Interval Computations
No 4, 1994

Rounding of Floating Point Intervals

Marc Daumas and David W. Matula

Correct rounding of the infinitely precise arithmetic as prescribed for the floating point operations is not always attainable with reasonable resource. Many functions return explicit or implicit interval outputs. Yet, the IEEE standard does not allow efficient use of this information. We present how the IEEE rounding mechanisms can be extended to accept interval rounding which is shown to be a first natural step toward non infinitely precise rounding. Along with the proposal of this faithful rounding for non-infinitely precise operation, we present the intrinsic properties of this new rounding mechanism.

Округление интервалов с плавающей точкой

М. Дома, Д. Матула

Корректное округление в арифметике с бесконечной разрядностью, специфицированное для операций с плавающей точкой, иногда требует для своей реализации неадекватного количества ресурсов. Хотя многие функции возвращают результат в виде явного или неявного интервала, стандарт IEEE не позволяет эффективно использовать эту информацию. Нами продемонстрировано, как можно дополнить механизмы округления стандарта IEEE введением интервального округления, которое, как показано, является первым естественным шагом на пути к округлению с конечной разрядностью. Предлагая такое точное округление для операций с конечной разрядностью, мы также выводим основные свойства, присущие этому новому механизму округления.

1 Introduction

The IEEE floating point standard has been adopted for most PC and workstation platforms and is likely to become an available option on mainframes and supercomputers in their next generation. Several distinct features of the standard have contributed to its success in removing the chaos from previous non-standardized floating point numeric computation systems. For the primitive arithmetic operations (addition, multiplication, division, square root...) the user-controlled fully specified rounding mechanism dictates unique results. However, this high level of requirement is sometimes not accessible with a reasonable amount of hardware or in bounded computation time. Such cases are often limiting the precision on transcendental functions [2] (sine, cosine, logarithm, exponential...) and vector operations (dot product, matrix multiplication...) [3, 6].

Yet there is no widely accepted specification to extend these notions to interval rounding. We present a coherent, fully specified system for the rounding of small intervals. Given a small enough interval, less than 1 ulp wide, this mechanism returns one unique value specified by the rounding mode chosen by the user. This rounding mechanism guarantees also the correctness of the rounding regarding directed error. The notions introduced detail well defined rounding mechanism that can be applied to transcendental functions and vector arithmetic which are usually implemented to compute a small interval that contain the real result. The reader interested in an in-depth justification of the rounding operation may find some good arguments in [7]. The authors of the book use semi-morphisms on the structure of a superset.

Section 2 is a review of the IEEE standard features. It includes definitions of quantities related to a *unit in the last place* and useful properties for the argumentation of the rounding mechanisms. This is also an opportunity to review the limits of the IEEE standard rounding possibilities. We present in Section 3 the proposal for our first step toward extending the IEEE deterministic rounding.

2 The IEEE standard

2.1 Three new concepts

When the ANSI/IEEE 754–1985 standard for binary floating point arithmetic was defined [4], the committee introduced three very successful notions that are now considered as natural requirements for computer arithmetic.

Type hierarchy. The data representation—bit level encoding—of the floating point numbers is specified to allow easy interchanges between machines and standardization of the applications at the hardware level. The standard defines the semantic, the position and the size of each field, yet the actual organization of the byte in memory may depend on the low level implementation (big endian vs. little endian).

- The *sign* bit—each floating point number is stored as a couple with a separate absolute value field and a separate sign.
- The size of the *truncated mantissa* depends on the data format; since the number is normalized its first binary digit is always set to the value 1, hence it is omitted from the storage.
- The *exponent* is coded with a bias; the size of this field and the value of the bias are given by the format.

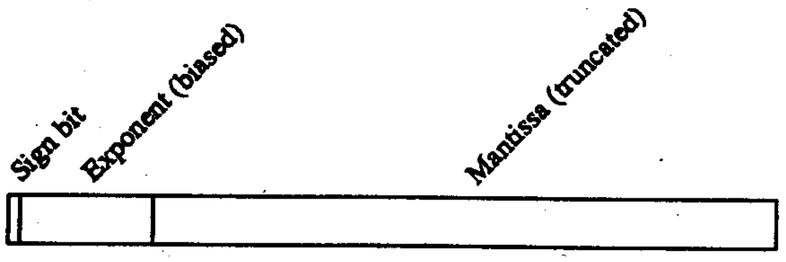
Moreover, the standard includes general ideas on extending the existing data hierarchy to higher levels of precision. We shall consider the data types: T_1 (single), T_2 (double), and T_4 (quad). The data type T conditions the numbers' precision (one unit in the last place, ulp) and the subset $\mathbf{F}$ (respectively $\mathbf{F}_1$, $\mathbf{F}_2$, and $\mathbf{F}_4$) of $\mathbf{R}$ which is supported. The set $\overline{\mathbf{F}}$ is the closure of F with the usual extension ($+\infty$, $-\infty$, NaN …).

Closure. To allow the computer to process floating point operations in a continuous stream of instructions, it is necessary to ensure that any operation returns a value. Closure is an important issue since the user should be responsible for the treatment—or the absence of treatment—of numerical exceptions. As an example, a division by zero should not stall all the machine forcing the user to restart the faulty process from the beginning. New values

These notions are presented in the standard [4]. The type hierarchy have been extended with the *de facto* new standards but it is still compatible with the underlying ideas who were the basis of the first data types definitions.

Type hierarchy

Type Name	Fraction (bits)	Exponent (bits)	Total Length (bytes)
Single	23	8	4
Single Extended	≥ 31	≥ 11	≥ 5
Double	52	11	8
Double Extended	≥ 63	≥ 15	≥ 10
PC-Double Extended	64	15	10
Quad	112	15	16



Closure

Infinities ($+\infty$, $-\infty$), Overflow, Denormalized, Not a Number (NaN)

Uniqueness

Operation	Domain	Result Specification
Addition	$\overline{\mathbf{F}}^2 \longrightarrow \overline{\mathbf{F}}$	$(v,w) \longmapsto s = \mathrm{Rx}(v+w)$
Multiplication	$\overline{\mathbf{F}}^2 \longrightarrow \overline{\mathbf{F}}$	$(v,w) \longmapsto p = \mathrm{Rx}(v \times w)$
Division	$\overline{\mathbf{F}}^2 \longrightarrow \overline{\mathbf{F}}$	$(v,w) \longmapsto q = \mathrm{Rx}(v/w)$
Square root	$\overline{\mathbf{F}} \longrightarrow \overline{\mathbf{F}}$	$v \longmapsto q = \mathrm{Rx}(\sqrt{v})$

Figure 1: The IEEE standard specification

are introduced into the set $\mathbf{F}$ to comply with this idea: projective infinity ($+\infty$ and $-\infty$), overflow representation (the correct mantissa is stored with a special exponent to signal the overflow) and not a number (NaN—to represent results that cannot be reduced, for example 0/0). In addition, the standard defines flags to trap on error; in this case, the user should install an exception handler.

Uniqueness. The rounding mechanism introduced by the IEEE standard has led to the notion of uniqueness; the result of any operation implemented by the standard is a totally specified unique floating point representable number $v \in \mathbf{F}$. The format specifies four rounding modes (RN — Round to Nearest, RU—Round Up towards $+\infty$, RD — Round Down towards $-\infty$ and RZ—Round towards Zero), and four operations according to the active rounding mode Rx. The result of an operation must be the exact rounding of the infinitely precise mathematical operation.

2.2 The *ulp* function

As a convention for notation and to allow a better understanding of this presentation: a real number will be represented by the letter $x \in \mathbf{R}$, a real interval by $[a, b] \subset \mathbf{R}$ and a finite floating point representable value (respectively arbitrary large) by the letters $v \in \mathbf{F}$ and $w \in \mathbf{F}$ (resp. $v \in \overline{\mathbf{F}}$ and $w \in \overline{\mathbf{F}}$).

$$x^+ = \min\left\{v \in \overline{\mathbf{F}} \mid v > x\right\} \qquad x^- = \max\left\{v \in \overline{\mathbf{F}} \mid v < x\right\}$$

$$ulp(v) = \begin{cases} v^+ - v & v \geq 0 \\ ulp(-v) & v < 0 \end{cases}$$

$$ulp(x) = ulp\Big(\mathrm{Trunc}(x)\Big)$$

$$v_{[\mathrm{Rx}]} = \{x \in \mathbf{R} \mid \mathrm{Rx}(x) = v\}$$

Figure 2: The *ulp* function

The set $\mathbf{F}$ of the representable numbers in floating-point format T is a non-uniform discrete set. For this reason, talking about relative error is sometimes dangerous. A unit in the last place is a quantity used frequently in floating point arithmetic; it is possible to define it as a function.

Definitions. Each set $\mathbf{F}$ is a finite subset of $\mathbf{R}$, thus this is a well ordered set and each real number $x \in \mathbf{R}$ has a unique successor in $\overline{\mathbf{F}}$. For the same reasons, $x \in \mathbf{R}$ has a unique predecessor in $\overline{\mathbf{F}}$.

Definition 1. *The type-T successor x^+ of the real number $x \in \mathbf{R}$ is the smallest number (possibly non finite) in $\overline{\mathbf{F}}$ larger than x. Identically the type-T predecessor x^- of the real number $x \in \mathbf{R}$ is the largest number (possibly non finite) in $\overline{\mathbf{F}}$ smaller than x.*

It is important to notice that for a representable floating point number $v \in \mathbf{F}$, the difference $v^+ - v$ and $v - v^-$ are not always equal. The *ulp* function is defined from these quantities. For some real numbers $x \in \mathbf{R}$ the values x^+ or x^- may not be finite. However, with the definition presented, the function *ulp* is finite on all number $v \in \mathbf{F}$ except possibly the largest and the smallest values of $\mathbf{F}$.

We have defined so far the function *ulp* on the set $\mathbf{F}$ of numbers representable in type T. We shall extended to two directions: to the real numbers and to the denormalized numbers. A denormalized number W is a word on the alphabet $\{0, 1\}$. For the denormalized numbers, we will make a distinction between the number (real value) and its expression (word). To avoid any confusion, we will note $|W| \in \mathbf{F}$ the floating point value of the denormalized number W using the usual conventions.

Definition 2. *The function ulp is the even function that associates to any type-T representable floating point non negative number $v \in \mathbf{F}^+$ the value of the difference $v^+ - v$. The ulp function is extended to real values and denormalized numbers. Let x be a real number, the value $ulp(x)$ is defined from the truncated value $v = Trunc(x)$: $ulp(x) = ulp(v)$. Identically, given a denormalized number W of rank l we define $ulp(W) = ulp(|W|)2^l$.*

A floating point value has two different meanings. When first introduced as a tool for physics, floating point numbers inherited from the properties of the measuring tools. It is usually possible in physics to get the sign of a quantity, so it is logical that the sign is coded independently from the absolute value, but it is barely impossible to get a result without an intrinsic uncertainty. When a floating point number is used to store a physical value, it represents an interval: its rounding set.

Definition 3. *The rounding set $v_{[Rx]}$ of a floating point number $v \in \mathbf{F}$ relative to the type T and the rounding mode Rx is the subset of $\mathbf{R}$ whose elements reduce to v by rounding.*

It is important to notice that sometimes a floating point value only represents one real number. This is verified when a process defines high precision numbers using multiple word operations for arbitrarily precise arithmetic: only the last word in the list of numbers represents an interval (its rounding set); all other numbers are the exact representations of their values.

Properties. For an easier understanding of this analysis, we will restrain to the floating point values $v \in \mathbf{F}$ that are away from the limits of the set $\mathbf{F}$ ($-\infty$, -0, $+0$, and $+\infty$), we will also restrict this work to normalized numbers. The behavior of the *ulp* function close to these points is very straight forward; however, the description of all special cases would make this presentation tedious. Identically the real number $x \in \mathbf{R}$ are implicitly chosen within the set of numbers that rounds to points within the subset just defined.

A binary number is represented by the infinite sum on a finite domain $\sum_{-\infty}^{+\infty} d_i 2^i$. The standard data format specifies that any floating point number $v \in \mathbf{F}$ is represented by a data structure with a $(n_m + 1)$-bit mantissa $1.m_v$ and a n_e-bit exponent e_v. Leading to the finite sum

$$(-1)^{s_v} 2^{e_v} \left(1 + \sum_{1}^{n_m} d_i 2^{-i} \right).$$

The IEEE standard types T_1, T_2, and T_4 are fully specified by these two integers n_m and n_e.

Looking at the literature, there are many definitions close to the one presented of the quantity *unit in the last place*; as a matter of fact, any natural definition of the function *ulp* is equivalent to the Definition 2 except possibly when $v = \pm 2^k$, $v = \pm(2^k)^+$ and $v = \pm(2^k)^-$.

Before going any further, we present quickly three lemmas to relate these definitions with the usual knowledge of relative error. The relative error is mathematically defined from an error bound Δ_{max} by $\epsilon = \Delta_{max}/x$. Application designers usually verify that the relative error is below 2^{n_m} or a fraction of this value. This also means that the process gives n_m bits of precision or something close to this value. It is interesting to connect the relative error and the *ulp* function.

Lemma 1. *The ulp function satisfies* $ulp(v) = 2^{e_v - n_m}$. *As a consequence, the function ulp defined on the representable floating point numbers is constant over intervals* $[2^i, 2^{i+1})$ *with a jump at* 2^i *for all* i. *For any representable floating point* v, *an* $ulp(v)$ *is exactly the value of the number encoded with the same exponent as* v (e_v) *and the mantissa equal to* $0.00\ldots001$.

Lemma 2. *For any real number* $x \in \mathbf{R}$ *there is at most one representable point in any interval containing* x *and of length* $\frac{1}{2}ulp(x) - \epsilon$ $(\epsilon > 0)$. *Vice versa, there is at least one representable value in any interval containing* x *and of length* $ulp(x) + \mu$ $(\mu \geq 0)$. *The value* $ulp(x)$ *is bounded by* $2^{\lceil \log_2 |x| \rceil - n_m}$.

Lemma 3. *A value* x *known from* x_0 *with a relative error bounded by* $\lambda \times 2^{-n_m} \leq 1/3$ *is known to be at most* $3/2\lambda \times ulp(x)$ *away from* x_0.

2.3 The limits of the standard

In the machine, infinitely precise rounding is often achieved by performing internal computation using more bits than the input or output formats. For example, the IEEE standard addition and multiplication are usually computed with implicit infinite length registers. The algorithms for the division and the square root operations are a little different since the remainder gives enough information to fix the sticky bit.

Many additional functions not required by the IEEE standard are implemented on existing chips. Floating point units are able to perform compound operations (multiply and add—IBM RS6000), or compute transcendental functions (sine, cosine, logarithm, exponential...). As a property, the correct rounding is strongly connected to the ability to carry on operation for additional precision. Moreover, the process must be able to deduce the extended sign (strictly positive, null or strictly negative) of the sticky bit.

For transcendental functions this methodology does not generalize. It is not possible to compute the result exactly in bounded time since its floating point representation is non-finite and there is no known algorithm to get the extended sign of a truncated tail portion. The case is fundamentally different for the accumulation process, but we are left to a comparable situation: the result can be computed exactly in finite time. However, this process is often considered too expensive.

Moreover, the user does not have the possibility of efficiently simulating this process. The only extensions now available to the user that is looking for some more accurate results is to extend the size of the mantissas invloved. This race for precision is not acceptable because it is very expensive regarding time if it is implemented in software or regarding silicon area if some circuit is available. We should turn to alternative methods as presented for vector operation in [1] and adapt our algorithm to the notions advocated.

Previous work has shown that for many useful functions, the IEEE roundings cannot be implemented in bounded computation time: there is no known algorithm able to compute the transcendental functions with the correct rounding in any situation with bounded execution time. Research has not led to any accepted standard for transcendental functions or for vector primitive operations. Some work [5] focuses on monotonous or piece-wise monotonous function behavior—their representation should also be monotonous.

The next theorem explains why it is not possible to round an interval (or equivalently a function) known with an uncertainty of arbitrary extended sign.

Theorem 1. *For any arbitrarily small non-null error interval $I = [\alpha, \beta]$ there is a real number x such that $x + \alpha$ and $x + \beta$ round to different values in the active rounding mode.*

Proof. Without loss of generality, we can suppose that $\alpha < 0 < \beta$. Otherwise, we use $x' = x + \frac{\alpha+\beta}{2}$, $\alpha' = \alpha - \frac{\alpha+\beta}{2}$, and $\beta' = \beta - \frac{\alpha+\beta}{2}$.

If the active rounding mode is Round to Nearest, the value $x = 1 + 1^+$ is the median of the interval $[2, 2^+]$, thus it satisfies $x - 2 = 2^+ - x = \frac{1}{2}ulp(x)$.

Hence $x + \alpha$ rounds to 2 and $x + \beta$ rounds to 2^+. Identically, if the user decides to work with directed rounding, the value $x = 1$ satisfies $x + \alpha < 1$ and $x + \beta > 1$. □

From the result of this Theorem and the algorithms used to compute transcendental functions, it is now possible to deduce that among the functions usually defined in floating point units, all the transcendental functions may lead to results different from the rounded infinitely precise result.

3 Faithful rounding

We have already utilized for the standard infinitely precise rounding mechanisms the initials Rx: RN (Round to Nearest), RU (Round Up towards $+\infty$), RD (Round Down towards $-\infty$), and RZ (Round towards Zero).

Standard Rounding Mode	RU	RD	RZ	RN
Corresponding Faithful Mode	RUU	RDD	RZZ	RND or RNU

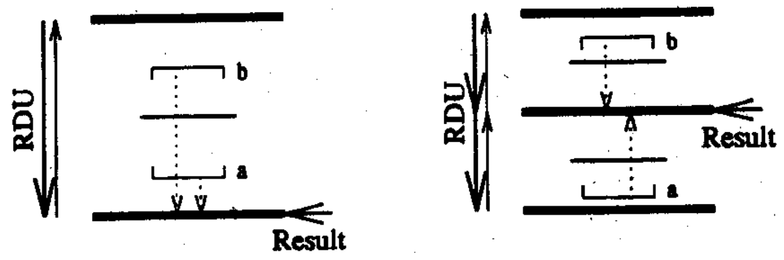
Table 1: Faithful rounding modes

The notation Rxy, where x and y may be one of the letters N, D, U, or Z, is proposed to extend these conventions to the faithful rounding of intervals. We should eventually propose five "standard" roundings RUU, RDD, RZZ, RND, and RNU to correspond to the existing modes RU, RD, RZ, and RN for standard arithmetic operations (see Table 1). We propose also the creation of two flags: Correct Rounding (CR) and Faithful Rounding (FR). If the interval can be correctly rounded the flag *CR* should be active and the correct rounding mode Rx induced from the faithful rounding mode Rxy returns the same result as the faithful rounding mode Rxy. Otherwise, if the interval can be faithfully rounded, only the *FR* flag is active, and the result of the faithful rounding Rxy is returned. If even faithful rounding is not possible, the unit should return a result yet to be specified and keep both flags inactive.

The non deterministic faithful rounding presents with a $\frac{1}{2}$ ulp requirement and a 2 ulps guarantee a very loose rounding, to be used as a last choice.

$$\mathrm{RUD}([a,b]) = \begin{cases} \mathrm{RU}(a) \text{ if } \mathrm{RD}(b) \leq \mathrm{RU}(a) \\ \text{not defined otherwise} \end{cases}$$

$$\mathrm{RDU}([a,b]) = \begin{cases} \mathrm{RD}(b) \text{ if } \mathrm{RU}(a) \geq \mathrm{RD}(b) \\ \text{not defined otherwise} \end{cases}$$



Rounding Down-Up Mode

$$v = \mathrm{RUD}([a,b]) \begin{cases} \Rightarrow & [a,b] \subset [v^-, v^+] & v^+ - v^- \leq 2ulp(v) \\ \Leftarrow & b - a \leq \frac{1}{2} \times ulp(x) & \exists x \in [a,b] \end{cases}$$

$$v = \mathrm{RDU}([a,b]) \begin{cases} \Rightarrow & [a,b] \subset [v^-, v^+] & v^+ - v^- \leq 2ulp(v) \\ \Leftarrow & b - a \leq \frac{1}{2} \times ulp(x) & \exists x \in [a,b] \end{cases}$$

Figure 3: Non deterministic faithful rounding

3.1 Non deterministic faithful rounding

All the rounding specifications presented in this section are deterministic in the way that a given interval rounds to either one precisely defined value or the interval cannot be rounded. This is the same way the IEEE standard infinitely precise rounding is deterministic. The so-called non determinism emphasizes the fact that the user cannot deduce that any value in the interval was either Rounds Down to the result or Rounds Up to the result. The difference between these two modes rely on the preference given when the interval can effectively be rounded following the IEEE standard.

Definition 4. *The interval $[a, b]$ Rounds Up-Down to $v = RU(a)$ if any value of the interval either Rounds Down to v or Rounds Up to v. Symmetrically the interval $[a, b]$ Rounds Down-Up to $v = RD(b)$ if any value of the interval either Rounds Up to v or Rounds Down to v.*

The user does not know which mode is used (Round Down towards $-\infty$ or Round Up towards $+\infty$) to round the interval, in fact different rounding modes can be used for the two parts of the interval. Given a rounded value the user can only present a 2-ulp guarantee on the initial result. An irregularity is introduced by the values were the function *ulp* is not continuous; in this case, the interval is only $\frac{3}{2}$ ulp large.

Theorem 2. *Given a result v rounded with the non deterministic faithful rounding modes RUD or RDU, the initial interval $[a, b]$ is known to be included into the interval $[v^-, v^+]$ of length at most $2 \times ulp(v)$. Conversely an interval $[a, b]$ of length less than $\frac{1}{2}ulp(x)$ (x can be arbitrarily chosen in the interval $[a, b]$) can be rounded using either one of the non deterministic faithful rounding modes RUD or RDU.*

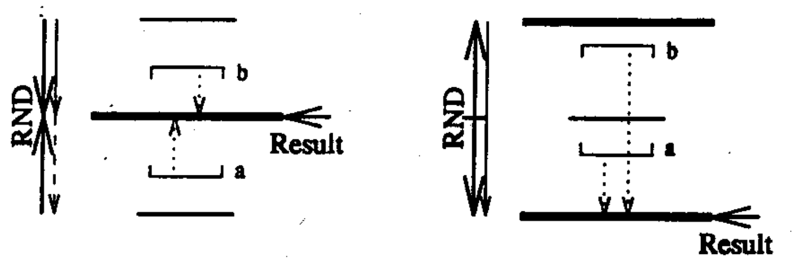
We have summed up in Figure 3 the properties of the two non deterministic rounding modes, RUD and RDU. With the definition of these modes, we have included a figure of the rounding of some floating point intervals. The real axis is projected vertically and thick horizontal bars cross the axis on representable floating point values. Some of the midpoints are represented for the RN mode. Both active rounding modes are represented of the left. The rounded result of the drawn interval is indicated by an arrow. We outline the properties of the rounding modes in the last statement: we present the consequence of $[a, b]$ being rounded to v and a sufficient condition to round an arbitrary interval $[a, b]$. We could have applied much stronger properties whenever $[a, b]$ is far enough from any power of 2, yet we have derived here general properties that could be implemented easily with no further tests.

Comments on the proof. The proof is straight forward. However, care must be given to details not overlooking any. The case where $v = \pm 2^k$ often leads to the loss of one bit meaning a factor of $\frac{1}{2}$ in precision.

3.2 Faithful rounding

$$\mathrm{RNU}([a,b]) = \begin{cases} \mathrm{RN}(a) & \text{if } \mathrm{RN}(a) = \mathrm{RN}(b) \\ \mathrm{RD}(a) & \text{if } \mathrm{RN}(a) \neq \mathrm{RN}(b) \text{ but } \mathrm{RD}(a) = \mathrm{RD}(b) \\ \text{not defined otherwise} \end{cases}$$

$$\mathrm{RND}([a,b]) = \begin{cases} \mathrm{RN}(a) & \text{if } \mathrm{RN}(a) = \mathrm{RN}(b) \\ \mathrm{RU}(a) & \text{if } \mathrm{RN}(a) \neq \mathrm{RN}(b) \text{ but } \mathrm{RU}(a) = \mathrm{RU}(b) \\ \text{not defined otherwise} \end{cases}$$



Rounding Nearest-Down Mode

$$v = \mathrm{RND}([a,b]) \begin{cases} \Rightarrow & \text{Infinitely precise rounding} & \text{RN or RD mode} \\ \Leftarrow & b - a \leq \frac{1}{4} \times ulp(x) & \exists x \in [a,b] \end{cases}$$

Figure 4: Faithful rounding RNy

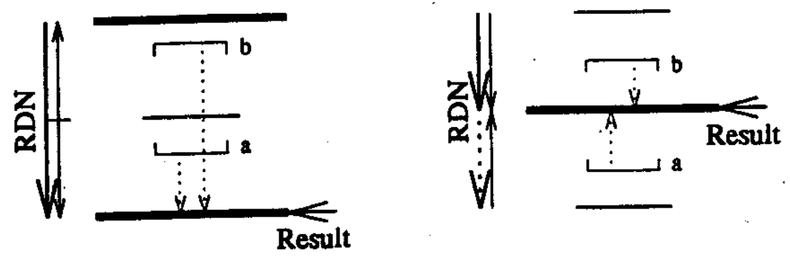
Given a smaller relative error, it is possible to deterministically define which rounding mode is used for the interval in a different set of rounding modes. The rounding modes used are Rounding Nearest-Up (RNU), Rounding Nearest-Down (RND), Rounding Up-Nearest (RUN) and Rounding Down-Nearest (RDN).

Definition 5. *The interval $[a,b]$ Rounds Nearest-Up to v if it Rounds to Nearest to v or else it Rounds Up to v. It Rounds Nearest-Down to v if it Rounds to Nearest to v or else it Rounds Down to v.*

If the interval cannot be rounded with the primary mode, Round to Nearest, the secondary mode is used to round the interval Up towards $+\infty$

$$\mathrm{RUN}([a,b]) = \begin{cases} \mathrm{RD}(a) & \text{if } \mathrm{RD}(a) = \mathrm{RD}(b) \\ \mathrm{RN}(a) & \text{if } \mathrm{RD}(a) \neq \mathrm{RD}(b) \text{ but } \mathrm{RN}(a) = \mathrm{RN}(b) \\ \text{not defined otherwise} \end{cases}$$

$$\mathrm{RDN}([a,b]) = \begin{cases} \mathrm{RU}(a) & \text{if } \mathrm{RU}(a) = \mathrm{RU}(b) \\ \mathrm{RN}(a) & \text{if } \mathrm{RU}(a) \neq \mathrm{RU}(b) \text{ but } \mathrm{RN}(a) = \mathrm{RN}(b) \\ \text{not defined otherwise} \end{cases}$$



Rounding Down-Nearest Mode

$$v = \mathrm{RDN}([a,b]) \begin{cases} \Rightarrow & \text{Infinitely precise rounding} & \text{RD or RN mode} \\ \Leftarrow & b - a \leq \frac{1}{4} \times ulp(x) & \exists x \in [a,b] \end{cases}$$

Figure 5: Faithful rounding RxN

or Down toward $-\infty$. If this is possible, this is the value returned by the floating point unit. On a process that guarantees faithful rounding (RNU or RND), the user is sure to get the correct rounding of the infinitely precise result in the primary mode (RN) or the secondary mode (RU or RD). Although the user cannot *a priori* predetermine which mode will be used, the user has *a posteriori* information on the mode used by just checking a flag which may be set by the computer. If the correct flag (CR) is active, the primary mode was used, otherwise, it was the secondary mode.

Definition 6. *The interval $[a, b]$ Rounds Up-Nearest to v if it Rounds Up to v or else it Rounds to Nearest to v. It Rounds Down-Nearest to v if it Rounds Down to v or else it Rounds to Nearest to v.*

There are many similarities between these four modes which are presented Figures 4, 5. RNU and RND on one part and RUN and RDN on the other part are very close to each other. Moreover, the situations that are faithful non correct roundings for the RND rounding mode are correct roundings for the RDN rounding mode, and vice versa.

The faithful rounding is not a loss in the uncertainty control of the function since the result is always the correct rounding of the infinitely precise value, but the users cannot impose the rounding mode used. Although the two active modes are set for the rounding, the user can only trap on the faithful rounding flag and the correct rounding flag to detect which one was used.

Theorem 3. *The faithful rounding mechanism guarantees that the value is the correct rounding of the infinitely precise result using the primary or the secondary mode.* Vice versa *an interval $[a, b]$ of maximum length $\frac{1}{4} \times ulp(x)$ can be faithfully rounded with any of the specified faithful modes.*

3.3 Directed faithful rounding

In the previous section, we have presented faithful rounding specifications that ensure the rounding of an interval to be the correct rounding of the infinitely precise mathematical interval with one predefined mode; the mode is not fully specified by the user, yet the process has to pick the rounding mode in a set of two modes specified by the user.

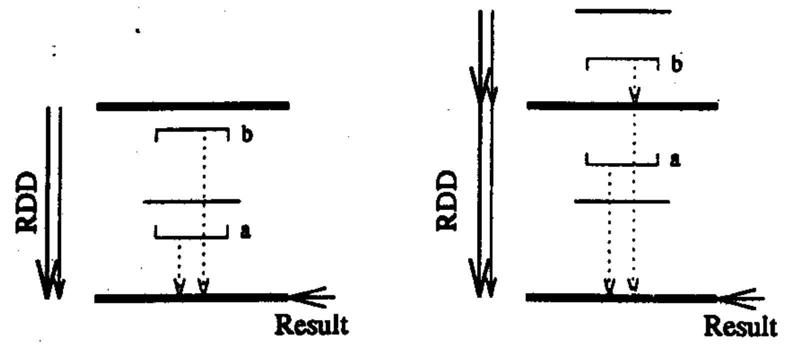
The directed rounding modes were introduced in the IEEE standard to allow interval arithmetic implementation at the user level. This is to guarantee that allowing error during the rounding process cannot make the user loose the knowledge that the result is a directed approximation of the infinitely precise result.

The major drawback of faithful rounding: it is not compatible with directed approximation. Still keeping the same uncertainty, we present Figure 6 the Round Up-Up and Round Down-Down mode from the following definition.

Definition 7. *The interval $[a, b]$ Rounds Up-Up to v if it Rounds Up to v or else it Rounds to Nearest to v^-. It Rounds Down-Down to v if it Rounds Down to v or else it Rounds to Nearest to v^+.*

$$\mathrm{RUU}([a,b]) = \begin{cases} \mathrm{RD}(a) \ \text{ if } \ \mathrm{RD}(a) = \mathrm{RD}(b) \\ \mathrm{RN}(a)^- \ \text{ if } \ \mathrm{RD}(a) \neq \mathrm{RD}(b) \ \text{ but } \ \mathrm{RN}(a) = \mathrm{RN}(b) \\ \text{not defined otherwise} \end{cases}$$

$$\mathrm{RDD}([a,b]) = \begin{cases} \mathrm{RU}(a) \ \text{ if } \ \mathrm{RU}(a) = \mathrm{RU}(b) \\ \mathrm{RN}(a)^+ \ \text{ if } \ \mathrm{RU}(a) \neq \mathrm{RU}(b) \ \text{ but } \ \mathrm{RN}(a) = \mathrm{RN}(b) \\ \text{not defined otherwise} \end{cases}$$



Rounding Down-Down Mode

$$v = \mathrm{RDD}([a,b]) \begin{cases} \Rightarrow \ \text{Infinitely precise rounding} & \text{RD or RN mode} \\ \Leftarrow \ b - a \leq \frac{1}{4} \times ulp(x) & \exists x \in [a,b] \end{cases}$$

Figure 6: Directed faithful rounding

The RZZ (Rounding Zero-Zero) mode can be obtained from the RDD and the RUU modes. Actually, any RxZ or RZx mode can be obtained from the associated Round Up towards $+\infty$ or Round Down towards $-\infty$ rounding modes.

4 Conclusion

Because the initial set of functions implemented through the IEEE standard was relatively small, the need for interval arithmetic and interval rounding were totally under-minded compared to the advantage of correct rounding and reproducibility notions.

Now that the community is well aware of the importance of reproducible computation, we need to back up from this paradigm. The faithful rounding is a first step toward non infinitely precise rounding. It is the interval rounding mechanism needed to extend the IEEE standard; this extension on interval rounding can be efficiently applied to transcendental function and vector operation rounding. Moreover, this mechanism can be implemented with no loss of precision for the user in comparison with infinitely precise rounding.

This mechanism may be added to existing hardware only with a few modifications. Transcendental functions are already correctly defined regarding the faithful rounding mechanism. To allow the user to control efficiently the error, a new flag is to be included that signals non infinitely precise rounding, i.e. faithful rounding. It is possible to associate with any IEEE specified rounding mode Rx a faithful rounding mode Rxy. Later, the user may have the hardware possibility to chose any couple Rxy.

References

[1] Bohlender, G. *What do we need beyond IEEE arithmetic?* In: “Computer Arithmetic and Self Validating Numerical Methods”, Academic Press, 1990, pp. 1–32.

[2] Braune, K. *Standard functions for real and complex point and interval arguments with dynamic accuracy.* Scientific computation with automatic result verification, Suppl. 6, Springer-Verlag, 1988.

[3] Daumas, M. and Matula, D. W. *Design of a fast reliable dot product operation.* In: “11th IEEE Symposium on Computer Arithmetic”, 1993.

[4] *IEEE standard for binary floating-point arithmetic.* ANSI/IEEE Std 754-1985. The Institute of Electrical Engineering and Electronics Engineers, New York, 1985.

[5] Ferguson Jr., W. E. and Brightman, T. *Accurate and monotone approximations of some transcendental functions.* In: “10th IEEE Symposium on Computer Arithmetic”, 1991, pp. 237–244.

[6] Kirchner, R. and Kulisch, U. *Arithmetic for vector processors.* In: “8th IEEE Symposium on Computer Arithmetic”, 1987, pp. 256–269.

[7] Kulisch, U. and Miranker, W. L. *Computer arithmetic in theory and practice*. Academic press, 1981.

Received: November 11, 1993
Revised version: September 4, 1994

D. W. Matula
Southern Methodist University
Dallas, TX 75275
Email: `Matula@Seas.Smu.Edu`

M. Daumas
École Normale Supérieure de Lyon
Lyon, France 69364
Email: `Marc.Daumas@Lip.Ens-Lyon.Fr`