Computing predecessor and successor in rounding to nearest

Siegfried M. Rump · **Paul Zimmermann** ·
Sylvie Boldo · **Guillaume Melquiond**

Abstract We give simple and efficient methods to compute and/or estimate the predecessor and successor of a floating-point number using only floating-point operations in rounding to nearest. This may be used to simulate interval operations, in which case the quality in terms of the diameter of the result is significantly improved compared to existing approaches.

Keywords Floating-point arithmetic · rounding to nearest, predecessor · successor · directed rounding

Mathematics Subject Classification (2000) MSC 68-04 · MSC 68N30

1 Introduction and notation

Throughout the paper we assume a floating-point arithmetic according to the IEEE 754 and IEEE 854 arithmetic standards [3,4] with rounding to nearest. Denote the set of

S. M. Rump
Institute for Reliable Computing, Hamburg University of Technology,
Schwarzenbergstraße 95, 21071 Hamburg, Germany,
and Visiting Professor at Waseda University, Faculty of Science and Engineering,
3–4–1 Okubo, Shinjuku-ku, Tokyo 169–8555, Japan
E-mail: rump@tu-harburg.de

P. Zimmermann
Centre de Recherche INRIA Nancy - Grand Est, Équipe-projet CACAO, Bâtiment A,
615 rue du jardin botanique, F-54600 Villers-lès-Nancy, France
E-mail: Paul.Zimmermann@loria.fr

S. Boldo
INRIA Saclay - Île-de-France, Parc Orsay Université - ZAC des Vignes,
4 rue Jacques Monod - Bâtiment N, F-91893 Orsay Cedex, France
E-mail: Sylvie.Boldo@inria.fr

G. Melquiond
Centre de recherche commun INRIA - Microsoft Research,
28 rue Jean Rostand, F-91893 Orsay Cedex, France
E-mail: Guillaume.Melquiond@inria.fr

(single or double precision) floating-point numbers by $\mathbb{F}$, including $-\infty$ and $+\infty$, and let $\mathrm{fl} : \mathbb{R} \to \mathbb{F}$ denote rounding to nearest according to IEEE 754. This includes especially the default rounding mode, to nearest "ties to even" and the rounding to nearest "ties to away" (away from zero) defined in the revision of the IEEE 754 standard, IEEE 754-2008, that we follow in this paper. Define for single and double precision $\mathbf{u}$, the relative rounding error unit, and η, the smallest positive subnormal floating-point number:

	single precision (binary32)	double precision (binary64)
$\mathbf{u}$	2^{-24}	2^{-53}
η	2^{-149}	2^{-1074}

Let β be the radix used in this floating-point format. We require β to be even and greater than one. This includes especially 2, 10, and their powers. For a format of precision p "digits" in radix β, we define $\mathbf{u}$ as half the distance between 1 and its successor, i.e.:

$$\mathbf{u} = \frac{\beta^{1-p}}{2}.$$

Then $\mathbf{u}$ and η satisfy:

$$\forall \circ \in \{+, -, \times, \div\} \; \forall \, a, b \in \mathbb{F} \setminus \{\pm\infty\} : \quad \mathrm{fl}(a \circ b) = (a \circ b)(1 + \lambda) + \mu \qquad (1.1)$$

with $|\lambda| \le \mathbf{u}$ and $|\mu| \le \eta/2$ and at least one of λ, μ is zero, provided $\mathrm{fl}(a \circ b)$ is finite. Note that for addition and subtraction μ is always zero. An important property of the rounding is the monotonicity, that is

$$\forall x, y \in \mathbb{R} : \quad x \le y \;\Rightarrow\; \mathrm{fl}(x) \le \mathrm{fl}(y). \qquad (1.2)$$

In particular, this implies

$$x \in \mathbb{R}, \; f \in \mathbb{F}, \; x \le f \quad \Rightarrow \; \mathrm{fl}(x) \le f, \qquad (1.3)$$

which means that rounding cannot "jump" over a floating-point number. The floating-point predecessor and successor of a *real* number $x \in \mathbb{R}$ are defined by

$$\mathrm{pred}(x) := \max\{f \in \mathbb{F} : f < x\} \quad \text{and} \quad \mathrm{succ}(x) := \min\{f \in \mathbb{F} : x < f\},$$

respectively, where, according to IEEE 754, $\pm\infty$ are considered to be floating-point numbers. For example, $\mathrm{succ}(1) = 1 + 2\mathbf{u}$. Using (1.1) it is not difficult to see that for finite $0 \le c \in \mathbb{F}$, as long as $\mathrm{succ}(c)$ is finite:

$$\min(c(1 - 2\mathbf{u}), c - \eta) \le \mathrm{pred}(c) \quad \text{and} \quad \mathrm{succ}(c) \le \max(c(1 + 2\mathbf{u}), c + \eta), \quad (1.4)$$

and similarly for $c < 0$. For $a, b \in \mathbb{F}$ and finite $c := \mathrm{fl}(a \circ b)$ the monotonicity (1.2) implies

$$a \circ b \in [c_1, c_2] \text{ where } c_1 := \mathrm{fl}(\mathrm{fl}(c - \mathrm{fl}(2\mathbf{u}|c|)) - \eta)$$
$$\text{and } c_2 := \mathrm{fl}(\mathrm{fl}(c + \mathrm{fl}(2\mathbf{u}|c|)) + \eta). \qquad (1.5)$$

(Note that similarly to (1.1), the above remains true if $a \circ b$ is replaced by any real y, for example $y = \sin x$, as long as c is the correct rounding of y.) This is the usual

basis of interval libraries to emulate directed roundings using only rounding to nearest (see, for example, [5]). It is disadvantageous because for $1.5 \cdot 2^k \leq |a \circ b| < 2^{k+1}$ the interval $[c_1, c_2]$ is twice as wide as it needed to be, i.e., 4 *ulps* (units in the last place) instead of 2 ulps.

The IEEE 754-2008 standard [3] requires availability of a function `nextUp` — and similarly `nextDown` — where `nextUp`(a) is the least floating-point number in the format of a that compares greater than a.

The function `nextUp` thus computes the successor of a floating-point number. This just amounts to adding η if directed roundings, as requested by IEEE 754, are available. However, performing a computation using directed roundings may not be supported by the programming language in use, or it may depend on a rounding mode change, which usually involves a flush of the processor pipeline. So the function benefits from using the default rounding mode only, which is to nearest with ties to even as requested for binary formats, and recommended for decimal formats by IEEE 754-2008.

In [2] a corresponding algorithm is given by splitting the floating-point number into two parts, treating the second part as an integer and adding/subtracting 1 regarding possible carry. The algorithm does the job, but is slow. In [1] a corresponding routine is given assuming that the exponent range is unlimited and that a fused multiply and accumulate instruction is available, that is $a \cdot b + c$ with only one rounding.

The contributions of this paper are the following. First we describe a simple and efficient routine (Algorithm 1) for any radix to compute an interval $[c_1, c_2]$ with $c_1, c_2 \in \mathbb{F}$ containing $a \circ b$ for all $a, b \in \mathbb{F}$ and $\circ \in \{+, -, \times, \div\}$ provided $\mathrm{fl}(a \circ b)$ is finite (Theorem 2.1). We then focus on binary arithmetic (radix 2), and prove that if $a \circ b$ is not a floating-point number, the result of Algorithm 1 is always best possible[1] except a small range near underflow (Theorem 2.2). Finally we describe a slightly more complicated variant (Algorithm 2) that always returns the best possible interval in binary arithmetic, and compare its efficiency with the C99 `nextafter` implementation.

2 The results

We use the "unit in the first place" $\mathrm{ufp}(x)$ defined for $x \in \mathbb{R}$ by

$$\mathrm{ufp}(0) := 0 \quad \text{and} \quad \mathrm{ufp}(x) := \beta^{\lfloor \log_\beta |x| \rfloor} \text{ for } x \neq 0.$$

It denotes the weight of the most significant digit in the representation of x. Then

$$\forall\, 0 \neq x \in \mathbb{R}: \qquad \mathrm{ufp}(x) \leq |x| < \beta\, \mathrm{ufp}(x). \tag{2.1}$$

This concept introduced by the first author proved to be useful in delicate analyses of the accuracy of floating-point algorithms [6]. The definition is independent of some

---

[1] Since we don't know if $c := \mathrm{fl}(a \circ b)$ is smaller or larger than $a \circ b$, the best possible interval is $[\mathrm{pred}(c), \mathrm{succ}(c)]$ of 2 ulps, unless c is a power of the radix.

floating-point format. Define

$$\overline{\mathbb{U}} := \{ f \in \mathbb{F} \ : \ |f| < \frac{\beta}{2} \mathbf{u}^{-1} \eta \}. \tag{2.2}$$

For example, in IEEE 754-2008 `binary64` format (double precision), $\overline{\mathbb{U}} = \{ f \in \mathbb{F} :$ $|f| < 2^{-1021} \}$. Note that $\frac{1}{2} \mathbf{u}^{-1} \eta$ is the smallest positive *normal* floating-point number. For positive $c \in \mathbb{F}$ such that $\mathrm{succ}(c)$ is finite the following properties are easily verified (see also [6]):

if $c \in \overline{\mathbb{U}}$: $\mathrm{pred}(c) = c - \eta, \quad \mathrm{succ}(c) = c + \eta,$ (2.3)

if $c \notin \overline{\mathbb{U}}, c \neq \beta^k$: $\mathrm{pred}(c) = c - 2\mathbf{u}\,\mathrm{ufp}(c), \quad \mathrm{succ}(c) = c + 2\mathbf{u}\,\mathrm{ufp}(c),$ (2.4)

if $c \notin \overline{\mathbb{U}}, c = \beta^k$: $\mathrm{pred}(c) = c - \frac{2}{\beta}\mathbf{u}\,\mathrm{ufp}(c), \quad \mathrm{succ}(c) = c + 2\mathbf{u}\,\mathrm{ufp}(c).$ (2.5)

Moreover, define for $c \in \mathbb{F} \setminus \{\pm\infty\}$ with $\mathrm{pred}(c)$ and $\mathrm{succ}(c)$ finite:

$$\mathscr{M}^-(c) := \frac{1}{2}\big(\mathrm{pred}(c) + c\big) \quad \text{and} \quad \mathscr{M}^+(c) := \frac{1}{2}\big(c + \mathrm{succ}(c)\big). \tag{2.6}$$

It follows for $c \in \mathbb{F}$, $x \in \mathbb{R}$,

$$\begin{aligned} x < \mathscr{M}^-(c) &\Rightarrow \mathrm{fl}(x) \leq \mathrm{pred}(c) \quad \text{and} \quad \mathscr{M}^+(c) < x \Rightarrow \mathrm{succ}(c) \leq \mathrm{fl}(x) \\ \mathscr{M}^-(c) &< x \Rightarrow c \leq \mathrm{fl}(x) \quad \text{and} \quad x < \mathscr{M}^+(c) \Rightarrow \mathrm{fl}(x) \leq c. \end{aligned} \tag{2.7}$$

2.1 General radix β

For $c \in \mathbb{F}$ consider the following algorithm.

Algorithm 1 *Bounds for predecessor and successor of finite $c \in \mathbb{F}$ in rounding to nearest*

$$e = \mathrm{fl}(\mathrm{fl}(\phi|c|) + \eta) \qquad \% \ \phi = \mathbf{u}\left(1 + \frac{4}{\beta}\mathbf{u}\right) = \mathrm{succ}(\mathbf{u})$$
$$\mathtt{cinf} = \mathrm{fl}(c - e)$$
$$\mathtt{csup} = \mathrm{fl}(c + e)$$

Note that we need a reasonable floating-point format to ensure that ϕ is a floating-point number. More precisely, we need $\eta \leq 2\mathbf{u}^2$ (which in turn implies $\eta \leq \frac{4}{\beta}\mathbf{u}^2$, or equivalently $2\mathbf{u} \notin \overline{\mathbb{U}}$, because η is a power of the radix). This does happen on any real-life floating-point format (all IEEE formats included). If not, then the last bit of ϕ is lost due to underflow.

Theorems 2.1 and 2.2 have been formally checked using the Coq automatic proof checker and a previous formalization of floating-point numbers [7, 8]. The proof file is available at `http://lipforge.ens-lyon.fr/www/pff/AlgoPredSucc.html`.

Theorem 2.1 *Let finite $c \in \mathbb{F}$ be given, and assume $\mathbf{u} \leq \frac{\beta^{-2}}{2}$. Let* `cinf` *and* `csup` *be the quantities computed by Algorithm 1. Then*

$$\mathtt{cinf} \leq \mathrm{pred}(c) \qquad \text{and} \qquad \mathrm{succ}(c) \leq \mathtt{csup}. \tag{2.8}$$

REMARK. The technical assumption $\mathbf{u} \leq \frac{\beta^{-2}}{2}$ is satisfied by any practical implementation of floating-point arithmetic: it means that we have at least 3 digits of precision.

PROOF OF THEOREM 2.1. Since $\mathbb{F} = -\mathbb{F}$ and $\mathrm{fl}(-x) = -\mathrm{fl}(x)$ for $x \in \mathbb{R}$, we may assume without loss of generality $c \geq 0$. If $c \in \overline{\overline{\mathbb{U}}}$, then $e \geq \eta$ and (2.3) imply (2.8).

It remains to prove (2.8) for $0 \leq c \notin \overline{\overline{\mathbb{U}}}$. One verifies (2.8) for c being the largest positive floating-point number, hence we assume without loss of generality that $\mathrm{succ}(c)$ is finite.

We first prove that for e as computed in Algorithm 1 we have

$$e > \mathbf{u}\,\mathrm{ufp}(c). \tag{2.9}$$

First note that $0 \leq c \notin \overline{\overline{\mathbb{U}}}$ means $c \geq \frac{\beta}{2}\mathbf{u}^{-1}\eta$. Also note that $\mathbf{u}\,\mathrm{ufp}(c) \in \mathbb{F}$. By (1.2), (2.1), (2.4) and (2.5),

$$c' := \mathrm{fl}(\phi c) = \mathrm{fl}(\mathrm{succ}(\mathbf{u})\,c) \geq \mathrm{fl}(\mathrm{succ}(\mathbf{u})\,\mathrm{ufp}(c)). \tag{2.10}$$

If $\mathrm{ufp}(c)\,\mathrm{succ}(\mathbf{u})$ is normal, i.e., $\mathrm{ufp}(c)\,\mathrm{succ}(\mathbf{u}) \geq \frac{1}{2}\mathbf{u}^{-1}\eta$, then

$$e = \mathrm{fl}(c' + \eta) \geq c' \geq \mathrm{fl}(\mathrm{ufp}(c)\,\mathrm{succ}(\mathbf{u})) = \mathrm{ufp}(c)\,\mathrm{succ}(\mathbf{u}) > \mathbf{u}\,\mathrm{ufp}(c),$$

which proves (2.9). If $\mathrm{ufp}(c)\,\mathrm{succ}(\mathbf{u}) \in \overline{\overline{\mathbb{U}}}$, i.e., $\mathrm{ufp}(c)\,\mathrm{succ}(\mathbf{u}) < \frac{\beta}{2}\mathbf{u}^{-1}\eta$, then (2.10), (1.2) and (2.1) imply

$$c' \geq \mathrm{fl}(\mathrm{ufp}(c)\,\mathrm{succ}(\mathbf{u})) \geq \mathrm{fl}(\mathbf{u}\,\mathrm{ufp}(c)) = \mathbf{u}\,\mathrm{ufp}(c). \tag{2.11}$$

We split the remaining in two cases. First, suppose $c' = \mathrm{fl}(\phi c) < \frac{\beta}{2}\mathbf{u}^{-1}\eta$. Then $c' \in \overline{\overline{\mathbb{U}}}$, and (2.3), (2.11) yield (2.9) as $e = \mathrm{fl}(c' + \eta) = c' + \eta > \mathbf{u}\,\mathrm{ufp}(c)$.

The second and last case corresponds to $\mathrm{ufp}(c)\,\mathrm{succ}(\mathbf{u}) < \frac{\beta}{2}\mathbf{u}^{-1}\eta$ and $c' \geq \frac{\beta}{2}\mathbf{u}^{-1}\eta$. Then

$$c' = \mathrm{fl}(\phi c) \geq \frac{\beta}{2}\mathbf{u}^{-1}\eta > \mathrm{ufp}(c)\,\mathrm{succ}(\mathbf{u}) > \mathbf{u}\,\mathrm{ufp}(c),$$

and finally $e = \mathrm{fl}(c' + \eta) \geq c' > \mathbf{u}\,\mathrm{ufp}(c)$, so that the proof of (2.9) is finished.

We now prove Theorem 2.1 for $c \notin \overline{\overline{\mathbb{U}}}$. By (2.4) and (2.5) we know

$$c - \mathbf{u}\,\mathrm{ufp}(c) \leq \mathscr{M}^-(c) \quad \text{and} \quad \mathscr{M}^+(c) = c + \mathbf{u}\,\mathrm{ufp}(c),$$

so (2.9) and (2.7) yield

$$c - e < \mathscr{M}^-(c) \quad \text{and} \quad \mathscr{M}^+(c) < c + e \tag{2.12}$$

and prove (2.8). The theorem is proved. $\qquad\square$

Note that the inequalities are far from sharp in a radix different from 2. The worst case in radix 10 corresponds to computing the predecessor of a power of the radix. For example, let $c = 1$ as in Figure 2.1, then $e = \phi$ and $\mathtt{cinf} = \mathrm{fl}(1 - \phi) = 1 - \mathbf{u}$ while $\mathrm{pred}(c) = 1 - \frac{\mathbf{u}}{5}$. The computed $\mathtt{cinf}$ is equal to $\mathrm{pred}(\mathrm{pred}(\mathrm{pred}(\mathrm{pred}(\mathrm{pred}(c)))))$.

REMARK. Theorem 2.1 remains valid if e is computed with a fused multiply-add (fma) in Algorithm 1. Let $e' = \mathrm{fl}(\phi|c| + \eta)$ the value computed with a fma, which might differ from $e = \mathrm{fl}(\mathrm{fl}(\phi|c|) + \eta)$. Either $\mathbf{u}\,\mathrm{succ}(c)$ is normal, then $e' \geq \mathrm{fl}(\phi|c|) = c' > \mathbf{u}\,\mathrm{ufp}(c)$ as demonstrated in the proof of Theorem 2.1; or $\mathbf{u}\,\mathrm{succ}(c)$ is subnormal, then $e' \geq \mathrm{fl}(\mathbf{u}c + \eta) \geq \mathrm{fl}(\mathbf{u}\,\mathrm{ufp}(c) + \eta) = \mathbf{u}\,\mathrm{ufp}(c) + \eta$ since $\mathbf{u}\,\mathrm{ufp}(c) \in \mathbb{F}$ and is subnormal in that case. Thus in both cases $e' > \mathbf{u}\,\mathrm{ufp}(c)$ remains valid.
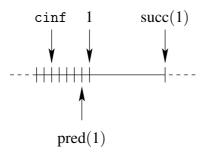
Fig. 2.1 Worst case for computing the predecessor in radix 10.

2.2 Binary arithmetic

In this section we assume $\beta = 2$, thus $\mathbf{u}$ is a power of two, and $\phi = \mathbf{u}(1 + 2\mathbf{u}) = \text{succ}(\mathbf{u})$. For a general radix we proved that the inequalities in (2.8) are satisfied for all finite floating-point numbers. Here we show that equality holds for binary arithmetic except a small range near the underflow threshold.

Theorem 2.2 *Assume binary arithmetic, i.e., $\beta = 2$, with at least 4 bits of precision, i.e., $\mathbf{u} \leq \frac{1}{16}$. Assume the rounding is "ties to even", or "ties to away" as defined by IEEE 754-2008. Then for all finite $c \in \mathbb{F}$ with $|c| \notin [\frac{1}{2}, 2]\mathbf{u}^{-1}\eta$ the quantities* cinf *and* csup *computed by Algorithm 1 satisfy*

$$\text{cinf} = \text{pred}(c) \qquad \text{and} \qquad \text{succ}(c) = \text{csup}. \tag{2.13}$$

REMARK 1.   For IEEE 754 `binary64` format the excluded range $[\frac{1}{2}, 2]\mathbf{u}^{-1}\eta$ is $[2^{-1022}, 2^{-1020}]$, which corresponds to two binades around the subnormal threshold. In this range Algorithm 1 returns $\text{cinf} = \text{pred}(\text{pred}(c))$ and $\text{csup} = \text{succ}(\text{succ}(c))$, i.e., it is one bit off.

REMARK 2.   It is easy to see from the proof that when $\mathbf{u} \leq \frac{1}{32}$, the assertions remain true for any tie-breaking rule in rounding to nearest.

PROOF OF THEOREM 2.2. Without loss of generality we assume $0 \leq c \in \mathbb{F}$. If $c < \frac{1}{2}\mathbf{u}^{-1}\eta$, then $c \leq \frac{1}{2}\mathbf{u}^{-1}\eta - \eta$ by (2.3), so that

$$\phi c \leq \frac{1}{2}(1 + 2\mathbf{u})\eta - \mathbf{u}(1 + 2\mathbf{u})\eta < \frac{1}{2}\eta$$

implies $\text{fl}(\phi c) = 0$. Hence $e = \eta$, and the theorem is proved for $c < \frac{1}{2}\mathbf{u}^{-1}\eta$.

It remains to prove (2.13) for $c > 2\mathbf{u}^{-1}\eta$. We first show

$$e \leq \frac{5}{2}\mathbf{u}\,\text{ufp}(c). \tag{2.14}$$

First,

$$c \leq \text{pred}(2\text{ufp}(c)) = 2(1 - \mathbf{u})\text{ufp}(c)$$

follows by (2.1) and (2.5) and also for $2\mathrm{ufp}(c)$ in the overflow range, so that

$$\phi c = \mathbf{u}(1+2\mathbf{u})c < 2\mathbf{u}(1+\mathbf{u})\mathrm{ufp}(c) =: (1+\mathbf{u})C. \qquad (2.15)$$

Since $\mathrm{ufp}(c) \geq 2\mathbf{u}^{-1}\eta$ it follows that $C = 2\mathbf{u}\,\mathrm{ufp}(c) \in \mathbb{F}$. If $C \geq \frac{1}{2}\mathbf{u}^{-1}\eta$, then

$$\phi c < (1+\mathbf{u})C = \mathcal{M}^+(C),$$

because $C = \mathrm{ufp}(C)$, and (2.7) yields $c' = \mathrm{fl}(\phi c) \leq C$. If $C < \frac{1}{2}\mathbf{u}^{-1}\eta$, then $C \leq \frac{1}{4}\mathbf{u}^{-1}\eta$ and (2.15) and $C \in \mathbb{F}$ give

$$c' = \mathrm{fl}(\phi c) \leq \mathrm{fl}(C + \frac{1}{4}\eta) = C,$$

so that $\mathrm{ufp}(c) \geq 2\mathbf{u}^{-1}\eta$ with $c' \leq C$ in both cases $C \geq \frac{1}{2}\mathbf{u}^{-1}\eta$ and $C < \frac{1}{2}\mathbf{u}^{-1}\eta$ proves

$$e = \mathrm{fl}(c' + \eta) \leq \mathrm{fl}(C + \frac{1}{2}\mathbf{u}\,\mathrm{ufp}(c)) = \mathrm{fl}(\frac{5}{2}\mathbf{u}\,\mathrm{ufp}(c)) = \frac{5}{2}\mathbf{u}\,\mathrm{ufp}(c)$$

and thus (2.14).

Now we prove the equalities in (2.13) for $c > 2\mathbf{u}^{-1}\eta$. From (2.4) and (2.14) and (2.5) we have

$$c + e \leq \mathrm{succ}(c) + \frac{1}{2}\mathbf{u}\,\mathrm{ufp}(c) < \mathcal{M}^+(\mathrm{succ}(c)),$$

since $\mathcal{M}^+(\mathrm{succ}(c)) \geq \mathrm{succ}(c) + \mathbf{u}\,\mathrm{ufp}(c)$, so that $\mathtt{csup} = \mathrm{fl}(c+e) \leq \mathrm{succ}(c)$ by (2.6). Together with Theorem 2.1, this proves the right equality in (2.13). A similar argument applies when $\mathrm{pred}(\mathrm{pred}(c)) \geq \mathrm{ufp}(c)$ and shows $\mathtt{cinf} = \mathrm{fl}(c-e) \geq \mathrm{pred}(c)$ in that case. It remains to prove the left equality in (2.13) for $\mathrm{ufp}(c) \in \{\mathrm{pred}(c), c\}$.

If $\mathrm{pred}(c) = \mathrm{ufp}(c)$, i.e., $c = \mathrm{ufp}(c)(1+2u)$, then we prove $e < \frac{5}{2}\mathbf{u}\,\mathrm{ufp}(c)$ as follows:

$$e = \mathrm{fl}(c' + \eta) \leq \mathrm{fl}(\mathbf{u}(\frac{3}{2} + 4\mathbf{u})\mathrm{ufp}(c)) \leq \mathrm{fl}(2\mathbf{u}\,\mathrm{ufp}(c)) = 2\mathbf{u}\,\mathrm{ufp}(c),$$

since we proved above that $C = 2\mathbf{u}\,\mathrm{ufp}(c)$ is in $\mathbb{F}$. Hence

$$c - e > c - \frac{5}{2}\mathbf{u}\,\mathrm{ufp}(c) = \mathrm{pred}(c) - \frac{1}{2}\mathbf{u}\,\mathrm{ufp}(c),$$

and $\mathrm{fl}(c-e) \geq \mathrm{pred}(c)$ follows.

Finally, if $c = \mathrm{ufp}(c)$, then c is a power of 2 and $c > 2\mathbf{u}^{-1}\eta$ implies $c \geq 4\mathbf{u}^{-1}\eta$. We distinguish three cases. First, assume $c \geq \frac{1}{2}\mathbf{u}^{-2}\eta$. Then $\phi c \in \mathbb{F}$ and $c' = \mathrm{fl}(\phi c) = \phi c = \mathbf{u}c + 2\mathbf{u}^2 c \leq \frac{9}{8}\mathbf{u}c$ by $\mathbf{u} \leq \frac{1}{16}$. Furthermore, $\mathbf{u}c \geq \frac{1}{2}\mathbf{u}^{-1}\eta \geq 8\eta$, so that $c' + \eta \leq \frac{9}{8}\mathbf{u}c + \frac{1}{8}\mathbf{u}c$ implies

$$e = \mathrm{fl}(c' + \eta) \leq \mathrm{fl}(\frac{5}{4}\mathbf{u}c) = \frac{5}{4}\mathbf{u}c < \frac{3}{2}\mathbf{u}c.$$

Second, assume $c < \frac{1}{4}\mathbf{u}^{-2}\eta$. Then $\mathbf{u}c \leq \phi c < \mathbf{u}c + \frac{1}{2}\eta$, which shows that $c' = \mathrm{fl}(\phi c) = \mathbf{u}c$. Hence

$$e = \mathrm{fl}(c' + \eta) \leq \mathrm{fl}(\frac{5}{4}\mathbf{u}c) = \frac{5}{4}\mathbf{u}c < \frac{3}{2}\mathbf{u}c.$$

Therefore (2.5) implies $\mathrm{pred}(c) = (1 - \mathbf{u})c$ and $\mathscr{M}^-(\mathrm{pred}(c)) = (1 - \frac{3}{2}\mathbf{u})c$, and (2.7) finishes the first and the second case.

Third, assume $c = \frac{1}{4}\mathbf{u}^{-2}\eta$. Then

$$\phi c = \mathbf{u}c + \frac{1}{2}\eta \ ,$$

and this is the only case where the tie-breaking rule is important. If the computation of $\mathrm{fl}(\phi c)$ is rounded to nearest with ties to even, then $c' = \mathrm{fl}(\phi c) = \mathbf{u}c$ and the proof of the second case holds. Let us finally assume rounding to nearest with "ties to away". Then with $\mathbf{u}c = \frac{1}{4}\mathbf{u}^{-1}\eta$:

$$c' = \mathrm{fl}(\phi c) = \mathrm{fl}(\mathbf{u}c + \frac{1}{2}\eta) = \mathbf{u}c + \eta$$

and

$$e = \mathrm{fl}(c' + \eta) = \mathbf{u}c + 2\eta \ .$$

Hence

$$\mathtt{csup} = \mathrm{fl}(c + e) = \mathrm{fl}(\mathbf{u}c(1 + \mathbf{u} + 8\mathbf{u}^2)) = \mathbf{u}c(1 + 2\mathbf{u}) = \mathrm{succ}(c) \ .$$

Finally, $\mathbf{u} \leq \frac{1}{16}$ and rounding ties to away implies

$$\mathtt{cinf} = \mathrm{fl}(c - e) = \mathrm{fl}(\mathbf{u}c(1 - \mathbf{u} - 8\mathbf{u}^2)) = \mathbf{u}c(1 - \mathbf{u}) = \mathrm{pred}(c) \ .$$

That last subcase ends the proof. □

We mention that the assumption $\mathbf{u} \leq \frac{1}{16}$ is necessary to prove the inequalities in (2.8) to be equalities. This is seen by $\mathbf{u} = \frac{1}{8}$, $\eta = \frac{1}{32}$ and $c = 1 \notin [\frac{1}{2}, 2]\mathbf{u}^{-1}\eta = [\frac{1}{8}, \frac{1}{2}]$.

Given $a, b \in \mathbb{F}$, rigorous and mostly sharp bounds for $a \circ b, \circ \in \{+, -, \times, \div\}$ can be computed by applying Algorithm 1 to $c := \mathrm{fl}(a \circ b)$. This holds for the square root as well. Although addition and subtraction cause no error if the result is in the underflow range, the extra term η cannot be omitted in the computation of e because it is needed for c slightly outside the underflow range.

Algorithm 1 computes floating-point numbers $\mathtt{cinf}$ and $\mathtt{csup}$ bounding the predecessor and successor of its input c from below and above, respectively. In binary arithmetic, although the inequalities in (2.8) are proved to be equalities outside a small range near underflow, one may wish to eliminate exceptional cases so that always equality holds in (2.8). This is achieved by the following algorithm.

Algorithm 2 *Computation of the predecessor and successor of finite $c \in \mathbb{F}$ in binary arithmetic with rounding to nearest*

$$
\begin{aligned}
&\textbf{if } |c| \geq \tfrac{1}{2}\mathbf{u}^{-2}\eta \textbf{ then} \\
&\quad e = \mathrm{fl}(\phi|c|) \qquad\qquad \% \; \phi = \mathbf{u}(1+2\mathbf{u}) = \mathrm{succ}(\mathbf{u}) \\
&\quad \texttt{cinf} = \mathrm{fl}(c-e) \\
&\quad \texttt{csup} = \mathrm{fl}(c+e) \\
&\textbf{elseif } |c| < \mathbf{u}^{-1}\eta \\
&\quad \texttt{cinf} = \mathrm{fl}(c-\eta) \\
&\quad \texttt{csup} = \mathrm{fl}(c+\eta) \\
&\textbf{else} \\
&\quad C = \mathrm{fl}(\mathbf{u}^{-1}c) \\
&\quad e = \mathrm{fl}(\phi|C|) \\
&\quad \texttt{cinf} = \mathrm{fl}(\mathrm{fl}(C-e)\cdot\mathbf{u}) \\
&\quad \texttt{csup} = \mathrm{fl}(\mathrm{fl}(C+e)\cdot\mathbf{u})
\end{aligned}
$$

Theorem 2.3 *Let finite $c \in \mathbb{F}$ be given in binary radix, and let $\texttt{cinf}, \texttt{csup} \in \mathbb{F}$ be the quantities computed by Algorithm 2. Assume $\mathbf{u} \leq \tfrac{1}{16}$, and that $\tfrac{1}{2}\mathbf{u}^{-3}\eta$ does not cause overflow. Then*

$$\texttt{cinf} = \mathrm{pred}(c) \qquad \textit{and} \qquad \mathrm{succ}(c) = \texttt{csup}. \tag{2.16}$$

PROOF. By the symmetry of $\mathbb{F}$ and the symmetry of the formulas in Algorithm 2 we may assume without loss of generality that $c > 0$. As for Algorithm 1 one verifies the assertions for c being the largest positive floating-point number, hence we assume without loss of generality that $\mathrm{succ}(c)$ is finite. We first prove (2.16) for the "if"-clause. Denote

$$
\begin{aligned}
e' &= \mathrm{fl}(\mathrm{fl}(\phi|c|)+\eta) \\
\texttt{cinf}' &= \mathrm{fl}(c-e') \\
\texttt{csup}' &= \mathrm{fl}(c+e')
\end{aligned}
$$

These are the quantities (with a prime appended) computed by Algorithm 1. The monotonicity (1.2) implies $e \leq e'$ and therefore

$$\texttt{cinf}' \leq \texttt{cinf} \qquad \text{and} \qquad \texttt{csup} \leq \texttt{csup}'.$$

Then $\mathbf{u} \leq \tfrac{1}{16}$, $|c| \geq 8\mathbf{u}^{-1}\eta$ and Theorem 2.2 imply

$$\texttt{cinf}' = \mathrm{pred}(c) \qquad \text{and} \qquad \mathrm{succ}(c) = \texttt{csup}'.$$

Hence, (2.16) is true if we prove

$$c - e < \mathscr{M}^-(c) \qquad \text{and} \qquad \mathscr{M}^+(c) < c+e$$

because this implies $\texttt{cinf} = \mathrm{fl}(c-e) \leq \mathrm{pred}(c)$ and $\texttt{csup} = \mathrm{fl}(c+e) \geq \mathrm{succ}(c)$, respectively. But $\mathbf{u}\,\mathrm{succ}(c) > \tfrac{1}{2}\mathbf{u}^{-1}\eta$ and (2.10) give

$$e = \mathrm{fl}(\phi c) \geq \mathrm{fl}(\mathbf{u}\,\mathrm{succ}(c)) = \mathbf{u}\,\mathrm{succ}(c) > \mathbf{u}\,\mathrm{ufp}(c),$$

and the result follows by (2.4) and (2.5).

The validity of (2.16) in the "elseif"-clause follows by (2.3). The "else"-clause is applied to $\mathbf{u}^{-1}\eta \leq |c| < \frac{1}{2}\mathbf{u}^{-2}\eta$, with $|C| \geq \mathbf{u}^{-2}\eta$, so that the computation of C cannot cause overflow and $C = \mathrm{fl}(\mathbf{u}^{-1}c) = \mathbf{u}^{-1}c$. Now the code is similar to the one in the "if"-clause applied to a scaled c, where the final multiplication by $\mathbf{u}$ is exact too. $\square$

If c is not very near to the underflow range, Algorithm 2 needs 2 flops to compute only one neighbor, and 3 flops to compute both. In any case a branch is needed, which we intentionally avoided in Algorithm 1.

Algorithms 1 and 2 may be used to bound the true value of an operation $\circ \in \{+, -, \times, \div\}$: If $c = \mathrm{fl}(a \circ b)$, then

$$\texttt{cinf} \leq a \circ b \leq \texttt{csup} \tag{2.17}$$

for $\texttt{cinf}, \texttt{csup}$ computed by Algorithm 1 or 2. However, no rounding error occurs for addition or subtraction of floating-point numbers if the result is less than $\mathbf{u}^{-1}\eta$. Hence one may try to develop an algorithm to compute $\texttt{cinf}, \texttt{csup}$ satisfying

$$\begin{aligned}
\texttt{cinf} &= \mathrm{pred}(c) \text{ and } \mathrm{succ}(c) = \texttt{csup} \text{ for } |c| \geq \mathbf{u}^{-1}\eta, \\
\texttt{cinf} &= c \qquad \text{and} \qquad c = \texttt{csup} \text{ otherwise.}
\end{aligned} \tag{2.18}$$

Then (2.17) would still be satisfied. To do this one may try to use

$$\begin{aligned}
e &= \mathrm{fl}(\phi'|c|) \\
\texttt{cinf} &= \mathrm{fl}(c - e) \\
\texttt{csup} &= \mathrm{fl}(c + e)
\end{aligned}$$

with a suitable factor ϕ' other than $\phi = \mathbf{u}(1 + 2\mathbf{u})$. In fact, in binary arithmetic, the factor ϕ in Algorithm 1 or 2 can be varied in a wide range without jeopardizing the quality of the results. With rounding to nearest with ties to even, there is no universal factor ϕ' to satisfy (2.18). Consider $\phi' = \mathbf{u}\,\mathrm{pred}(\frac{3}{2})$ and $c = \mathbf{u}^{-1}\eta$, then

$$e = \mathrm{fl}(\mathrm{pred}(\frac{3}{2})\eta) = \eta \quad \text{and} \quad \texttt{csup} = \mathrm{fl}(\mathbf{u}^{-1}\eta + \eta) = \mathbf{u}^{-1}\eta = c \neq \mathrm{succ}(c),$$

so $\phi' \geq \frac{3}{2}\mathbf{u}$ is necessary. But then we obtain for $c = 1$

$$e = \mathrm{fl}(\frac{3}{2}\mathbf{u}) = \frac{3}{2}\mathbf{u} \quad \text{and} \quad \texttt{cinf} = \mathrm{fl}(1 - \frac{3}{2}\mathbf{u}) = 1 - 2\mathbf{u} = \mathrm{pred}(\mathrm{pred}(c)) \neq \mathrm{pred}(c).$$

With rounding to nearest ties to away, both cases above give as expected $\mathrm{succ}(c)$ and $\mathrm{pred}(c)$, respectively, and one might hope to find a universal factor for this rounding mode. However, consider $\phi' = \mathbf{u}$ and $c = \mathrm{pred}(\mathbf{u}^{-1}\eta) = \mathbf{u}^{-1}(1 - \mathbf{u})\eta$. Then

$$e = \mathrm{fl}((1 - \mathbf{u})\eta) = \eta \quad \text{and} \quad \texttt{csup} = \mathrm{fl}(c + e) = \mathrm{succ}(c) \neq c,$$

so $\phi' < \mathbf{u}$ is necessary. But for $c = 1$ then

$$e = \mathrm{fl}(\phi') = \phi' < \mathbf{u} \quad \text{and} \quad \texttt{csup} = \mathrm{fl}(1 + \mathbf{u}) = 1 \neq \mathrm{succ}(c),$$

and (2.18) is again not satisfied.

We have implemented Algorithm 2 in the C language, and compared it to the C99 `nextafter(c, +Inf)` function call which in GNU `libc` implements the algorithm from [2]. We obtained the following timings in seconds for 10 million calls on a 2.667Ghz Core 2 under Linux with gcc 4.3.0 (using inline code):

input	`nextafter`	Algorithm 2
$[2^{-969}, 2^{1024})$	1.020s	0.464s
$[2^{-1021}, 2^{-969})$	1.020s	0.639s
$[2^{-1022}, 2^{-1021})$	1.020s	6.220s
$[2^{-1074}, 2^{-1022})$	12.434s	30.743s
NaN	0.425s	0.006s
+Inf	0.596s	0.208s
-Inf	0.990s	0.497s

The range $[2^{-969}, 2^{1024})$ corresponds to the "if"-branch of Algorithm 2, $[2^{-1021}, 2^{-969})$ to the "else"-branch, and $[2^{-1074}, 2^{-1021})$ to the "elseif"-branch. The timings for $[2^{-1022}, 2^{-1021})$ are better than those for $[2^{-1074}, 2^{-1022})$; this is most probably due to the fact that $[2^{-1022}, 2^{-1021})$ is still in the normal range, while $[2^{-1074}, 2^{-1022})$ is in the subnormal range, for which floating-point operations are usually slower in hardware. In conclusion, Algorithm 2 is clearly faster than `nextafter`, except in a tiny range near the subnormal domain.

3 Conclusion

We presented algorithms to compute (bounds for) the predecessor and successor of a floating-point number. The user may choose between the branch-free and fast variant Algorithm 1, the results of which always bound the true result, and are equal to neighbors except for a tiny range near underflow. Or the slightly slower Algorithm 2 computing always the neighbors of a floating-point number, without exception, which clearly outperforms `nextafter`.

Our algorithms may be used to simulate interval operations without changing the rounding mode. The advantage of Algorithm 1 over formula (1.5) for computing rigorous bounds of $a \circ b$ is that the width is often halved (see Fig. 3.1).

Fig. 3.1 Minimal, median, average and maximal difference in ulps between bounds of enclosing interval with formula (1.5) and Algorithm 1, for IEEE 754 `binary64` format. For each range between consecutive powers of two, 1,000,000 random values of c were used. The excluded range $[\frac{1}{2}, 2]\mathbf{u}^{-1}\eta$ from Theorem 2.2 corresponds to $[2^{-1022}, 2^{-1020}]$.

range	formula (1.5)	Algorithm 1
$[1/2, 1]$	2 / 2 / 2.999696 / 4	2 / 2 / 2.000000 / 2
$[2^{-1020}, 2^{-1019}]$	2 / 4 / 3.250584 / 4	2 / 2 / 2.000000 / 2
$[2^{-1021}, 2^{-1020}]$	4 / 4 / 4.000000 / 4	2 / 2 / 2.999696 / 4
$[2^{-1022}, 2^{-1021}]$	4 / 4 / 4.999696 / 6	4 / 4 / 4.000000 / 4
$[2^{-1023}, 2^{-1022}]$	4 / 4 / 4.000000 / 4	2 / 2 / 2.000000 / 2
$[2^{-1024}, 2^{-1023}]$	2 / 2 / 2.000000 / 2	2 / 2 / 2.000000 / 2

However, this applies only if $a, b \in \mathbb{F}$. In applications often interval operations $A \circ B$ for thick intervals A, B are executed. The wider A and B are, the less is the gain of Algorithm 1 compared to (1.5).

References

1. S. Boldo and J.-M. Muller. Some Functions Computable with a Fused-mac. In Paolo Montuschi and Eric Schwarz, editors, *Proceedings of the 17th Symposium on Computer Arithmetic*, pages 52–58, Cape Cod, USA, 2005.
2. W.J. Cody, Jr. and J.T. Coonen. Algorithm 722: Functions to support the IEEE standard for binary floating-point arithmetic. *ACM Transactions on Mathematical Software*, 19(4):443–451, 1993.
3. Institute of Electrical, and Electronic Engineers. IEEE Standard for Floating-Point Arithmetic. *IEEE Standard 754-2008*. Revision of ANSI-IEEE Standard 754-1985. Approved June 12, 2008: IEEE Standards Board.
4. American National Standards Institute, Institute of Electrical, and Electronic Engineers. IEEE Standard for Radix-Independent Floating-Point Arithmetic. *ANSI/IEEE Standard, Std. 854-1987*, 1987.
5. R.B. Kearfott, M. Dawande, K. Du, and C. Hu. Algorithm 737: INTLIB: A Portable Fortran 77 Interval Standard-Function Library. *ACM Transactions on Mathematical Software*, 20(4):447–459, 1994.
6. S.M. Rump, T. Ogita, and S. Oishi. Accurate Floating-point Summation Part I: Faithful Rounding. *SIAM Journal on Scientific Computing (SISC)*, 31(1):189–224, 2008.
7. M. Daumas, L. Rideau, and L. Théry. *A generic library of floating-point numbers and its application to exact computing*. In 14th International Conference on Theorem Proving in Higher Order Logics, Edinburgh, Scotland, pp. 169–184, 2001.
8. S. Boldo. Preuves formelles en arithmétiques à virgule flottante. Ph.D. dissertation, École Normale Supérieure de Lyon, 2004.