varrho\colon [-R, R] \to \mathbb{F} \quad \text{with} \quad R := \text{realmax} \;\text{ and }\; \varrho \in \{\square_E, \square_A, \nabla, \Delta, \diamond\} \tag{4}$$

according to (5) (see [Brent and Zimmermann 2010, Section 3.1.9], [IEEE 2019]):

$$\begin{array}{lll}
\square_E & \text{to nearest} & |\mathrm{fl}_{\square_E}(x) - x| = \min\{|f - x|\colon\ f \in \mathbb{F}\}, \text{ rounding ties to even} \\
\square_A & \text{to nearest} & |\mathrm{fl}_{\square_A}(x) - x| = \min\{|f - x|\colon\ f \in \mathbb{F}\}, \text{ rounding ties to away} \\
\nabla & \text{downwards} & \mathrm{fl}_\nabla(x) := \max\{f \in \mathbb{F}\colon\ f \le x\} \\
\Delta & \text{upwards} & \mathrm{fl}_\Delta(x) := \min\{f \in \mathbb{F}\colon\ x \le f\} \\
\diamond & \text{towards zero} & \mathrm{fl}_\diamond(x) := \mathrm{sign}(x) \cdot \mathrm{fl}_\nabla(|x|).
\end{array} \tag{5}$$

For real numbers outside the representable range, i.e., $x \in \mathbb{R}\backslash[-R,R]$ roundings are defined by

$$\square_E(x) = \begin{cases} \text{sign}(x)\cdot\text{realmax} & \text{if } |x| < (\beta^p - \frac{1}{2})\beta^{E_{\max}} \\ \text{sign}(x)\cdot\text{realmax} & \text{if } |x| = (\beta^p - \frac{1}{2})\beta^{E_{\max}} \text{ and } \beta \text{ is odd} \\ \text{sign(x)}\cdot\infty & \text{otherwise} \end{cases}$$
$$\square_A(x) = \begin{cases} \text{sign}(x)\cdot\text{realmax} & \text{if } |x| < (\beta^p - \frac{1}{2})\beta^{E_{\max}} \\ \text{sign(x)}\cdot\infty & \text{otherwise.} \end{cases} \tag{6}$$

The definitions for $\varrho \in \{\nabla, \Delta\}$ are as in (5), and $\diamond(x) = \text{sign}(x)\cdot\text{realmax}$ for $x \in \mathbb{R}\backslash[-R,R]$, i.e., no operation may cause overflow in rounding towards zero.

The definition of roundTiesToEven for even base β is clear because the mantissa of exactly one of the neighbors of the midpoint of two adjacent precision-p base-β is even[3]. For odd base β, however, the precision-4 base-3 numbers 1112_3 and 1120_3 are adjacent numbers but both are even in base 3. We follow the suggestion in [Brent and Zimmermann 2010] to require that in $m\cdot\beta^e$ the whole significand m interpreted as an integer in the interval $[-\beta^p+1, \beta^p-1]$ should be even. In our example, the first number is 41 in decimal, the second 42 is even and wins.

Roundings respect the order, i.e., are monotonic in the sense that

$$x, y \in \mathbb{R},\ x \le y \quad \Rightarrow \quad \text{fl}_\varrho(x) \le \text{fl}_\varrho(y) \qquad \text{for} \quad \varrho \in \{\square_E, \square_A, \nabla, \Delta, \diamond\}. \tag{7}$$

We also use $\text{fl}_\varrho^{\mathbb{Z}}\colon \mathbb{R} \to \mathbb{Z}$, the rounding into $\mathbb{Z}$ according to the rounding mode ϱ. For $\varrho \in \{\nabla, \Delta, \diamond\}$ this corresponds in Matlab/Octave to `floor`, `ceil` and `fix`, for $\varrho = \square_A$ it corresponds to `round`. For $\varrho = \square_E$ there used to be no immediate Matlab statement, but from Version 2022a Matlab's function `round` allows several "tiebreaker" options including ties to even.

3. THE UNDERLYING ARITHMETIC

The precision-p base-β arithmetic will be emulated by some underlying machine arithmetic $\mathcal{A}$ covering the nonnegative integers in $[0, \beta^{2p})$, and allowing addition, subtraction, multiplication and division by 2.

For the remaining of this section we fix the precision p, the base β and the exponent range $[E_{\min}, E_{\max}]$ and write shortly $\mathfrak{F}$ and $\mathbb{F}$ for $\mathfrak{F}_{p,\beta,E}$ and $\mathbb{F}_{p,\beta,E}$, respectively.

3.1. Computing a proxy result using the underlying arithmetic $\mathcal{A}$

Let x denote the true real result of an operation. The problem of "double rounding" is avoided by computing some intermediate result x' such that the precision-p base-β rounded results of x and x' coincide. That is similar to ordinary division in IEEE-754 where the true result is replaced by some approximation with finite bit representation. If x' is computed using a precision-q base-β arithmetic, then for even base β double rounding cannot occur for large enough q, however, for odd base β double rounding may occur for any $q > p$ [Rump 2017]. Thus we need some method to represent and compute the intermediate result x'.

In our approach, the intermediate result x' is, up to scaling by a power of β, equal to A/B for computable integers A, B. In some sense that is a base-free representation. The scaling is chosen such that the correctly rounded integer division A/B is equal to

[3] An exception is $\mathbb{F}_{1,2,E}$ comprising only powers of 2, so that the mantissa of a nonzero number always equals 1. In that case we follow the "0.5"-rule so that, for example, 6 is rounded into 8, or 12 into 16.

the desired precision-p base-β rounded result (see Subsection 5.1). In the following we sketch that process.

For given $0 \neq x \in \mathbb{R}$ a pair (M', e') representing $\mathrm{fl}_\varrho(x)$ according to Table I is computed in three basic steps. We assume that special cases like $\pm\infty$, NaN and alike are treated separately. The first and main step is as follows.

First step.
Define a proxy $(\mu, e) := \mathcal{P}(x)$ such that $x' := \mu \cdot \beta^e$ satisfies $\mathrm{fl}_\varrho(x') = \mathrm{fl}_\varrho(x)$.

The proxy is not unique, only the property is important. More precisely, let $a, b \in \mathfrak{F}_{p,\beta,E}$ with $a, b \neq 0$ be given, i.e., $a = m_a\beta^{e_a}$ and $b = m_b\beta^{e_b}$ with (m_a, e_a) and (m_b, e_b) according to Table I. Then $A, B \in \mathcal{A}$ are computed such that $\mu := A/B$ has the desired property. Care is necessary that all operations for computing A and B are effectively computable in the underlying arithmetic in $\mathcal{A}$.

For addition, subtraction and multiplication, B is some power of β, so that μ is a scaled integer. For division and square root, both A and B are some integers such that $\mu := A/B$ has the desired property. In all cases μ is a real number satisfying $\mathrm{fl}_\varrho(\mu \cdot \beta^e) = \mathrm{fl}_\varrho(x)$.

Second step.
Calculate $M := \mathrm{fl}^{\mathbb{Z}}_\varrho(\mu)$ satisfying $\mathrm{fl}_\varrho(x) = M\beta^e$.

The mantissa M is correct for infinite exponent range, however, (M, e) may not satisfy the representation specified in Table I. Basically, $M = A/B$ where care is necessary that M is computed correctly according to the rounding mode ϱ. In particular, in `uint64` results larger than $\varrho := 2^{64} - 1$ are always rounded into ϱ.

Third step.
The pair (M, e) is normalized into $(M', e') := \mathcal{N}(M, e)$ as described in Section 6 with the property that $M'\beta^{e'}$ is the correctly rounded result $\mathrm{fl}_\varrho(x) = M'\beta^{e'} = M\beta^e$ and satisfies the representation according to Table I including exceptional cases.

After that final step we arrive at the desired representation.

A pair $(\mu, e) := \mathcal{P}(x)$ according to Table II is called a proxy of $0 \neq x \in \mathbb{R}$. The proxy is a relation, not necessarily a function, it states properties of $\mu \in \mathbb{R}, e \in \mathbb{Z}$. In the first two cases (μ, e) is the unique pair satisfying $x = \mu\beta^e$ and the representation stated in Table II. In the third case, any pair (μ, e) with the stated properties satisfies $|\mu|\beta^e \in I := \left(0, \frac{1}{2}\beta^{E_{\min}}\right)$ and $\mu x > 0$, so that $\mathrm{fl}_\varrho(x) = \mathrm{fl}_\varrho(\mu\beta^e)$ for all $|x| \in I$ and all roundings $\varrho \in \{\square_E, \square_A, \nabla, \Delta, \diamond\}$. That allows some freedom to choose any μ with $|\mu|$ in the open interval $(0, \frac{1}{2})$.

Table II. Proxy of $0 \neq x \in \mathbb{R}$ as a pair $\mathcal{P}(x) = (\mu, e)$.

condition on x	property of μ	property of e
$\beta^{p-1+E_{\min}} \leq \lvert x\rvert$	$\beta^{p-1} \leq \lvert\mu\rvert < \beta^p$	$E_{\min} \leq e$
$\frac{1}{2}\beta^{E_{\min}} \leq \lvert x\rvert < \beta^{p-1+E_{\min}}$	$\frac{1}{2} \leq \lvert\mu\rvert < \beta^{p-1}$	$e = E_{\min}$
$0 < \lvert x\rvert < \frac{1}{2}\beta^{E_{\min}}$	$\lvert\mu\rvert \in \left(0, \frac{1}{2}\right),\ \mu x > 0$	$e = E_{\min}$

For the remaining of Section 3 we assume $a = m_a\beta^{e_a}$ and $b = m_b\beta^{e_b}$ to be given s.t.

$$(m_a, e_a) \text{ and } (m_b, e_b) \quad \text{are according to Table I} \tag{8}$$

and satisfy

$$a, b \in \mathfrak{F}_{p,\beta,E} \quad \text{and} \quad a, b \neq 0. \tag{9}$$

Therefore, in particular $1 \le |m_a|, |m_b| < \beta^p$ and $E_{\min} \le e_a, e_b \le E_{\max}$. If a pair $a, b \in \mathbb{F}^*$ does not meet (9), the precision-p base-β result is computed by some case distinctions.

Let an operation $\circ \in \{+, -, \times, /, \sqrt{\cdot}\}$, a rounding $\varrho \in \{\square_E, \square_A, \nabla, \Delta, \diamond\}$ and a, b according to (8) and (9) be given. Next we show how to compute the integer triple (A, B, e) using the underlying arithmetic $\mathcal{A}$ such that $a \circ b = \mu \cdot \beta^e$ for $\mu := A/B$ and $\circ \in \{+, -, \times, /\}$, and $\sqrt{a} = \sqrt{\mu} \cdot \beta^e$ for $\mu := A$ and $\circ = \sqrt{\cdot}$. Moreover, the following properties will hold true with one exception for (16):

$$A, B, e \text{ are effectively computable using the given operations in } \mathcal{A}, \tag{10}$$

$$\text{all case distinctions are effectively executable,} \tag{11}$$

$$(\mu, e) \text{ satisfies the representation in Table II,} \tag{12}$$

$$\mathrm{fl}_\varrho(a \circ b) = \mathrm{fl}_\varrho(\mu \cdot \beta^e) \text{ for } \circ \in \{+, -, \times, /\}, \text{ and} \tag{13}$$

$$\mathrm{fl}_\varrho(\sqrt{a}) = \mathrm{fl}_\varrho(\mu \cdot \beta^e), \tag{14}$$

$$\text{the quantity } A \text{ is a positive integer in the range } [1, \beta^{2p}), \tag{15}$$

$$\text{the quantity } B \text{ is a positive integer in the range } [1, \beta^{2p-1}], \tag{16}$$

$$\text{the quantity e is an integer satisfying } |e| < 2^{53}. \tag{17}$$

By "effectively computable" we mean that only the required operations in $\mathcal{A}$ are used. The mentioned exception for (16) is base $\beta = 2$, underlying arithmetic `uint64`, precision $p = 32$ and one particular case of multiplication. In that, and only that case $B = \beta^{2p} = 2^{64}$ may happen which is not in `uint64`. However, in Section 4 we will show how to resolve that.

Concerning (13), we show in fact that $a \circ b = A/B \cdot \beta^e$ for $\circ \in \{+, -, \times, /\}$ except for results $a \circ b$ in the denormalized range.

The problem to compute $M = \mathrm{fl}^{\mathbb{Z}}_\varrho(\mu)$, where $\mu = A/B$ for $\circ \in \{+, -, \times, /\}$ and $\mu = \sqrt{A}$ for positive integers A, B in the interval $(0, \beta^{2p})$ depends on the underlying arithmetic used to compute (μ, e), and on the rounding mode. We discuss in Sections 4 and 5 two possibilities to do that, namely `uint64` and `binary64`, both with pros and cons. Note that in any case the mantissa is in the interval $(-\beta^p, \beta^p) \subseteq (-2^{32}, 2^{32})$ and may be stored in `binary64`. Thus, using `uint64` or `binary64` as the underlying arithmetic does not affect the storage scheme of a precision-p base-β number. Only for $(m_a, e_a) + (m_b, e_b)$ care is necessary when computing $C := \beta^{e_a - e_b} m_a + m_b$ because the type cast `uint64(`m_b`)` produces 0 for negative m_b. Then, set `B := uint64(abs(`m_b`))` and use $C = \beta^{e_a - e_b} m_a + B$ for positive b, and by $\beta^{e_a - e_b} m_a - B$ for negative b.

Recall that the exponent range $E_{\min}, E_{\max}$ is stored in binary64 numbers thus allowing for an exponent range limited by (3), i.e., $E_{\max} - E_{\min} \lesssim 10^{15}$.

3.2. Logarithm to base β

At certain places we need $e := \lfloor \log_\beta(M) \rfloor$ for an integer $M \in [1, \beta^{2p})$. In principle, e is effectively computable by storing the powers of β in an array and comparisons using a binary search. When using `uint64` as the integer precision-q binary arithmetic, another possibility is

```
betapowers = uint64(beta).^uint64(0:2*p-1);
[~,e] = max( fliplr(x(:) >= betapowers) , [] , 2 );
e = reshape( length(betapowers)-e , size(x) );     % 2p - e
```

which is working for arrays `x` as well. If there is a binary floating-point format with division according to IEEE-754 and with enough bits to store the integers in the inter-

val $[1, \beta^{2p})$, then it suffices to compute $\lfloor \log_\beta(M) \rfloor = \lfloor \log_2(M)/\log_2(\beta) \rfloor$ in floating-point using the function `log2`, the latter just extracting the mantissa bits. Then only one rounding error can occur, in the division, and the result is correct because β^e is a floating-point number and there cannot be another floating-point number between the true and the computed result, regardless of the rounding mode.

3.3. Addition and subtraction

Since $\mathfrak{F}_{p,\beta,E} = -\mathfrak{F}_{p,\beta,E}$, it suffices to treat addition. Let $x := a + b$ and note that $a, b \neq 0$ by assumption (9). After some case distinctions and using the notation (8) we may assume without loss of generality $a > 0$, $a \geq |b|$ and $x \neq 0$, so that $e_a \geq e_b$ and $x > 0$. To satisfy that, the operands a and b are possibly interchanged and the signs are changed. This has to be taken into account for directed roundings using $\nabla(-x) = -\Delta(x)$.

First, assume $e_a > e_b + p$. Then

$$|b| = |m_b|\beta^{e_b} \leq (\beta^p - 1)\beta^{e_b} \leq (\beta^p - 1)\beta^{e_a - p - 1} < \beta^{e_a - 1} \leq \frac{1}{2}\beta^{e_a}.$$

Hence, $\mathrm{fl}_{\square_E}(a + b) = \mathrm{fl}_{\square_A}(a + b) = a$. For the other rounding modes it follows that the result is either a itself or one of its neighbors, and that is easily computed based on some case distinctions on the sign of b and on the rounding mode. Note that $e_a > e_b + p$ implies that a is normalized.

Second, assume $e_a \leq e_b + p$ and define $C := \beta^{e_a - e_b} m_a + m_b \in \mathbb{N}$ so that $x = C\beta^{e_b}$. Using $e_a - e_b \geq 0$ and $x > 0$ it follows

$$1 \leq C \leq \beta^{e_a - e_b}(\beta^p - 1) + \beta^p - 1 < \beta^{2p} \tag{18}$$

so that C is a positive integer in the interval $[1, \beta^{2p})$ and is effectively computable. For $e_C := \lfloor \log_\beta(C) \rfloor$, which is also effectively computable, it follows $0 \leq e_C \leq 2p - 1$. Set $k := e_C - p + 1$ and define the triple (A, B, e) by

$$(A, B, e) := \begin{cases} \big(C\,,\, \beta^k\,,\, k + e_b\big) & \text{if } \; k \geq 0 \text{ and } k + e_b \geq E_{\min} \\ \big(C\beta^{-k}\,,\, 1\,,\, k + e_b\big) & \text{if } \; k < 0 \text{ and } k + e_b \geq E_{\min} \\ \big(C\beta^{e_b - E_{\min}}\,,\, 1\,,\, E_{\min}\big) & \text{if } \; k + e_b < E_{\min}. \end{cases} \tag{19}$$

In order to verify that the powers of β in (19) are effectively computable, we use $0 \leq e_C \leq 2p-1$ and $k = e_C - p + 1$ in the first two cases, and in the third case $0 \leq e_b - E_{\min} < -k = p - 1 - e_C \leq p - 1$, so that only powers β^ℓ with $0 \leq \ell \leq p$ are used.

A straightforward computation confirms $a + b = \mu \cdot \beta^e$ for $\mu := A/B$ in all three cases. In the first case $0 \leq k \leq p$, so that B satisfies (16) in all three cases of (19). In the first and second case

$$\beta^{p-1} = \beta^{e_C - k} \leq C\beta^{-k} < \beta^{e_C + 1 - k} = \beta^p, \tag{20}$$

and in the third case $e_C - p + e_b = k + e_b - 1 \leq E_{\min} - 2$ together with $e_b \geq E_{\min}$ and

$$1 \leq C\beta^{e_b - E_{\min}} \leq C\beta^{p - e_C - 2} < \beta^{e_C + 1}\beta^{p - e_C - 2} = \beta^{p-1}, \tag{21}$$

so that, together with (18), A satisfies (15) in all three cases of (19). In the first two cases (20) and in the third case (21) prove that (μ, e) satisfies the representation in Table II. Finally note that (3) verifies that all integer computations in (19) concerning the exponent e are effectively computable, and that (17) is satisfied.

3.4. Multiplication

By assumption (9) we may assume without loss of generality $a, b > 0$, so that $x := ab > 0$. In case of sign changes and directed rounding, $\nabla(-x) = -\Delta(x)$ may be used.

We use the notation (8) and denote $\mu' := m_a m_b$ and $e_c := e_a + e_b$. Hence $x = \mu' \beta^{e_c}$ and Table I implies $1 \le \mu' < \beta^{2p}$. Thus, according to Subsection 3.2, $e' := \lfloor \log_\beta(\mu') \rfloor$ is effectively computable and $0 \le e' \le 2p-1$. Abbreviate $k := e' - p + 1$ and define the triple (A, B, e) by

$$(A,B,e) := \begin{cases} \left(m_a m_b \,,\, \beta^k \,,\, k + e_c\right) & \text{if } k \ge 0,\ k + e_c \ge E_{\min} \\ \left(m_a m_b \beta^{-k} \,,\, 1 \,,\, k + e_c\right) & \text{if } k < 0,\ k + e_c \ge E_{\min} \\ \left(m_a m_b \,,\, \beta^{E_{\min} - e_c} \,, E_{\min}\right) & \text{if } k \ge 0,\ E_{\min} - p \le k + e_c < E_{\min} \\ \left(m_a m_b \beta^{-k} \,,\, \beta^{-k + E_{\min} - e_c} \,, E_{\min}\right) & \text{if } k < 0,\ E_{\min} - p \le k + e_c < E_{\min} \\ (1 \,,\, 4 \,, E_{\min}) & \text{if } k + e_c < E_{\min} - p. \end{cases} \tag{22}$$

Suppose $k < 0$. If a is normalized, then $\beta^{p-1} \le m_a$ and $m_b = \mu'/m_a < \beta^{e'+1-(p-1)} = \beta^{k+1} \le 1$ implies $m_b = 0$, a contradiction. Applying the same argument to b it follows that both a and b must be denormalized if $k < 0$, so that

$$k < 0 \quad \Rightarrow \quad m_a, m_b < \beta^{p-1},\ e_a = e_b = E_{\min} \ \text{ and } \ 1 \le \mu' < \beta^{2p-2}. \tag{23}$$

In the second and fourth case, $k < 0$ implies $A \in \mathbb{Z}$ and

$$1 \le m_a m_b \beta^{-k} < \beta^{e'+1+p-1-e'} = \beta^p.$$

Hence $1 \le m_a m_b < \beta^{2p}$ in all five cases of (22), so that A satisfies (15) in all five cases. In the first case $0 \le k = e' - p + 1 \le p.$ and in the third case

$$0 \le k < E_{\min} - e_c \le k + p = e' + 1 \le 2p. \tag{24}$$

If $E_{\min} - e_c < k + p$ or $e' < 2p - 1$, then $B = \beta^{E_{\min} - e_c} \le \beta^{2p-1}$ and (16) is satisfied. If $E_{\min} - e_c = k + p = e' + 1$ and $e' = 2p - 1$, then $k = p$ and $E_{\min} - e_c = 2p$. That is the exception to (16) mentioned in Section 3.1. In that case

$$1 \le m_a m_b = A \le (\beta^p - 1)^2 < \beta^{2p} - 1 \quad \text{and} \quad B = \beta^{2p}. \tag{25}$$

Note that 2^{64} is not in `uint64`, and in case $\beta^{2p} = 2^{64}$ the `uint64` computation of β^{2p} results in $2^{64} - 1$. That special case (25) will be addressed separately in Section 4.

In the fourth case

$$0 = -k + k < -k + E_{\min} - e_c \le p, \tag{26}$$

and therefore B satisfies (16) in all five cases but the exception (25). All quantities A, B, e are effectively computable, and obviously $x = A/B \cdot \beta^e$ in the first four cases. Define $\mu := A/B$ and recall $\beta^{e'} \le \mu' = m_a m_b < \beta^{e'+1}$. In the first two cases of (22),

$$\beta^{p-1} = \beta^{e' - (e' - p + 1)} = \beta^{e' - k} \le \mu < \beta^{e'+1-k} = \beta^p$$

shows that the pair (μ, e) satisfies the representation in Table II. Furthermore, $k + e_c - E_{\min} \le -1$ together with

$$0 < \mu = m_a m_b \beta^{e_c - E_{\min}} < \beta^{e'+1+e_c-E_{\min}} = \beta^{p+k+e_c-E_{\min}} \le \beta^{p-1}$$

proves this is also true in the third and fourth case. In the fifth case,

$$0 < x = m_a m_b \beta^{e_c} < \beta^{e'+1+e_c} = \beta^{k+e_c+p} \le \beta^{E_{\min}-1} \le \frac{1}{2}\beta^{E_{\min}}$$

and $e = E_{\min}$ imply $\mu = \beta^{-e} x \in (0, \frac{1}{2})$ and $\mu x > 0$. Thus, (μ, e) is according to the representation in Table II. By (3) all integer computations in (22) concerning the exponent e satisfy (17).

Concerning the powers of β in (22), we use $0 \le e' \le 2p-1$ to see $-p+1 \le k = e'-p+1 \le p$ in the first two cases, (24) in the third and (26) fourth case, so that subject to the exception (25) only powers β^ℓ with $0 \le \ell \le 2p-1$ are used.

3.5. Division

By assumption (9), we may assume without loss of generality $a, b > 0$, so that $x := ab > 0$. In case of sign changes and directed rounding, $\nabla(-x) = -\Delta(x)$ may be used.

We use the notation (8) and define $\mu' := m_a/m_b$, and we set $e_a := \lfloor \log_\beta(m_a) \rfloor$ and $e_b := \lfloor \log_\beta(m_b) \rfloor$. Then

$$\beta^{e_a-e_b-1} < \mu' < \beta^{e_a-e_b+1} \tag{27}$$

and therefore $\lfloor \log_\beta(\mu') \rfloor \in e_a - e_b + \alpha, \alpha \in \{-1, 0\}$. Note that

$$\lfloor \log_\beta(\mu') \rfloor = e_a - e_b \;\Leftrightarrow\; m_a/m_b \ge \beta^{e_a-e_b} \;\Leftrightarrow\; m_a \ge \beta^{e_a-e_b} m_b \;\Leftrightarrow\; \beta^{e_b-e_a} m_a \ge m_b$$

and define

$$e' := \begin{cases} e_a - e_b & \text{if } e_a - e_b \ge 0 \text{ and } m_a \ge \beta^{e_a-e_b} m_b \\ e_a - e_b & \text{if } e_a - e_b < 0 \text{ and } \beta^{e_b-e_a} m_a \ge m_b \\ e_a - e_b - 1 & \text{otherwise.} \end{cases} \tag{28}$$

Now $|e_a - e_b| \le p-1$ implies that both products in the first two cases are integers in $[1, \beta^{2p-1})$, so that $e' = \lfloor \log_\beta(\mu') \rfloor$ is effectively computable. Moreover, $0 \le e_a, e_b \le p-1$ yields

$$-p \le e' \le p-1. \tag{29}$$

Denote $e_c := e_a - e_b$ so that $x = a/b = \mu' \beta^{e_c}$. Abbreviate $k := e' - p + 1$ and define

$$(A, B, e) := \begin{cases} \left(\beta^{-k} m_a \,,\, m_b \,,\, k + e_c\right) & \text{if } k + e_c \ge E_{\min} \\ \left(\beta^{-k} m_a \,,\, \beta^{-k-e_c+E_{\min}} m_b \,,\, E_{\min}\right) & \text{if } E_{\min} - p \le k + e_c < E_{\min}. \\ (1 \,,\, 4 \,,\, E_{\min}) & \text{if } k + e_c < E_{\min} - p. \end{cases} \tag{30}$$

Then (29) gives

$$0 \le -k = p - 1 - e' \le 2p - 1, \tag{31}$$

thus $\beta^{-k} \in \mathbb{Z} \cap [1, \beta^{2p-1}]$ and

$$1 \le \beta^{-k} m_a = \beta^{p-1-e'} \frac{m_a}{m_b} m_b < \beta^{p-1-e'+e'+1+p} = \beta^{2p}$$

show that A satisfies (15) in all cases of (30). In the second case of (30),

$$E_{\min} - p \le k + e_c \le E_{\min} - 1 \quad \Rightarrow \quad 1 \le -k - e_c + E_{\min} \le p, \tag{32}$$

and that proves B satisfies (16) in all cases.

Defining $\mu := A/B$, a straightforward computation yields $x := a/b = \mu \cdot \beta^e$ in the first two cases of (30). All quantities A, B, e are effectively computable. Next we show that (μ, e) with $\mu := A/B$ satisfies the specification in Table II. In the first case, $\beta^{e'} \le \mu' = m_a/m_b = \beta^k \mu < \beta^{e'+1}$ gives

$$\beta^{p-1} = \beta^{-k+e'} \le \mu < \beta^{-k+e'+1} = \beta^p,$$

and in the second case $-p \le k + e_c - E_{\min} \le -1$ and $\mu = \mu' \beta^{e_c - E_{\min}}$ yield

$$\beta^{-1} \le \beta^{k+p-1+e_c-E_{\min}} = \beta^{e'+e_c-E_{\min}} \le \mu < \beta^{e'+1+e_c-E_{\min}} = \beta^{k+p+e_c-E_{\min}} \le \beta^{p-1}.$$

That means, due to $e = E_{\min}$, that x belongs to the second or third case of Table II, and the specification is satisfied because $x = \mu \cdot \beta^e$. In the third case of (30), $k + e_c + p \le E_{\min} - 1$ implies

$$0 < x = \frac{m_a}{m_b}\beta^{e_c} < \beta^{e'+1+e_c} = \beta^{k+p+e_c} \le \beta^{E_{\min}-1} \le \frac{1}{2}\beta^{E_{\min}},$$

so that $e = E_{\min}$ yields $\mu = \beta^{-e}x \in (0, \frac{1}{2})$ and $\mu x > 0$. Thus, again (μ, e) is according to the representation in Table II. Finally, by (3) all integer computations concerning the exponent e meet (17).

Concerning the powers of β in (30), we use (31) in the first, and $0 < -k - e_c + E_{\min} \le p$ in the second case, so that only powers β^ℓ with $0 \le \ell \le 2p-1$ are used.

3.6. Square root

Let $x := \sqrt{a}$ for positive a. We use the notation (8) and define $\mu' := \sqrt{m_a}$, so that $x = \mu'\beta^{e_a/2}$. According to Subsection 3.2, $e' := \lfloor \log_\beta(m_a) \rfloor$ is effectively computable and

$$0 \le e' \le p-1 \quad \text{and} \quad 1 \le \beta^{-e'} m_a < \beta. \tag{33}$$

Define the pair (A, e) by

$$(A, e) := \begin{cases} \left(\beta^{2p-2-e'} m_a\,,\ \frac{2-2p+e_a+e'}{2}\right) & \text{if } 2 \mid (e_a+e'),\ 1-2p+e_a+e' \ge 2E_{\min} \\ \left(\beta^{2p-1-e'} m_a\,,\ \frac{1-2p+e_a+e'}{2}\right) & \text{if } 2 \nmid (e_a+e'),\ 1-2p+e_a+e' \ge 2E_{\min} \\ \left(\beta^{e_a-2E_{\min}} m_a\,,\ E_{\min}\right) & \text{if } 1-2p+e_a+e' < 2E_{\min} \end{cases} \tag{34}$$

Setting $\mu := \sqrt{A}$, a straightforward computation yields $x := \sqrt{a} = \mu \cdot \beta^e$. The factors of m_a in the first two cases are integers because $2p-2-e' \ge p-1 \ge 0$, and

$$\beta^{2p-2} \le \beta^{2p-2-e'} m_a < \beta^{2p-1-e'} m_a < \beta^{2p}$$

shows (15) in the first two cases. In the third case,

$$0 \le -E_{\min} \le e_a - 2E_{\min} \le 2p-1-e' \le 2p-1 \tag{35}$$

shows that the factor of m_a is an integer, and

$$\beta^{e_a-2E_{\min}} m_a < \beta^{e_a-2E_{\min}+e'+1} < \beta^{2p}$$

proves that A satisfies (15) in the third and therefore in all cases. Moreover, the quantities e' and A are effectively computable. In the first two cases, using (33),

$$\beta^{p-1} = \beta^{\frac{2p-2}{2}} \le \beta^{\frac{2p-2}{2}} (\beta^{-e'} m_a)^{\frac{1}{2}} \le \sqrt{A} \le \beta^{\frac{2p-1}{2}} (\beta^{-e'} m_a)^{\frac{1}{2}} < \beta^{\frac{2p-1}{2}} \sqrt{\beta} = \beta^p.$$

In the third case, the condition implies $e_a - 2E_{\min} + e' + 1 \le 2p-1$, so that, again using (33),

$$0 < \sqrt{A} < \beta^{\frac{e_a-2E_{\min}+e'+1}{2}} \le \beta^{p-\frac{1}{2}}.$$

Since $e = E_{\min}$ in that case, the pair (μ, e) satisfies the second or third line of the specification in Table II. Finally, by (3) all integer computations concerning the exponent e satisfy (17). In particular, division by 2 occurs only for even numerator.

Concerning the powers of β in (34), we use $0 \le e' \le p-1$ to see $p-1 \le 2p-2-e' < 2p-1-e' \le 2p-1$ in the first two, and (35) in the third case, so that only powers β^ℓ with $0 \le \ell \le 2p-1$ are used.

4. IMPLEMENTATION USING `UINT64`

For integers A, B computed by the methods of the previous sections we show how to compute the correctly rounded image of A/B and $\sqrt{A}$. Note that in Matlab the rounding does not affect the result of an integer operation, i.e., for `uint64` variables `A,B` with real quotient q and $m \le q < m+1$ for $m \in \mathbb{N}$, the Matlab statement `Q = A/B` gives

$$\mathtt{Q} = \begin{cases} m & \text{if } q < m + 0.5 \\ m+1 & \text{otherwise} \end{cases}$$

independent of the rounding mode[4]. The result of a division by zero is $2^{64}-1$ if the numerator is nonzero, otherwise the result is zero.

Our general assumption for the implementation using `uint64` is

$$\beta^p \le 2^{32} \qquad \text{for } \circ \in \{+,-,\times,/,\sqrt{\cdot}\}, \tag{36}$$

so that the positive integers A, B in the interval $(0, \beta^{2p})$ are representable in `uint64`; the exception (25) will be addressed separately. We first treat the four basic operations, and after that the square root. Then

$$\mathtt{R} = \mathtt{rem}(\mathtt{A}, \mathtt{B})$$

assures[5] that `Q = (A-R)/B` is an integer and computed in `uint64` without error. It follows `A = BQ + R`. We show that $M = \mathrm{fl}^{\mathbb{Z}}_{\varrho}(\mu) = \mathrm{fl}^{\mathbb{Z}}_{\varrho}(A/B)$ for the rounding modes in (5) and $\circ \in \{+,-,\times,/\}$ is, using the operations in `uint64`, correctly computed according to

$$\begin{array}{ll} \varrho = \square_A & \mathtt{M} = \mathtt{A}/\mathtt{B} \\ \varrho = \square_E & \mathtt{M} = \begin{cases} \mathtt{Q} & \text{if } 2\mathtt{R} < \mathtt{B} \\ \mathtt{Q}+1 & \text{if } 2\mathtt{R} > \mathtt{B} \\ \mathtt{Q} & \text{if } \mathtt{R} = \mathtt{B}/2 \text{ and } 2*(\mathtt{Q}/2) = \mathtt{Q} \\ \mathtt{Q}+1 & \text{otherwise} \end{cases} \\ \varrho \in \{\nabla, \diamond\} & \mathtt{M} = \mathtt{Q} \\ \varrho = \Delta & \mathtt{M} = \begin{cases} \mathtt{Q}+1 & \text{if } \mathtt{R} > 0 \\ \mathtt{Q} & \text{otherwise} \end{cases} \end{array} \tag{37}$$

Assume that we are not in the exceptional case (25), so that $B \le \beta^{2p-1}$ by (16). Since `Q` and `R` are computed without error, the computed value for `M` is correct for $\varrho \in \{\square_A, \nabla, \diamond, \Delta\}$. For $\varrho = \square_E$ we note that $2\mathtt{R} < 2\mathtt{B} \le \beta^{2p}$ implies that `2R` is computed without error as well and the first two case distinctions for $\square_E$ are effectively computable. Furthermore we use the trick that if neither of the first two cases of $\square_E$ applies, then `2R = B` follows, `B` must be even and therefore `R = B/2`. Thus, the computed value of `M` is correct for all five roundings $\varrho \in \{\square_A, \square_E, \nabla, \diamond, \Delta\}$.

Next, consider the exceptional case (25) where we have to treat the case $\beta^{2p} = 2^{64}$. Then $1 \le \mathtt{A} < 2^{64}-1$, and $\mathtt{B} = 2^{64}$ is not in `uint64` but stored[6] as `B'` $:= 2^{64}-1$. That means that `B` is replaced by `B'` in the case distinctions in (37). The remainder

[4] As noted by the referee, Matlab's function `idivide` offers rounding modes `fix`, `round`, `floor` and `ceil`. However, compared to the built-in integer division those are up to three times slower.

[5] Note that `rem(A,B)` denotes the nonnegative remainder of the integer division `A/B`.

[6] The result of an `uint64` operation is always in $[0, 2^{64}-1]$; if the true result exceeds that range, it is rounded into the corresponding boundary value.

`R = rem(A,B') = rem(A,B) = A` does not change, and `Q = (A-R)/B = (A-R)/B' = 0`. Furthermore, $1 \le \mathtt{A} < 2^{64} - 1$ implies that `A/B = A/B'`, so that the computed result is correct for $\varrho \in \{\square_A, \nabla, \diamond, \Delta\}$.

For $\varrho = \square_E$ the comparison `2R < B` changes into `2R < B'`. Now `R = A` and

$$\mathtt{2R} < \mathtt{B}' \;\Leftrightarrow\; \mathtt{2A} < 2^{64} - 1 \;\Leftrightarrow\; \mathtt{A} \le 2^{63} - 1$$

imply that `Q = 0` is correct in the first case. Furthermore, with operations in `uint64`,

$$\mathtt{2R} > \mathtt{B}' \;\Leftrightarrow\; \mathtt{2A} > 2^{64} - 1$$

is not possible in `uint64` because $2^{64} - 1$ is the maximal integer in `uint64`. Next, in `uint64` division $\mathtt{B'/2} = 2^{63}$, so that

$$\mathtt{R} = \mathtt{B}'/2 \;\Leftrightarrow\; \mathtt{A} = 2^{63}$$

and `Q = 2*(Q/2) = 0` imply that the result $\mathtt{M} = 0$ in rounding ties to even is correct. Recall that the second case `2R > B` is not possible for computation in `uint64`. Thus, if neither of the first three cases in the case distinction for $\square_E$ holds true, then $\mathtt{A} > 2^{63}$ and the rounded result `M = 1` is correct.

For the square root we have to compute $\lfloor\sqrt{A}\rfloor$ for the given `uint64` integer `A` with $1 \le \mathtt{A} < \beta^{2p}$. Consider the Matlab code

```
a = double(A);
s = sqrt(a);
S = uint64(fix(s));
```

The first statement may cause a conversion error because of the 53-bit precision in `binary64`, but in the last statement the type cast by `uint64` does not change the result because $s \le \beta^p \le 2^{32}$. Hence, $\mathtt{S} = \mathtt{fix(s)}$. We will show

$$\mathtt{S} - 1 \le \lfloor\sqrt{A}\rfloor \le \mathtt{S}. \tag{38}$$

The standard error analysis for `binary64` with $\mathbf{u} = 2^{-53}$ denoting the relative rounding error unit yields

$$A(1-\mathbf{u}) \le a \le A(1+\mathbf{u}) \qquad \text{and} \qquad s/(1+\mathbf{u}) \le \sqrt{a} \le s/(1-\mathbf{u}).$$

Suppose $m^2 \le A < (m+1)^2$ for an integer m in $[1, \beta^p)$. Note that m is exactly representable in `binary64`. Then $\lfloor\sqrt{A}\rfloor = m$ and $\beta^p \le 2^{32}$ give

$$\mathtt{S} = \mathtt{fix(s)} \le \mathtt{s} \le (1+\mathbf{u})\sqrt{a} \le \sqrt{A}(1+\mathbf{u})^{3/2} < (m+1)(1+\mathbf{u})^{3/2} < \lfloor\sqrt{A}\rfloor + 2$$

and show the left inequality in (38). Conversely, first suppose $A = m^2$. Then

$$|\sqrt{a} - m| = \frac{|a - A|}{\sqrt{a} + \sqrt{A}} \le \frac{\mathbf{u}A}{\sqrt{A}(\sqrt{1-\mathbf{u}} + 1)} < \mathbf{u}m$$

shows that $\mathtt{s} = \mathtt{sqrt(a)} = \mathtt{sqrt(double(m^2))} = m = \sqrt{A}$ if $m = \sqrt{A}$ is not a power of 2, and otherwise that is trivially true. Since $m = \sqrt{a}$ is in `binary64`, it follows $s = \sqrt{A}$ and $S = \lfloor\sqrt{A}\rfloor$ if $A = m^2$. It remains the case $m^2 < A < (m+1)^2$. Then $\mathtt{double(A)} \ge \mathtt{double(m^2)}$ yields

$$\mathtt{s} = \mathtt{sqrt(A)} \ge \mathtt{sqrt(double(m^2))} = m,$$

and therefore $\mathtt{S} = \mathtt{fix(s)} \ge m = \lfloor\sqrt{A}\rfloor$. That proves the right inequality in (38). An example with equality on the left in (38) is $m = 67108865$ and $A = m^2 - 1$, so that $\lfloor\sqrt{A}\rfloor = m - 1$, but $S = m$.

It is clear from the proof that rarely $\lfloor\sqrt{A}\rfloor = S-1$. Now $A \le \beta^{2p}-1$ and $S \le \lfloor\sqrt{\beta^{2p}-1}\rfloor < \beta^p$ imply that $S^2 > A$ is effectively decidable. By setting $R := S-1$ if $S^2 > A$ and $R := S$ otherwise, we obtain the true downward rounded square root $R := \lfloor\sqrt{A}\rfloor$ of A.

The rounded image $M = \mathrm{fl}_\varrho^{\mathbb{Z}}(\mu) = \mathrm{fl}_\varrho^{\mathbb{Z}}(\sqrt{A})$ is computed using `uint64` arithmetic according to

$$\begin{array}{ll} \varrho \in \{\nabla, \diamond\} & \mathtt{M} = \mathtt{R} \\ \varrho \in \{\square_A, \square_E\} & \mathtt{M} = \begin{cases} \mathtt{R} & \text{if } \mathtt{A} - \mathtt{R} \le \mathtt{R}^2 \\ \mathtt{R}+1 & \text{otherwise} \end{cases} \\ \varrho = \Delta & \mathtt{M} = \begin{cases} \mathtt{R} & \text{if } \mathtt{A} = \mathtt{R}^2 \\ \mathtt{R}+1 & \text{otherwise} \end{cases} \end{array} \tag{39}$$

The rationale behind that is as follows. Since always $A \ne (R+\frac{1}{2})^2$, no care about ties is necessary.[7] Therefore, in rounding to nearest, the result is R if

$$A < (R+\frac{1}{2})^2 \quad \Leftrightarrow \quad A \le R^2 + R \quad \Leftrightarrow \quad A - R \le R^2$$

with the latter being effectively decidable. Since $R^2 \le A$, the condition $A = R^2$ is effectively decidable as well. The correctness of (39) for the other rounding modes is clear, and that finishes the part how to compute the correctly rounded result in `uint64`.

When emulating precision-p base-β arithmetic in `uint64`, the maximal precision $p = \lfloor 32/\log_2(\beta)\rfloor$ for base β is given in Table III.

Table III. Maximal precision p for emulating base-β arithmetic in `uint64`.

base β	2	3	4	5	6	7	8	9	10	16	32	64
max p	32	20	16	13	12	11	10	10	9	8	6	5

5. IMPLEMENTATION USING `BINARY64`

In order to emulate the precision-p base-β arithmetic in a precision-q binary floating-point arithmetic, we need stronger assumptions on the precision p, namely

$$\begin{array}{ll} \beta^{2p} \le 2^q & \text{if } \circ \in \{+,-,\times\} \\ \beta^{2p} \le 2^{q-1} & \text{if } \circ = / \\ \beta^{2p} \le 2^{q-3} & \text{if } \circ = \sqrt{\cdot} \end{array} \tag{40}$$

Later we will show that the restrictions in (40) on p cannot be improved. When using `binary64` we have $q = 53$.

In addition to (4) and following, we denote by $\mathrm{fl}_\varrho^{\mathbb{Z}}$ rounding into $\mathbb{Z}$, by $\mathrm{fl}_\varrho^{(p)}$ rounding into precision-p base-β, and by $\mathrm{fl}_\varrho^{(2)}$ rounding into the precision-q binary floating-point arithmetic, all following the rounding mode ϱ. Since, according to (40), all integers $\{m \in \mathbb{Z}\colon\ |m| < \beta^{2p}\}$ are exactly representable in the precision-q binary floating-point

[7] That is not true without the assumption $E_{\min} \le 0 \le E_{\max}$ as by a 3-digit decimal arithmetic and exponent range $(E_{\min}, E_{\max}) = (3,5)$. Then $x = 225 \cdot 10^4$ implies that $\sqrt{x} = 15 \cdot 10^2$ is equal to the midpoint between the denormalized numbers $1 \cdot 10^3$ and $2 \cdot 10^3$.

arithmetic, we can compute for the four basic operations and the square root the non-negative quantities A, B as described in the previous sections. Then (15) and (16) imply that $1 \le A, B < \beta^{2p} \le \beta^q$. Note that the special case (25) causes no problems because (40) implies that $B = \beta^{2p}$ is representable in the precision of the binary floating-point arithmetic.

Denote $\mu = A/B$ for $\circ \in \{+, -, \times, /\}$ and $\mu = \sqrt{A}$ for $\circ = \sqrt{\cdot}$. Then $\mu \in \mathcal{M} := (-\beta^{2p}, \beta^{2p})$ because μ satisfies the representation in Table II. We will show that

$$\mathrm{fl}_\varrho^{\mathbb{Z}}(\mu) = \mathrm{fl}_\varrho^{\mathbb{Z}}(\mathrm{fl}_\varrho^{(2)}(\mu)) \qquad \text{for } \varrho \in \{\square_E, \square_A, \nabla, \Delta, \diamond\}, \tag{41}$$

i.e., there is no harm by double rounding and $\mathrm{fl}_\varrho^{(p)}(x) = M \cdot \beta^e$ for the pair (M, e) as in Subsection 3.1.

5.1. Double rounding in `binary64`

The set $\mathcal{S}_\varrho \subset \mathbb{R}$ of "switching points"[8] of a rounding $\varrho \in \{\square_E, \square_A, \nabla, \Delta, \diamond\}$ is defined to be the set of real numbers s with the property that in every ε-neighborhood of s there exist x_1, x_2 with $\mathrm{fl}_\varrho(x_1) \ne \mathrm{fl}_\varrho(x_2)$. For rounding of $\mu \in \mathcal{M}$ into $\mathbb{Z}$ it follows $\mathcal{S}_\varrho = \mathcal{M} \cap \mathbb{Z}$ for a directed rounding $\varrho \in \{\nabla, \Delta, \diamond\}$, whereas $\mathcal{S}_\square = \{m - 0.5 : m \in \mathcal{M}\} \cup \{(\beta^p - 0.5) \cdot \beta^E\}$ for a nearest rounding $\square \in \{\square_E, \square_A\}$. Any real μ not in $\mathcal{M}$ must be rounded to one of its neighbors in $\mathcal{M}$, and there is a relevant switching point s deciding that based on the order relation between x and s.

We first consider directed roundings. Then the set of relevant switching points for $\mathrm{fl}_\varrho^{\mathbb{Z}}(\cdot)$ are the integers from $-\beta^p + 1$ to $\beta^p - 1$. An erroneous double rounding can only occur if μ and $\mathrm{fl}_\varrho^{(2)}(\mu)$ are on opposite sides of a switching point s including the case that one of them is equal to s. But $q \ge p$ implies that all of them are representable in precision-q binary, so this is not possible by the monotonicity of the rounding (7).

It remains to show that erroneous double rounding cannot occur for a nearest rounding. In that case the relevant set of switching points for $\mathrm{fl}_\square^{\mathbb{Z}}(\cdot)$ with $\square \in \{\square_E, \square_A\}$ are the midpoints of adjacent integers from 0 to β^p. By $q \ge p + 1$ all of them are representable in precision-q binary as well. Again, erroneous double rounding can only occur if μ and $\mathrm{fl}_\square^{(2)}(\mu)$ are on opposite sides of such a switching point.

Let s be the relevant switching point for μ for the rounding $\mathrm{fl}_\square^{\mathbb{Z}}(\cdot)$ with $\square \in \{\square_E, \square_A\}$. We first consider the four basic operations, that is the case $\mu = A/B$ with $\beta^{2p} \le 2^q$ for $\circ \in \{+, -, \times\}$ and $\beta^{2p} \le 2^{q-1}$ for division, respectively.

Since s is representable in precision-q binary and rounding is monotone, an erroneous double rounding can only occur if $\mu \ne s$ but $\mathrm{fl}_\square^{(2)}(\mu) = s$. The rounding in q-bit binary and $\mu > 0$ imply

$$|s - \mu| = |\mathrm{fl}_\square^{(2)}(\mu) - \mu| \le \mathbf{u}\mu$$

for the relative rounding error unit $\mathbf{u} := 2^{-q}$. Moreover, $\mu \ne s$ implies $s - \mu = s - A/B \ne 0$ for $A, B \in \mathbb{Z}$ with $1 \le A, B < \beta^{2p}$ and $0 < \mu < \beta^p$. Since s is half-way between adjacent integers, $2sB - 2A$ is a nonzero integer.

First, suppose that $|2sB - 2A| \ge 2$. Then (40) yields

$$\frac{1}{B} \le \frac{|2sB - 2A|}{2B} = |s - \mu| \le \mathbf{u}\mu \le \beta^{-2p}\frac{A}{B} < \frac{1}{B} \tag{42}$$

for $\circ \in \{+, -, \times, /\}$, a contradiction.

[8] called rounding boundary in [Brent and Zimmermann 2010]

Second, suppose that $|2sB - 2A| = 1$. For division we use $\beta^{2p} \le 2^{q-1}$ and obtain similarly to (42)

$$\frac{1}{2B} = \frac{|2sB - 2A|}{2B} = |s - \mu| \le \mathbf{u}\mu \le \frac{1}{2}\beta^{-2p}\frac{A}{B} < \frac{1}{2B},$$

again a contradiction. It remains $\circ \in \{+, -, \times\}$ provided that $|2sB - 2A| = 1$. We first note that according to (19) and (22) the denominator B is always a power of β, say $B = \beta^k$, and $2s \in \mathbb{N}$ and $|2sB - 2A| = 1$ imply that β is odd. Next we use the refined error estimate [Sterbenz 1974; Knuth 1998]

$$\frac{|s - \mu|}{|\mu|} \le \frac{\mathbf{u}}{1 + \mathbf{u}} < \mathbf{u}. \tag{43}$$

If $A \le 2^{q-1}$, then (43) implies with

$$\frac{1}{2B} = \frac{|2sB - 2A|}{2B} = |s - \mu| < \mathbf{u}\mu = 2^{-q}\frac{A}{B} \le \frac{1}{2B}$$

a contradiction. Hence we may assume without loss of generality $2^{q-1} < A$. The rounded result $s = \mathrm{fl}_{\square}^{(2)}(\mu)$ is half-way between adjacent integers v and $v + 1$, and $|2sB - 2A| = 1$ together with $s = v + \frac{1}{2}$ and $B = \beta^k$ being odd yields

$$2^{q-1} < A = sB \pm \frac{1}{2} \quad \Rightarrow \quad 2^{q-1} < sB.$$

Now $s \ne \mu$ and $s = v + \frac{1}{2}$ not being power of 2 implies with

$$\frac{1}{2B} = |s - \mu| \ge \mathbf{u}s = \frac{2^{-q}sB}{B} > \frac{1}{2B}$$

again a contradiction. That proves (41) for $\circ \in \{+, -, \times, /\}$ for q satisfying (40).

Finally we consider the square root. In that case $\mu = \sqrt{A}$ with $1 \le A < \beta^{2p}$ and $\beta^{2p} \le 2^{q-3}$. As before an erroneous double rounding can only occur if $\mu \ne s$ but $\mathrm{fl}_{\square}^{(2)}(\mu) = s$. It follows $4\mu^2 = 4A \ne 4s^2$, where $4s^2$ is an integer. Hence, abbreviating $M = \max(|\mu|, |s|)$,

$$1 \le |4(s^2 - A)| = 4|s + \mu||s - \mu| < 8M|s - \mu| \tag{44}$$

using $s \ne \mu$. Thus

$$\frac{1}{8M} < |s - \mu| \le \mathbf{u}M = 2^{-q}M \le \frac{1}{8}\beta^{-2p}M,$$

and therefore $M^2 > \beta^{2p}$, a contradiction to $M \le \beta^p$.

One may try to relax the condition on q in (40) for division into $\beta^{2p} \le 2^q$, and for the square root into $\beta^{2p} \le 2^{q-2}$. From the proofs above it follows that the only exceptions for that are $|2sB - 2A| = 1$ and $|4(s^2 - A)| = 1$, respectively. As we will see in the next subsection, these cases do indeed exist, so the conditions on q in (40) are optimal.

When emulating precision-p base-β arithmetic in IEEE-754 binary64 corresponding to 53 mantissa bits, the precision p for addition, subtraction and multiplication must satisfy $\beta^{2p} \le 2^{53}$, for division $\beta^{2p} \le 2^{52}$, and for the square root $\beta^{2p} \le 2^{50}$. Although the first condition is weaker, the maximal p is the same as for division. The corresponding maximal values of p for different bases are given in Table IV. Compared to Table III using `uint64` as underlying arithmetic allows at least one more β-digit precision.

Table IV. Maximal precision p for emulating base-β arithmetic in binary64.

base β	2	3	4	5	6	7	8	9	10	16	32	64
max p for $+,-,\times,/$	26	16	13	11	10	9	8	8	7	6	5	4
max p for $\sqrt{\cdot}$	25	15	12	10	9	8	8	7	7	6	5	4

5.2. Optimality of the choice of q

We will show that $\beta^{2p} \le 2^q$ for division and $\beta^{2p} \le 2^{q-2}$ for the square root does not suffice to avoid errors in the double rounding (41), i.e.,

$$\mathrm{fl}^{\mathbb{Z}}_{\square}(\mu) \neq \mathrm{fl}^{\mathbb{Z}}_{\square}(\mathrm{fl}^{(2)}_{\square}(\mu))$$

may happen. First, consider $\beta = 3$ and $p = 4$ with $\beta^{2p} = 6561 \le 2^q = 8192$ for $q = 13$. Define

$$A = 4455_{10} = 20010000_\beta \quad \text{and} \quad B = 67_{10} = 2111_\beta.$$

Denote by $\mathrm{fl}^r_b(\cdot)$ rounding to nearest in precision r and base b. Then

$$\mathrm{fl}^p_\beta(A/B) = \mathrm{fl}^p_\beta(2110.111022001\ldots_\beta) = 2110_\beta = 66_{10},$$

and, for $q = 2, p = 8$,

$$\mathrm{fl}^q_2(A/B) = \mathrm{fl}^q_2(1000010.011111100001\ldots_2) = 1000010.100000_2 = 66.5_{10} = 2110.\overline{1}_\beta, \quad (45)$$

the midpoint between 2110_β and 2111_β. Hence roundTiesToAway in base β implies $\mathrm{fl}^p_\beta(\mathrm{fl}^q_2(A/B)) = 2111_\beta = 67_{10}$ in contrast to (45). For roundTiesToEven in precision base $\beta = 3$ with $p = 7$ and $q = 23$, choosing

$$A = 4343382_{10} = 22011200000000_\beta \quad \text{and} \quad B = 2111_{10} = 2220012_\beta$$

produces

$$\mathrm{fl}^p_\beta(A/B) = \mathrm{fl}^p_\beta(2211012.111111022\ldots_\beta) = 2211012_\beta = 2057_{10},$$

but

$$\mathrm{fl}^q_2(A/B) = 2057.5_{10} = 2211012.111\ldots_\beta,$$

the midpoint between 2211012_β and 2211020_β. Hence rounding ties to even in base β results in $\mathrm{fl}^p_\beta(\mathrm{fl}^q_2(A/B)) = 2211020_\beta = 2058_{10}$.

One might think that for even base β things are different because the midpoint between adjacent base-β numbers is representable. That is not the case. Choose $\beta = 6$ and $p = 4$, so that $q = 21$. Then roundTiesToAway causes a double rounding error for

$$A = 1382832_{10} = 45350000_\beta \quad \text{and} \quad B = 1085_{10} = 5005_\beta,$$

and in roundTiesToEven for

$$A = 1303776_{10} = 43540000_\beta \quad \text{and} \quad B = 1027_{10} = 4431_\beta$$

for rounding to even in base β.

Next we show that $\beta^{2p} \le 2^{q-2}$ for square root does not suffice to avoid errors in the double rounding. Let $p = 4$ and $\beta = 12$, so that $q = 31$. Then $A = 20735_{10} = BBBB_\beta$ implies

$$\mathrm{fl}^p_\beta(\sqrt{A}) = \mathrm{fl}^p_\beta(BB.BABBBBBBBBBB8049\ldots_\beta) = BB.BB_\beta,$$

but

$$\begin{aligned}\mathrm{fl}^q_2(\sqrt{A}) &= \mathrm{fl}^q_2(1000111.111111111000111000111000100001\ldots_2)\\ &= 1000111.111111111000111000111001_2\\ &= BB.BB6000058416011 1A50B7A28\ldots_\beta\end{aligned}$$

just above the midpoint between $BB.BB_\beta$ and 100_β. Thus $\mathrm{fl}_\beta^p(\mathrm{fl}_2^q(\sqrt{A})) = 100_\beta = 144_{10}$ instead of $BB.BB_\beta$ due to a double rounding error both in roundTiesToEven and roundTiesToAway.

6. FINAL NORMALIZATION IN BINARY

So far we showed how to compute a proxy $(\mu, e) = \mathcal{P}(x)$ according to Table II. Let a rounding $\varrho \in \{\square_E, \square_A, \nabla, \Delta, \diamond\}$ be given, and set $M := \mathrm{fl}_\varrho^{\mathbb{Z}}(\mu)$. Note that $0 \le M \le \beta^p$. Then $\mathrm{fl}_\varrho(x) = M \cdot \beta^e$ if $\mathrm{fl}_\varrho(x)$ is in $\mathbb{F}_{p,\beta,E}$, i.e., is real and finite, but the representation (M, e) may not be according to Table I. A final normalization $\mathcal{N}\colon \mathbb{Z}^2 \to \mathbb{Z}^* \times \mathbb{Z}$ is defined by

$$\mathcal{N}(M,e) := \begin{cases} (M/\beta, e+1) & \text{if } |M| = \beta^p, e < E_{\max} \\ (+\infty, E_{\max}) & \text{if } M = \beta^p, e \ge E_{\max}, \varrho \in \{\square_E, \square_A, \Delta\} \\ (\beta^p - 1, E_{\max}) & \text{if } M = \beta^p, e \ge E_{\max}, \varrho \in \{\nabla, \diamond\} \\ (-\infty, E_{\max}) & \text{if } M = -\beta^p, e \ge E_{\max}, \varrho \in \{\square_E, \square_A, \nabla\} \\ (-\beta^p + 1, E_{\max}) & \text{if } M = -\beta^p, e \ge E_{\max}, \varrho \in \{\Delta, \diamond\} \\ (M, e) & \text{otherwise.} \end{cases} \tag{46}$$

Checking the individual cases verifies that the pair $(M', e') := \mathcal{N}(M, e)$ is the correctly rounded precision-p base-β result $\mathrm{fl}_\varrho(x)$ and satisfies the representation in Table I.

7. CONVERSION BETWEEN BINARY64 AND BASE β

We add some remarks on the conversion from `binary64` into a precision-p base-β format. As has been noticed, the exponent range of the latter may be huge. Very large or very small `binary64` input may not be converted without error, so some care is necessary to avoid overflow or underflow.

Let `d` be a positive `binary64` number. Define

$$\ell := \begin{cases} 51 + \lceil \log_2(\beta^p) \rceil & \text{if } d < 2^{-1023}\beta^p \\ -2 & \text{if } d > 2^{1022} \\ 0 & \text{otherwise} \end{cases} \tag{47}$$

and set $d' := 2^\ell d$. In the first case,

$$d' = 2^{51+\lceil \log_2(\beta^p) \rceil} d \ge 2^{51}\beta^p 2^{-1074} = 2^{-1023}\beta^p,$$

so that in either case $2^{-1023}\beta^p \le d' \le 2^{1022}$. Setting $e := \lfloor \log_2(d') \rfloor$ it follows

$$2^{-1022} \le 1/d' \le \beta^{p-1}/d' \le \beta^{p-1-e} < \beta^p/d' \le 2^{1023},$$

so that β^{p-1-e} is a normalized floating-point number. Define $m' := d'\beta^{p-e-1}$. Then

$$\beta^{p-1} = m'\beta^e/d' \le m' = d'\beta^{p-e-1} < \beta^p,$$

and $d' = m'\beta^{1+e-p}$ is a precision-p base-β representation for infinite exponent range. If one of the first two cases in (47) applies, then set $e' := \lfloor \log_2 2^{-\ell} m' \rfloor$, note that $2^{-\ell}m'$ is a normalized floating-point number, and set $m := 2^{-\ell}m'\beta^{p-e'-1}$. Then $\beta^{p-1} \le m < \beta^p$ and

$$d = 2^{-\ell}d' = 2^{-\ell}m'\beta^{1-e-p} = m\beta^{2+e+e'-2p}$$

is a valid precision-p base-β representation of d, still for infinite exponent range. Finally, that representation is adapted to Table I.

8. SOME COMPARISON WITH OTHER PACKAGES

The arithmetic described in this note has been implemented in the flbeta-toolbox which will be part of Version 13 of INTLAB [Rump 1999], the Matlab/Octave toolbox for reliable computing. We use Matlab Version 2023a under Windows 10 on a Laptop. The data in the following tables V and VI show the time ratio against ordinary `binary64` floating-point. We compare the flbeta-toolbox using `uint64` and `binary64` representation, the quarter precision Matlab package fp8 and half precision package vfp16 from [Moler 2019], and flap from [Stewart 2009].

All packages but the last provide a correctly rounded result according to IEEE 754. That is not true for the flap package, so that the comparison is not quite fair. In addition, the flbeta-toolbox allows for bases from 2 to 64, so that the approach is more general than all other packages. In that sense the comparison is biased.

The tested operations displayed in Table V are $a+b$, $a\times b$, a/b and $\sqrt{a}$ for scalar a, b. We use $\beta = 2$ for the flbeta-toolbox; there is no difference in timing when using another base β. As can be seen there is hardly a difference between `uint64` and `binary64` representation for the flbeta-toolbox, and fp8 and vfp16 are generally faster than flbeta. Note that fp8 does not provide a square root.

Table V. Time ratio to `binary64` for $a+b$, $a*b$, a/b and $\sqrt{a}$ and scalar a, b.

operation	flbeta(uint64)	flbeta(binary64)	fp8	vfp16	flap
plus	78.7	75.1	40.0	60.0	28.5
mtimes	84.2	81.1	43.2	67.5	32.1
mrdivide	86.1	83.0	45.4	67.0	66.1
sqrt	48.9	48.4	-	46.9	22.2

Next we perform the same test for a being a 10×10 matrix and scalar b and show the results in Table VI. In addition to the previous operations we show the time ratio for reading access `a(index)` and writing access into the array a. Again there is not much difference between `uint64` and `binary64` representation for the flbeta-toolbox, but fp8 and vfp16 are mostly slower than flbeta.

Table VI. Time ratio to `binary64` for a 10×10 matrix a and scalar b.

operation	flbeta(uint64)	flbeta(binary64)	fp8	vfp16	flap
plus	116.0	103.9	204.0	229.3	37.3
mtimes	101.5	98.4	143.8	236.7	39.2
mrdivide	103.8	98.8	141.7	227.1	77.3
sqrt	39.8	38.8	-	163.2	21.4
indexread	533.2	532.1	1038.0	1731.9	248.7
indexwrite	749.2	755.1	685.1	1102.2	262.7

There are three other packages, namely CPFloat [Fasi and Mikaitis 2020], chop [Higham and Pranesh 2019] and floatp [Meurant 2020], all of them emulating binary floating-point arithmetic in different formats. The first two packages CPFloat and chop provide a rounding from binary64 into the desired binary format. Arithmetic expressions like $a+bc$ are evaluated as `chop(a+chop(b*c))`, and the result is correctly rounded without double rounding [Roux 2014; Rump 2017] because the desired binary precision does not exceed 26 bits.

The comparison with respect to CPFloat is biased because it could not be compiled using OpenMP, so only one thread is used.

Computations in CPFloat and chop are fast because there is no operator concept implying severe interpretation overhead, only roundings have to be added. Therefore, comparison with the other packages is not quite fair because no toolbox with operators is provided. The third package floatp is a toolbox with operator overloading, however, it seems not tuned for performance. In particular, division and square root is implemented using a Newton iteration, therefore we did not include it in the previous tests.

The data up to now is particularly biased because our flbeta-toolbox, in contrast to all others, emulates a general base-β arithmetic whereas other packages are restricted to binary or decimal. In that sense comparing the other packages to INTLAB's fl-toolbox is more fair. The latter is based on [Rump 2017] where the emulation of a lower precision base-β arithmetic based on a given base-β arithmetic is shown. The fl-toolbox implements the special case $\beta = 2$, i.e., simulation of binary in binary.

Given two matrices A, B, we show results for all packages including CPFloat, chop and floatp for the matrix multiplication AB in the sense that all intermediate products and sums are computed using the emulated arithmetic. In order to save roundings and to reduce interpretation overhead [Rump 2012] we use outer products $AB = \sum A_{:k}B_{k:}$ to compute AB. Outer products are used in flap and the toolboxes of INTLAB, and we added the matrix multiplication to CPFloat, chop and fp8. For flbeta we emulated a decimal arithmetic with maximal precision, i.e., $p = 7$ for `binary64` underlying arithmetic; the timing for `uint64` is almost identical.

The packages CPFloat and chop provide the rounding of real numbers (vectors, matrices) into the desired format. As has been mentioned before a statement like $a + bc$ has to be transformed, e.g., into `chop(a+chop(b*c))`. The other packages provide Matlab classes so that `a+b*c` becomes an executable statement.

This degree of comfort is payed by a significant overhead for interpreting the user-defined classes. To add a more fair comparison we wrote small Matlab toolboxes for CPFloat and chop using classes and operator overloading. We show results for both the original code and when using classes. Moreover, we add data on INTLAB's fl-toolbox emulating 26-bit precision binary arithmetic.

Table VII. Matrix multiplications $A \cdot B$ for $n \times n$ matrices A, B, time in [sec].

n	fl	CPFloat(class)	chop(class)	flap	flbeta	fp8	vfp16	floatp
10	0.00069	0.00004(20)	0.00059(98)	0.0012	0.0090	0.0080	0.068	1.71
25	0.0013	0.00014(42)	0.0028(49)	0.0057	0.033	0.084	1.00	25.4
50	0.0034	0.00057(138)	0.012(21)	0.024	0.11	0.63	12.2	
100	0.018	0.0034(73)	0.059(115)	0.14	0.59	4.9		
200	0.089	0.030(103)	0.38(93)	1.15	3.83	39.1		
500	3.6	0.66(1.77)	9.2(19.7)	16.9	65.0			
1000	29	5.2(13.4)	75(153)	134				

The timings are shown in Table VII, where places are left empty when large computing times are to be expected. In order to save horizontal space, the timings for CPFloat and chop without and with using classes (in parenthesis) are displayed in one column. The interpretation is that the trailing digits are to be replaced by the digits in parenthesis to obtain the timing for using classes. For example, the timing for $n = 50$ for the original CPFloat without classes is 0.00057, the timing with classes is 0.00138 seconds.

According to the documentation, the newer CPFloat has the same functionality as chop, but in our measurements it is about one order of magnitude faster. CPFloat is also faster than our fl-toolbox.

Both CPFloat and chop, also with toolbox, and flap are faster than flbeta, where fp8, vfp16 and floatp are slower. However, the flbeta-toolbox addresses arbitrary bases β, and that has a price.

We tested other combinations of precisions and bases, but encountered no surprises concerning computing time. As the algorithms are the same for different combinations of precision and bases, that comes as expected.

The flbeta-toolbox offers to use an arbitrary base $\beta \geq 2$, provides correct rounding, treatment of subnormal numbers and exceptions according to the IEEE754 standard. Sparse data storage and an interval arithmetic with precision-p base-β bounds is included as well. This makes the package well suited for experimenting in non-binary bases with variable precision floating-point arithmetic in Matlab.

ACKNOWLEDGMENTS

Many thanks to Massimiliano Fasi for many constructive remarks and for helping to get the Matlab version of CPFLoat to run under Windows. The author likes to express his warm thanks to the two anonymous referees for their thorough reading and many very valuable comments.

REFERENCES

R. Brent and P. Zimmermann. 2010. *Modern computer arithmetic*. Cambridge University Press, New York, NY, USA (2010).

M. Fasi and M. Mikaitis. 2020. CPFloat: A C library for simulating low-precision arithmetic *MIMS EPrint 2020.22*, University of Manchester, (2020), accepted for publication in ACM TOMS https://doi.org/10.1145/3585515, see also http://eprints.maths.manchester.ac.uk/2855/1/fami22.pdf

Samuel A. Figueroa. 1995. When is double rounding innocuous? *SIGNUM Newsl.* 30, 3 (1995), 21–26.

G. Flegar, F. Scheidegger, V. Novaković, G. Mariani, A.E. Tomàs, A.C.I. Malossi, and E.S. Quintana-Ortí. FloatX: A C++ library for customized floating-point arithmetic. *ACM Trans. Math. Softw.*, 45:1–40, 2019.

L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. Research Report RR-5753, INRIA, 2005. Code and documentation available at http://hal.inria.fr/inria-00000818.

N. J. Higham. 2002. *Accuracy and stability of numerical algorithms* (2nd ed.). SIAM Publications, Philadelphia.

N.J. Higham and S. Pranesh. Simulating low precision floating-point arithmetic. *SIAM Journal on Scientific Computing*, 41(5):C585–C602, 2019.

IEEE 1987. *ANSI/IEEE 854-1987, Standard for radix-independent floating-point arithmetic*, 1987.

IEEE 2008. *ANSI/IEEE 754-2008: IEEE standard for floating-point arithmetic*. IEEE, New York.

IEEE 2019. IEEE standard for floating-point arithmetic. *IEEE Std 754-2019 (revision of IEEE 754-2008)*, pages 1–84, 2019.

C.-P. Jeannerod and S.M. Rump. On relative errors of floating-point operations: optimal bounds and applications. *Math. Comp.*, 87:803–819, 2017.

D.E. Knuth. 1998. *The art of computer programming: seminumerical algorithms* (third ed.). Vol. 2. Addison Wesley, Reading, Massachusetts.

V. Lefèvre. 2013. Sipe: a mini-library for very low precision computations with correct rounding. (2013). http://hal.inria.fr/hal-00864580 submitted.

G. Meurant. FLOATP_toolbox, Matlab software, variable precision floating point arithmetic. https://gerard-meurant.pagesperso-orange.fr/soft_meurant_n.html.

C.B. Moler. Variable format half and quarter precision floating-point arithmetic. Cleve's corner, 2019. https://de.mathworks.com/matlabcentral/fileexchange/59085-cleve_lab.

J.-M. Muller, N. Brunie, F. de Dinechin, C.-P. Jeannerod, M. Joldes, V. Lefèvre, G. Melquiond, R. Revol, and S. Torres. 2018. *Handbook of floating-point arithmetic*. 2nd edition, Birkhäuser, Boston.

P. Roux. Innocuous double rounding of basic arithmetic operations. *Journal of Formalized Reasoning*, 7(1):131–142, Jan. 2014.

S.M. Rump. 1999. INTLAB - INTerval LABoratory. In *Developments in Reliable Computing*, Tibor Csendes (Ed.). Kluwer Academic Publishers, Dordrecht, 77–104. http://www.ti3.tu-harburg.de/rump/intlab/index.html

S.M. Rump. Fast interval matrix multiplication. *Numerical Algorithms*, 61(1):1–34, 2012.

S.M. Rump. IEEE754 precision-k base-β arithmetic inherited by precision-m base-β arithmetic for $k < m$. *ACM Trans. Math. Software*, 43(3), 2017.

P.H. Sterbenz. *Floating-point computation*. Prentice Hall, Englewood Cliffs, NJ, 1974.

G.W. Stewart. 2009. Flap: A Matlab package for adjustable precision floating-point arithmetic. http://www.cs.umd.edu/~stewart/flap/flap.html. (2009).

G.W. Stewart. 2014. private communication. (2014).

Received February 0000; revised March 0000; accepted June 0000