A Library for Symbolic Floating-Point Arithmetic

Claude-Pierre Jeannerod, Nicolas Louvet, Jean-Michel Muller, Antoine Plet

A Library for Symbolic Floating-Point Arithmetic

Claude-Pierre Jeannerod[1,*], Nicolas Louvet[2,*], Jean-Michel Muller[3,*], and Antoine Plet[4,*]

[1]Inria, email: claude-pierre.jeannerod@ens-lyon.fr
[2]UCB Lyon 1, email: nicolas.louvet@ens-lyon.fr
[3]CNRS, email: jean-michel.muller@ens-lyon.fr
[4]ENS de Lyon, email: antoine.plet@ens-lyon.fr
[*]LIP (CNRS, ENS de Lyon, Inria, UCB Lyon 1), Université de Lyon

November 23, 2015

Abstract

To analyze a priori the accuracy of an algorithm in floating-point arithmetic, one usually derives a uniform error bound on the output, valid for most inputs and parametrized by the precision p. To show further that this bound is sharp, a common way is to build an input example for which the error committed by the algorithm comes close to that bound, or even attains it. Such inputs may be given as floating-point numbers in one of the IEEE standard formats (say, for $p = 53$) or, more generally, as expressions parametrized by p, that can be viewed as *symbolic floating-point* numbers. With such inputs, a sharpness result can thus be established for virtually all reasonable formats instead of just one of them. This, however, requires the ability to run the algorithm on those inputs and, in particular, to compute the correctly-rounded sum, product, or ratio of two symbolic floating-point numbers. The goal of this paper is to show how these basic arithmetic operations can be performed automatically. We introduce a way to model symbolic floating-point data, and present algorithms for round-to-nearest addition, multiplication, fused multiply-add and, in some cases, division. An implementation as a Maple library is also described, and experiments using examples from the literature are provided to illustrate its interest in practice.

Keywords. Floating-point arithmetic; correct rounding; rounding error analysis; symbolic computation; Maple library.

1 Introduction

When designing floating-point algorithms that are building blocks for numerical computing (for example, arithmetic operations on complex numbers, or the evaluation of low-degree polynomials), it is important to provide rigorous and

sharp error bounds, that can be safely used for analyzing the behavior of larger algorithms based on them.

To illustrate the sharpness of an error bound for a floating-point algorithm, one way is to provide examples for which the error generated is very close to or reaches the bound. A possible approach to find such examples is to run the algorithm on a set of random inputs, or to perform exhaustive tests in low precisions. However, since the number of possible inputs is exponential in the precision p, one can hardly expect to find examples in practical precisions ($p = 24$ or 53 for the commonly used binary32 and binary64 formats [1]) with such approaches. Hence, it is common to build examples parametrized by the precision p, and to show that the error committed by the algorithm either attains the error bound, or is equivalent to the error bound as $p \to \infty$; see for example [2, 3, 4].

Let us cite an example taken from [3]. In this paper, the authors analyze the algorithm below due to Kahan (Algorithm 1) for evaluating $x = ad - bc$, where a, b, c, d are floating-point numbers, and assuming a fused multiply-add is available. Here and hereafter, RN denotes the function that rounds a real to the floating-point number nearest to it and, in case of tie, whose last significand digit is even.

Algorithm 1 (Evaluation of $x = ad - bc$)

algorithm Kahan(a, b, c, d)
  $\widehat{w} \leftarrow \mathrm{RN}(bc)$;
  $e \leftarrow \mathrm{RN}(\widehat{w} - bc)$;
  $\widehat{f} \leftarrow \mathrm{RN}(ad - \widehat{w})$;
  $\widehat{x} \leftarrow \mathrm{RN}(\widehat{f} + e)$;

They show the following result (in base β and precision p): if no underflow or overflow occurs, then $|\widehat{x} - x| \leqslant 2u|x|$, where $u = \frac{1}{2}\beta^{1-p}$ is the unit roundoff; moreover, for a fixed even β, this relative error bound is asymptotically optimal as $p \to \infty$. To prove the asymptotic optimality of the error bound, the authors provide the following floating-point inputs parametrized by β and p (Example 4.6 in [4]):

$$\begin{aligned} a = b &= \beta^{p-1} + 1, \\ c &= \beta^{p-1} + \tfrac{\beta}{2}\beta^{p-2}, \\ d &= 2\beta^{p-1} + \tfrac{\beta}{2}\beta^{p-2}; \end{aligned} \tag{1}$$

then, they show that the relative error on the final result is

$$\frac{2u}{1+2u},$$

which is equivalent to $2u$ as $u \to 0$ (or, β being fixed, as $p \to \infty$). Note that in this example, the inputs a, b, c and d are functions of the base β and the precision p.

Such examples can be built by first testing the error generated by the algorithm in low precisions, then guessing some inputs parametrized by β and p, and next carrying paper-and-pencil calculations to check if the guessed inputs effectively correspond to cases we are looking for. Our goal is to automate this last step, in order to save time, to avoid errors, and to be able to try more candidates.

We refer to functions representing floating-point numbers parametrized by the precision p (with $\beta \geqslant 2$ fixed and even) as *symbolic floating-point numbers*. We propose to manipulate symbolic floating-point numbers, in a computer algebra system such as Maple, very much as real numbers or polynomials can be manipulated within such a system.

In this paper, we first define in section 2 the set $\mathbb{S}$ of sums of exponentials functions that will serves as the domain of the rounding function, and we give the properties that will be used for handling the elements of $\mathbb{S}$. Considering the precision p as a linear function of a variable k, we introduce in section 3 the set $\mathbb{SF}_p$ of symbolic floating-point numbers in precision p. We then define a round-to-nearest function RN_p from $\mathbb{S}$ to $\mathbb{SF}_p$, we give an algorithm to compute it in practice, and we show how it matches the roundTiesToEven rounding-direction attribute of the IEEE 754 [1]. Under some technical assumptions, we also provide an algorithm for rounding to the nearest the division of two symbolic floating-point numbers. An implementation as a Maple library is described in section 4, and experiments using examples from the literature are provided to illustrate its practical use.

Notation. Here and hereafter, k is an integer variable and β is a fixed integer corresponding the floating-point base. We assume that β is even and at least two, and in practice we shall take $\beta = 2$ or $\beta = 10$ as prescribed by the IEEE 754-2008 standard [1].

We write $\mathbb{Z}$ and $\mathbb{N}$ to denote the integers and the nonnegative integers, respectively. To indicate that a property holds asymptotically, we will often write "$P(k)$ as $k \to \infty$" as a shortcut for the fact that "there exists $k_0 \in \mathbb{N}$ such that $P(k)$ holds for all $k \geqslant k_0$."

Let $\mathbb{L}$ denote the set of linear functions in k with integer coefficients:

$$\mathbb{L} = \{ak + b \,:\, a, b \in \mathbb{Z}\};$$

let also

$$\mathbb{L}_{\geqslant \tau} = \{\ell \in \mathbb{L} \,:\, \ell(k) \geqslant \tau \text{ as } k \to \infty\}, \quad \tau \in \mathbb{R}.$$

Given a function $g : \mathcal{X} \to \mathbb{R}$ such that $\min_{x \in \mathcal{X}} g(x)$ is well defined, we denote by $\arg\min_{x \in \mathcal{X}} g(x)$ the set

$$\big\{x^* \in \mathcal{X} \;:\; g(x^*) = \min_{x \in \mathcal{X}} g(x)\big\}.$$

2 Sums of exponentials with base β

Before introducing symbolic floating-point numbers, we have to formalize a more general set $\mathbb{S}$ which will be the domain of our rounding function. The elements of $\mathbb{S}$ will be sums of exponentials in base β, with linear exponents over the variable k and integer coefficients. We then present the main properties satisfied by the elements of $\mathbb{S}$ and the arithmetic operations that are of interest for us, which can be performed according to these properties.

Definition 1. *$\mathbb{S}$ is the set of sums of exponentials*

$$f(k) = \sum_{i=1}^{n} c_i \beta^{e_i(k)}, \qquad k \in \mathbb{N}, \tag{2a}$$

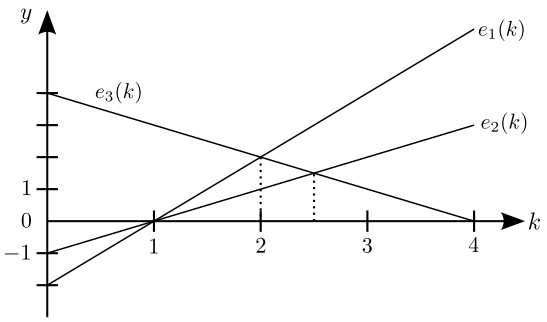

Figure 1: The linear functions $e_1(k) = 2k-2$, $e_2(k) = k-1$ and $e_3(k) = -k+4$, such that $e_1(k) > e_2(k) > e_3(k)$ for all integer $k \geqslant 3$.

where $n \in \mathbb{N}$ *and where, for* $n \geqslant 1$*, the* c_i *and* e_i *satisfy*

$$|c_i| \in \{1, 2, \ldots, \beta - 1\} \quad \textit{and} \quad e_i \in \mathbb{L}, \tag{2b}$$

$$e_1(k) > e_2(k) > \cdots > e_n(k) \quad \textit{as } k \to \infty; \tag{2c}$$

for $n = 0$*, we take* $f(k) = 0$.

If we define the β-adic fractions, generalizing the dyadic fractions, as the rational numbers whose denominator is a power of β, then the elements of $\mathbb{S}$ are functions from $\mathbb{N}$ to the β-adic fractions, and can be seen as special cases of so-called *exponential polynomials*, with base β and exponents linear in k; see for example [5, p. 7]. The asymptotic ordering of these linear exponents is illustrated by Figure 1 for $n = 3$.

Writing

$$e_i(k) = a_i k + b_i, \qquad i = 1, \ldots, n, \tag{3}$$

we see in particular that the ordering of the e_i implies

$$a_1 \geqslant a_2 \geqslant \cdots \geqslant a_n. \tag{4}$$

Let $\mathbb{S}_{\geqslant 0}$ denote the subset of $\mathbb{S}$ restricted to the functions $f = \sum_i c_i \beta^{e_i}$ for which all the linear functions e_i are in $\mathbb{L}_{\geqslant 0}$. For $f \in \mathbb{S}_{\geqslant 0}$, there exists $k_0 \in \mathbb{N}$ such that $\forall k \geqslant k_0$, $f(k) \in \mathbb{Z}$.

2.1 Representations for $\mathbb{S}$

There are at least two ways to represent the functions in $\mathbb{S}$. A first possible representation is directly suggested by (2a) and (3), and consists of the finite sequence

$$((a_i, b_i, c_i))_{i=1:n}. \tag{5}$$

Because of the redundancy in the digit set used for the c_i, two or more different sequences can represent the same function. For example, since $\beta^k - \beta^{k-1} = (\beta - 1)\beta^{k-1}$, one can represent $f(k) = \beta^{k-1}$ by $((1, 0, 1), (1, -1, -1))$ or $((1, -1, \beta - 1))$. (This is of course similar to classical redundant, signed-digit number systems *à la* Avizienis [6].)

An alternative representation consists of identifying $f \in \mathbb{S}$ with the Laurent polynomial $F(X) = \sum_{i=1}^{n} c_i \beta^{b_i} X^{a_i}$ after the change of variable $\beta^k = X$. More precisely, since $a_i, b_i, c_i \in \mathbb{Z}$, we have $F \in \mathbb{P}$, where

$$\mathbb{P} = R[X, X^{-1}], \qquad R = \{n\beta^m \,:\, n, m \in \mathbb{Z}\}.$$

By expanding the positive elements of R in base β, it is easy to check that the map $\varphi : f \mapsto F$ is bijective. Thus, writing F in the basis $1, X, X^{-1}, X^2, X^{-2}, \ldots$ of $\mathbb{P}$ as $F(X) = \sum_{j \in J} C_j X^{E_j}$ with $(C_j, E_j) \in R\backslash\{0\} \times \mathbb{Z}$ and J a finite subset of $\mathbb{Z}$, we deduce that f can also be represented by the (unique) finite sequence

$$((C_j, E_j))_{j \in J}. \tag{6}$$

Both representations will be useful in this paper, and we shall use one or the other depending on what is most convenient for the operation at hand.

2.2 Addition and multiplication in $\mathbb{S}$

Since the map φ is bijective and since $\mathbb{P}$ is a commutative ring with 1 (for the usual + and × on polynomials), the set $\mathbb{S}$ inherits the same algebraic structure, with addition and multiplication defined as

$$f_1 \operatorname{op} f_2 = \varphi^{-1}\big(\varphi(f_1) \operatorname{op} \varphi(f_2)\big)$$

for $f_1, f_2 \in \mathbb{S}$ and $\operatorname{op} = +, \times$.

In practice, this implies that the system $\mathbb{S}$ is closed under addition and multiplication, and that operating on two functions $f_1, f_2 \in \mathbb{S}$ given in the representation (5) can be done simply by converting them into their polynomial counterparts (6), operating on the latter using standard polynomial arithmetic, and converting the result back from (6) to (5).

2.3 Sign, exponent, and splitting of elements of $\mathbb{S}$

Besides + and ×, it will be useful to define several low-level functions on elements of $\mathbb{S}$. Especially, as a subset of the real numbers, the β-adic fractions have a *sign*, and when they are nonzero, they also have a base-β *exponent* (the base-β exponent of $x \neq 0$ is the largest integer e such that $\beta^e \leqslant |x|$). These low-level functions are essential when defining and manipulating conventional floating-point numbers, we will also need them to deal with symbolic floating-point numbers. We begin with the following property about elements of $\mathbb{S}$, which allows us to define the functions previously mentioned.

Property 1. *Let $f \in \mathbb{S}$ as in (2) and (3) with $n \geqslant 1$, and let $c \in \mathbb{Q}$ such that*

$$c = \sum_{i\,:\,a_i = a_1} c_i \beta^{b_i - b_1}. \tag{7}$$

Then

(i) $|f(k)| \leqslant (\beta - \beta^{1-n})\beta^{e_1(k)}$ *as* $k \to \infty$*;*

(ii) $f(k) \sim c\,\beta^{e_1(k)}$ *as* $k \to \infty$*;*

(iii) $\operatorname{sign}(c) = \operatorname{sign}(c_1)$.

Proof. Using the triangle inequality together with the fact that $|c_i| \leqslant \beta - 1$ gives $|f(k)| \leqslant (\beta-1)\sum_{i=1}^{n} \beta^{e_i(k)}$. Since the strict ordering in (2c) implies that each e_i satisfies

$$e_i(k) \leqslant e_1(k) - i + 1 \quad \text{as } k \to \infty,$$

we deduce that

$$|f(k)| \leqslant (\beta-1)\beta^{e_1(k)} \sum_{i=1}^{n} \beta^{-i+1} \quad \text{as } k \to \infty$$

and, rewriting the sum as $(\beta - \beta^{1-n})/(\beta-1)$, we obtain (i).

To prove (ii), we use (4) to rewrite $f(k)$ as

$$\begin{aligned} f(k) &= \sum_{i\,:\,a_i = a_1} c_i \beta^{e_i(k)} + \sum_{i\,:\,a_i < a_1} c_i \beta^{e_i(k)} \\ &= \left(c + \sum_{i\,:\,a_i < a_1} c_i \beta^{e_i(k) - e_1(k)} \right) \beta^{e_1(k)} \end{aligned}$$

with c as in (7). Now, the c_i are independent of k and, on the other hand, the strict inequality $a_i < a_1$ implies that the difference $e_i(k) - e_1(k)$ tends to $-\infty$ as $k \to \infty$. Hence the ratio $f(k)/\beta^{e_1(k)}$ tends to c as $k \to \infty$, thus establishing (ii).

Finally, let us prove (iii). Since $|c_1| \geqslant 1$ by (2b), it suffices to check that $|c - c_1| < 1$. We have

$$|c - c_1| \leqslant \sum_{2 \leqslant i\,:\,a_i = a_1} |c_i| \beta^{b_i - b_1},$$

where $|c_i| \leqslant \beta - 1$ and where, due to (2c),

$$a_i = a_1 \quad \Rightarrow \quad b_i \leqslant b_1 - i + 1.$$

Hence $|c - c_1| < (\beta-1)\sum_{i \geqslant 2} \beta^{-i+1} = 1$, as wanted. □

This property shows that any nonzero function $f \in \mathbb{S}$ is eventually either positive or negative, that is, there exists $k_0 \in \mathbb{N}$ such that either $f(k) > 0$ for all $k \geqslant k_0$, or $f(k) < 0$ for all $k \geqslant k_0$. Hence there is a unique $s \in \{-1, 1\}$ such that $s = \operatorname{sign}(f(k))$ as $k \to \infty$. Since this sign is the one of the leading constant c_1, this allows us to define the function $\operatorname{sign} : \mathbb{S} \to \{-1, 0, 1\}$ as follows:

$$\operatorname{sign}(f) = \begin{cases} \operatorname{sign}(c_1) & \text{if } f \in \mathbb{S} \backslash \{0\}, \\ 0 & \text{if } f = 0. \end{cases}$$

As an immediate consequence, we can define further the comparison operations $=, \neq, \leqslant, <, \geqslant, >$ of two elements of $\mathbb{S}$ (by computing the sign of their difference). Also, recalling that $\mathbb{S}$ is a commutative ring with 1, we get an absolute value by multiplying any element of $\mathbb{S}$ by its sign:

$$|f| = \operatorname{sign}(f) f.$$

Using the comparisons and absolute value in $\mathbb{S}$, we can now derive the following property.

Property 2. *Let $f \in \mathbb{S}$ as in (2) and (3) with $n \geqslant 1$. Then there exists a unique $e \in \mathbb{L}$ satisfying*

$$\beta^e \leqslant |f| < \beta^{e+1}.$$

Furthermore, writing e_c to denote the unique integer such that $\beta^{e_c} \leqslant |c| < \beta^{e_c+1}$ for c as in (7), we have

$$e = \begin{cases} e_1 + e_c - 1 & \text{if } |f| < \beta^{e_1+e_c}, \\ e_1 + e_c & \text{otherwise.} \end{cases} \tag{8}$$

Proof. Property 1 implies that $|f| = |c|\beta^{e_1}(1+\epsilon)$ with $\epsilon(k) \to 0$ as $k \to \infty$. Hence

$$|f| = \beta^{e_1}\big(\beta^{e_c+1} + \nu\big) = \beta^{e_1}\big(\beta^{e_c-1} + \pi\big),$$

where $\nu = |c| - \beta^{e_c+1} + |c|\,\epsilon$ and $\pi = |c| - \beta^{e_c-1} + |c|\,\epsilon$. Now, by definition of e_c and since $\beta > 1$, we have the strict inequalities $\beta^{e_c-1} < |c| < \beta^{e_c+1}$ and, therefore, ν is eventually negative and π is eventually positive:

$$\nu(k) < 0 < \pi(k) \quad \text{as } k \to \infty.$$

Since $e_1 \in \mathbb{L}$ and $e_c \in \mathbb{Z}$, both $\beta^{e_1+e_c-1}$ and $\beta^{e_1+e_c-1}$ are in $\mathbb{S}$, and comparing them to $|f| \in \mathbb{S}$ as $k \to \infty$ gives

$$\beta^{e_1+e_c-1} < |f| < \beta^{e_1+e_c+1}.$$

The existence of e as defined in Property 2 thus follows from comparing $\beta^{e_1+e_c} \in \mathbb{S}$ to $|f|$.

Since the elements of $\mathbb{L}$ are integer-valued, it is easy to show by contradiction that if a linear function $e' \in \mathbb{L}$ satisfies $\beta^{e'} \leqslant |f| < \beta^{e'+1}$ then $e'(k) = e(k)$ as $k \to \infty$ and, therefore, $e' = e$. This establishes the uniqueness of e. □

Property 2 allows us to take the exponent of any element of $\mathbb{S}$ by defining the following function $\mathbb{S} \to \mathbb{L}$:

$$\text{exponent}(f) = \begin{cases} e \text{ as in (8)} & \text{if } f \in \mathbb{S}\backslash\{0\}, \\ 0 & \text{if } f = 0. \end{cases}$$

Finally, we define the split function that decomposes any $f \in \mathbb{S}$ into a high and a low part with respect to a given linear function $e \in \mathbb{L}$. Given a representation of f by $\sum_i c_i\beta^{e_i}$ as in (2), let j be the unique index in $\{0, 1, \ldots, n\}$ such that

$$e_j(k) \geqslant e(k) > e_{j+1}(k) \quad \text{as } k \to \infty, \tag{9}$$

where by convention we set $e_0 = +\infty$ and $e_{n+1} = -\infty$. Then we note

$$\text{split}(f, e) = (f_h, f_\ell)$$

with

$$f_h = \sum_{i \leqslant j} c_i\beta^{e_i} \quad \text{and} \quad f_\ell = \sum_{i > j} c_i\beta^{e_i}.$$

Since the representation of f as a sum of exponentials is not unique, this decomposition may not be unique. However, any representation can be used to ensure the correctness of the algorithms presented in the rest of the paper. For this reason, we consider split as a function from $\mathbb{S} \times \mathbb{L}$ to $\mathbb{S} \times \mathbb{S}$.

The main properties of the split functions are stated in the following property which will allow us to build algorithms on top of this function.

Property 3. *Given $f \in \mathbb{S}$ as in (2) and (3) with $n \geqslant 1$, and $e \in \mathbb{L}$, let $j \leqslant n$ be defined as in (9), $e_f = \text{exponent}(f)$ and $(f_h, f_\ell) = \text{split}(f, e)$. Then*

(i) $f = f_h + f_\ell$;

(ii) $|f_\ell| \leqslant \beta^e - \beta^{e-n+j}$;

(iii) $f_h = M\beta^e$ *with* $M \in \mathbb{S}_{\geqslant 0}$ *and* $\beta^{e_f - e} \leqslant |M| \leqslant \beta^{e_f - e + 1}$.

Proof. The proof of *(i)* is straightforward from the definitions of f_h and f_ℓ. *(ii)* follows from Property 1*(i)* and the definition of j in (9). For *(iii)*, by definition of f_h we have $f_h = M\beta^e$ with $M = \sum_{i \leqslant j} c_i \beta^{e_i - e}$. Since $e_i(k) \geqslant e_j(k) \geqslant e(k)$ as $k \to \infty$ implies $e_i - e \in \mathbb{L}_{\geqslant 0}$, we deduce that $M \in \mathbb{S}_{\geqslant 0}$. Moreover, from the triangular inequality and the definition of the exponent, we have $\beta^{e_f} - |f_\ell| \leqslant |M|\beta^e < \beta^{e_f+1} + |f_\ell|$. From *(ii)* we deduce $\beta^{e_f - e} - 1 < |M| < \beta^{e_f - e + 1} + 1$, and *(iii)* follows from $M \in \mathbb{S}_{\geqslant 0}$. □

3 Symbolic floating-point arithmetic

The purpose of the definition of the set $\mathbb{S}$ and its arithmetic was to formally define the symbolic data that we want to manipulate, and to define some low-level functions. We now have all that is needed to formally define symbolic floating-point numbers, as special elements of $\mathbb{S}$. In conventional, base-β floating-point arithmetic (and assuming an unlimited exponent range for the sake of simplicity), a nonzero precision-p floating-point number is a number that can be written $M \times \beta^e$, where M is an integer in the range $[\beta^{p-1}, \beta^p)$ and e is an integer. The formalism introduced earlier allows us to define a set of symbolic floating-point numbers, with $\mathbb{S}_{\geqslant 0}$ for the integer M and $\mathbb{L}$ for the exponent e. More precisely,

Definition 2. *Given a function $p \in \mathbb{L}_{\geqslant 2}$, let*

$$\mathbb{SF}_p = \{0\} \cup \{M\beta^e \,:\, M \in \mathbb{S}_{\geqslant 0},\, e \in \mathbb{L},\, \beta^{p-1} \leqslant |M| < \beta^p\}.$$

We have $\mathbb{SF}_p \subset \mathbb{S}$ and every function $f \in \mathbb{SF}_p$ satisfies the following: there exists $k_0 \in \mathbb{N}$ such that for all $k \geqslant k_0$, its value $f(k)$ is a floating-point number in base β and precision $p(k) \in \mathbb{N}_{\geqslant 2}$.

The set $\mathbb{SF}_p$ can thus be considered as a set of "symbolic floating-point numbers" whose precision p is parametrized by the variable k. In particular, $0 \in \mathbb{SF}_p$ and $\mathbb{SF}_p = -\mathbb{SF}_p$. Moreover, if $f \in \mathbb{SF}_p \backslash \{0\}$ and $e_f = \text{exponent}(f)$, then f can be written as $f = M\beta^{e_f + 1 - p}$, with $M \in \mathbb{S}_{\geqslant 0}$ and $\beta^{p-1} \leqslant |M| < \beta^p$: we call $|M|$ the significand of f.

We now explain how the split function behaves with respect to this new formalism, as a consequence of Property 3.

Corollary 1. *Given $p \in \mathbb{L}_{\geqslant 2}$ and a nonzero $f \in \mathbb{S}$, let $e_f = \text{exponent}(f)$ and $(f_h, f_\ell) = \text{split}(f, e_f - p + 1)$. If $c \in \{0, \text{sign}(f_\ell)\}$, then*

(i) $\beta^{e_f} \leqslant |f_h + c\beta^{e_f - p + 1}| \leqslant \beta^{e_f + 1}$;

(ii) $f_h + c\beta^{e_f - p + 1} \in \mathbb{SF}_p$.

Proof. From *(iii)* in Property 3, we have $f_h = M\beta^{e_f - p + 1}$, with $\beta^{p-1} \leqslant |M| \leqslant \beta^p$. Thus, using *(i)* in Property 3 we get $f = (M + c)\beta^{e_f - p + 1} + f_\ell - c\beta^{e_f - p + 1}$. Moreover, *(ii)* in Property 3 gives $|f_\ell - c\beta^{e_f - p + 1}| < \beta^{e_f - p + 1}$:

- if $c = 0$, it is straightforward;
- otherwise, $|c| = 1$ (and $f_\ell \neq 0$), hence $|f_\ell - c\beta^{e_f-p+1}| = \beta^{e_f-p+1} - |f_\ell| < \beta^{e_f-p+1}$.

Together with the definition of e_f, this implies $\beta^{p-1} - 1 < |M + c| < \beta^p + 1$. Since $M + c \in \mathbb{S}_{\geqslant 0}$, we deduce that $\beta^{p-1} \leqslant |M + c| \leqslant \beta^p$, and *(i)* follows. Finally, either $|M + c| = \beta^p$ or $|M + c| < \beta^p$, and in both cases *(ii)* holds. □

Property 4. *Let $p \in \mathbb{L}_{\geqslant 2}$, and $f, g \in \mathbb{SF}_p$, if $0 < |f| < |g|$, then $|f - g| \geqslant \beta^{e_f-p+1}$.*

Proof. Using the definition of $\mathbb{SF}_p$, we write $f = M_f\beta^{e_f-p+1}$ and $g = M_g\beta^{e_g-p+1}$. We have $|f-g| = |M_f-M_g\beta^{e_g-e_f}|\beta^{e_f-p+1}$, and $0 < |f| < |g|$ implies $e_g-e_f \geqslant 0$, so that $M_f - M_g\beta^{e_g-e_f} \in \mathbb{S}_{\geqslant 0}$. Since $f \neq g$, $|M_f - M_g\beta^{e_g-e_f}| \geqslant 1$ and the property follows. □

3.1 Round-to-nearest function from $\mathbb{S}$ to $\mathbb{SF}_p$

In conventional floating-point arithmetic, if the arithmetic operation op is correctly rounded, and if the rounding function is RN, then whenever $a \operatorname{op} b$ is computed, where a and b are floating-point numbers, the obtained result is $\mathrm{RN}(a \operatorname{op} b)$. Hence a natural way of defining the arithmetic operations on the symbolic floating-point numbers is to first define a rounding function: the result of an arithmetic operation in symbolic floating-point arithmetic will be equal to the rounding function applied to the exact result. In the following, we will define the *round-to-nearest* function with ties-to-even tie breaking rule. Very similarly, one could define ties-to-away tie breaking rule, and also the directed rounding modes standardized in [1].

Property 5. *Given $f \in \mathbb{S}$ and $p \in \mathbb{L}_{\geqslant 2}$, $\min_{g\in\mathbb{SF}_p} |f - g|$ is well defined, and this minimum is attained by at most two functions in $\mathbb{SF}_p$.*

Proof. If $f = 0$, then $g = f$ reaches the minimum. Hence, let us now assume $f \neq 0$. We denote exponent(f) by e_f, and we define

$$\Omega := \{g \in \mathbb{SF}_p : |f - g| < \beta^{e_f-p+1}\}.$$

If $(f_h, f_\ell) = \mathrm{split}(f, e_f-p+1)$, then f_h is in Ω as a consequence of Property 3: Corollary 1 states that $f_h \in \mathbb{SF}_p$ and *(i)* implies $|f - f_h| < \beta^{e_f-p+1}$. This proves that Ω is nonempty.

If g_1 and g_2 are two distinct functions in Ω, then

$$|g_1 - g_2| \leqslant |g_1 - f| + |g_2 - f| < 2\beta^{e_f-p+1}. \tag{10}$$

Moreover, using the triangular inequality, and since $p \in \mathbb{L}_{\geqslant 2}$, we have $|g_i| \geqslant |f| - \beta^{e_f-p+1} \geqslant \beta^{e_f-1}$. From Property 4,

$$|g_1 - g_2| \geqslant \beta^{e_f-p}. \tag{11}$$

Now, let us assume that Ω contains m distinct elements g_i such that $g_1 < g_2 < \cdots < g_m$. Then (10), the triangular inequality, and (11), imply $2\beta^{e-p+1} > |g_m - g_1| \geqslant (m-1)\beta^{e_f-p}$, hence $m < 2\beta + 1$, which proves that Ω is finite.

Since Ω is nonempty and finite, $\min_{g\in\Omega}|f-g|$ is well defined and if $h \in \arg\min_{g\in\Omega}|f-g|$, then for all $g \in \mathbb{SF}_p$, $|h-f| \leqslant |g-f|$, so that $\min_{g\in\mathbb{SF}_p}|f-g|$ is also well defined. Moreover, we cannot have more than two functions of $\mathbb{S}$ equidistant to f since $|f-g| = d$ implies $g = f \pm d$. Hence, the minimum is obtained for at most two functions. □

Let us now define $\mathbb{SM}_p$ as the subset of $\mathbb{S}$ restricted to the functions f for which $\min_{g\in\mathbb{SF}_p}|f-g|$ is obtained exactly twice. Elements of $\mathbb{SM}_p$ are called *midpoints* in precision p.

Given a tie-breaking function r from $\mathbb{SM}_p$ to $\mathbb{SF}_p$ such that, for $f \in \mathbb{SM}_p$, $r(f) \in \arg\min_{g\in\mathbb{SF}_p}|f-g|$, we define a round-to-nearest function $\mathrm{RN}_p : \mathbb{S} \to \mathbb{SF}_p$ as follows:

$$\mathrm{RN}_p(f) = \begin{cases} \arg\min_{g\in\mathbb{SF}_p}|f-g| & \text{if } f \notin \mathbb{SM}_p, \\ r(f) & \text{otherwise.} \end{cases} \tag{12}$$

Note that, for a given tie-breaking function r, $\mathrm{RN}_p(f)$ is uniquely defined for any $f \in \mathbb{S}$.

To define a tie-breaking function analogous to the standard ties-to-even rule, let us define the notion of parity for elements of $\mathbb{S}_{\geqslant 0}$. Given $M \in \mathbb{S}_{\geqslant 0}$, we say that M is even (resp. odd) if there exists $h \in \mathbb{S}_{\geqslant 0}$ such that $M = 2h$ (resp. $M = 2h+1$); it is equivalent to say that $M(k)$ is even (resp. odd) as $k \to \infty$.

In the following, we consider the tie-breaking rule 'to even' which chooses $\widehat{f} = M\beta^e$ with M even. It is well defined since two consecutive symbolic floating-point numbers cannot satisfy this property simultaneously.

Property 6. *The round-to-nearest function defined by (12) with the tie-breaking function 'to the nearest even' given above satisfies:*

(i) RN_p *is nondecreasing;*

(ii) $\mathrm{RN}_p(-f) = -\mathrm{RN}_p(f)$*;*

(iii) for any exponent $e \in \mathbb{L}$, $\mathrm{RN}_p(\beta^e f) = \beta^e \mathrm{RN}_p(f)$.

Proof. We first prove *(i)*. Given $f < g \in \mathbb{S}$ let $\widehat{f} = \mathrm{RN}_p(f)$ and $\widehat{g} = \mathrm{RN}_p(g)$. We assume by contradiction $\widehat{f} > \widehat{g}$:

- If $\widehat{f} \leqslant g$, then $0 \leqslant g - \widehat{f} < g - \widehat{g}$.
- Similarly, if $\widehat{g} \geqslant f$, then $0 \leqslant \widehat{g} - f < \widehat{f} - f$.
- Otherwise, $\widehat{g} < f < g < \widehat{f}$ and $0 < \widehat{f} - g < g - \widehat{g}$.

Hence, in all the cases we have a contradiction either with the definition of $\widehat{f}$ or with the one of $\widehat{g}$.

(ii) comes from $|f-g| = |-f-(-g)|$ and from the fact that the significand remains unchanged when taking the opposite.

Likewise, *(iii)* comes from $\beta^e|f-g| = |\beta^e f - \beta^e g|$ and the fact that the significand remains unchanged when multiplying by a power of β. □

Algorithm 2 (Rounding to the nearest function)

algorithm round(f, p)
  $e_f \leftarrow \text{exponent}(f)$;
  $(f_h, f_\ell) \leftarrow \text{split}(f, e_f - p + 1)$;
  if $|f_\ell| < \frac{1}{2}\beta^{e_f-p+1}$ **then** $g \leftarrow f_h$;
  else if $|f_\ell| > \frac{1}{2}\beta^{e_f-p+1}$ **then** $g \leftarrow f_h + \text{sign}(f_\ell)\beta^{e_f-p+1}$;
  else
    // $|f_\ell| = \frac{1}{2}\beta^{e_f-p+1}$ and $f_h = \sum_{i=1}^{j} c_i\beta^{e_i}$
    if $e_j = e_f - p + 1$ and c_j is odd **then**
      $g \leftarrow f_h + \text{sign}(f_\ell)\beta^{e_f-p+1}$;
    else $g \leftarrow f_h$;

3.2 Round-to-nearest algorithm

In the previous subsection, we defined the round-to-nearest function from $\mathbb{S}$ to $\mathbb{SF}_p$, for a given $p \in \mathbb{L}_{\geqslant 2}$. Now we provide the algorithm for evaluating this function.

Property 7. *Given $p \in \mathbb{L}_{\geqslant 2}$ and $f \in \mathbb{S} - \{0\}$, Algorithm 2 computes $g \in \mathbb{SF}_p$ such that $\beta^{e_f} \leqslant |g| \leqslant \beta^{e_f+1}$ and either $|f-g| < \frac{1}{2}\beta^{e_f-p+1}$ or $|f-g| = \frac{1}{2}\beta^{e_f-p+1}$ and the significand of g is even.*

Proof. We know from Corollary 1 that $g \in \mathbb{SF}_p$ and that $\beta^{e_f} \leqslant |g| \leqslant \beta^{e_f+1}$. Let us consider the first two cases:

- If $|f_\ell| < \frac{1}{2}\beta^{e_f-p+1}$, then $|f - g| = |f_\ell|$.
- If $|f_\ell| > \frac{1}{2}\beta^{e_f-p+1}$, then $|\text{sign}(f_\ell)| = 1$ and Property 3 imply $|f - g| = \beta^{e_f-p+1} - |f_\ell|$.

Hence, in both cases $|f - g| < \frac{1}{2}\beta^{e_f-p+1}$.

Otherwise, we have $|g - f| = \frac{1}{2}\beta^{e_f-p+1}$, and we have to prove that the significand of g is even. According to Corollary 1, we can write $g = M_g\beta^{e_f-p+1}$ with $\beta^{e_f} \leqslant |M_g| \leqslant \beta^{e_f+1}$ and $M_g = 2h + c_j\beta^{e_j-(e_f-p+1)} + c$, where $c \in \{0, \text{sign}(f_\ell)\}$ and

$$h := \sum_{i=1}^{j-1} c_i \tfrac{\beta}{2}\beta^{e_i-(e_f-p+1)-1}.$$

For $i < j$, $e_i(k) > e_j(k)$ as $k \to \infty$, hence $h \in \mathbb{S}_{\geqslant 0}$, and the parity of M_g is the same as the parity of $c_j\beta^{e_j-(e_f-p+1)} + c$.

- If $e_j = e_f - p + 1$ and c_j is odd, then $c = \text{sign}(f_\ell)$ and $c_j\beta^{e_j-(e_f-p+1)} + c = c_j + c$ is even so that M_g is even.
- Otherwise, since $e_j(k) > e_f(k) - p(k) + 1$ as $k \to \infty$, $c = 0$ and β is even, we also have M_g even.

If $|M_g| < \beta^p$, then the significand of g is $|M_g|$; otherwise, the significand of g is β^{p-1} and the result follows. □

Corollary 2. *Given $p \in \mathbb{L}_{\geqslant 2}$ and $f \in \mathbb{S}$, Algorithm 2 returns $g \in \mathbb{SF}_p$ such that $g = \text{RN}_p(f)$, with RN_p defined by (12).*

Proof. If $f = 0$, the algorithm computes 0. We now assume that $f \neq 0$. We denote by g the output of the algorithm and $\widehat{f} = \mathrm{RN}_p(f)$. We know from Property 7 that $g \in \mathbb{SF}_p$, $\beta^{e_f} \leqslant |g| \leqslant \beta^{e_f+1}$, and by definition of $\widehat{f}$, we have $|\widehat{f} - f| \leqslant |g - f|$. We analyze separately the two possible cases for $|g - f|$.

If $|g-f| < \frac{1}{2}\beta^{e_f-p+1}$ then, using the triangular inequality, $|g-\widehat{f}| \leqslant |g-f| + |f-\widehat{f}| \leqslant 2|g-f|$ so $|g-\widehat{f}| < \beta^{e_f-p+1}$. Moreover, since RN_p is nondecreasing, $\beta^{e_f} \leqslant |\widehat{f}|$ and the result follows from Property 4.

Otherwise, $|g - f| = \frac{1}{2}\beta^{e_f-p+1}$ and the significand of g is even. We deduce from the same application of the triangular inequality that $|\widehat{f}-f| = |g-f|$, that is $g \in \arg\min_{h\in\mathbb{SF}_p} |h-f|$. Since the significand of g is even, we have $\widehat{f} = g$. □

Corollary 3. *Given $p \in \mathbb{L}_{\geqslant 2}$ and $f \in \mathbb{S}$, if $\widehat{f} = \mathrm{RN}_p(f)$, then $\widehat{f}(k) = \mathrm{RN}_{p(k)}(f(k))$ as $k \to \infty$.*

Proof. If $f = 0$, then $\widehat{f} = 0$ and the result is true. We now assume $f \neq 0$. Then, from Property 7, there exists $k_0 \in \mathbb{N}$ such that for all $k \geqslant k_0$, $\widehat{f}(k)$ is a floating-point number in precision $p(k)$, and we have $\beta^{e_f(k)} \leqslant |\widehat{f}(k)| \leqslant \beta^{e_f(k)+1}$, and $|\widehat{f}(k) - f(k)| \leqslant \frac{1}{2}\,\mathrm{ulp}(f(k))$ with equality iff the significand of $\widehat{f}(k)$ is even, hence $\widehat{f}(k) = \mathrm{RN}_{p(k)}(f(k))$. □

3.3 Symbolic floating-point division

In this subsection, we are interested in the division of two symbolic floating-point numbers $f, g \in \mathbb{SF}_p$ with $g \neq 0$. Unlike the previous operations $+$, $-$ and $\times$, the function f/g, defined for k large enough, does not generally belong to $\mathbb{S}$. We cannot compute the exact division in $\mathbb{S}$ and then round it to $\mathbb{SF}_p$. The algorithm we implemented for this division is based on the Taylor expansion of $1/(1-z)$. This method is related to Newton-Raphson iteration for division (see for instance Chapter 8 in [7]). The algorithm is valid under the following assumptions on $g = \sum_{i=1}^{n} c_i\beta^{e_i}$:

$$|c_1| = 1 \quad \text{and} \quad (n = 1 \text{ or } a_1 > a_2). \tag{13}$$

Note that $g \neq 0$ implies $n \geqslant 1$.

Algorithm 3 (Symbolic floating-point division)

algorithm div(f, g, p)
  if $n = 1$ **then** $h \leftarrow \mathrm{RN}_p(f) \times c_1\beta^{-e_1}$;
  else
    $e_f \leftarrow \mathrm{exponent}(f)$;
    $F \leftarrow f \times \beta^{-e_f}$;
    $N \leftarrow \lfloor 2a/(a_1 - a_2) \rfloor$; // $p = ak + b$ and $e_i = a_i k + b_i$
    $M \leftarrow -2p - 2$;
    $z \leftarrow -\sum_{i=2}^{n} \frac{c_i}{c_1}\beta^{e_i-e_1}$;
    $S \leftarrow F \times \sum_{i=0}^{N} z^i$;
    $(S_h, S_\ell) \leftarrow \mathrm{split}(S, M)$;
    $h \leftarrow c_1\beta^{e_f-e_1} \times \mathrm{RN}_p(S_h)$;

Theorem 1. *Given $p \in \mathbb{L}_{\geqslant 2}$ and $f, g \in \mathbb{SF}_p$ with $g \neq 0$ as in (13), Algorithm 3 returns $h \in \mathbb{SF}_p$ such that*

$$h(k) = \mathrm{RN}_{p(k)}(f(k)/g(k)) \text{ as } k \to \infty. \tag{14}$$

Proof. If $n = 1$, then $|c_1| = 1$ implies $\mathrm{RN}_{p(k)}(f(k)/g(k)) = c_1\beta^{-e_1(k)}\mathrm{RN}_{p(k)}(f(k))$ as $k \to \infty$ and (14) follows. If $f = 0$ then it is easily checked that $h = 0$, so that (14) holds. Hence, we now assume that $n \geqslant 2$ and $f \neq 0$. Assumption (13) implies that N is well defined. Moreover, z is built so that $g = c_1\beta^{e_1}(1 - z)$. If we define

$$\tau(k) = \frac{F(k)}{1 - z(k)} \text{ as } k \to \infty,$$

then $f(k)/g(k) = c_1\beta^{e_f(k)-e_1(k)}\tau(k)$ as $k \to \infty$ and we recover $\mathrm{RN}_{p(k)}(f(k)/g(k))$ by multiplying $\mathrm{RN}_{p(k)}(\tau(k))$ and $c_1\beta^{e_f(k)-e_1(k)}$. Considering the last assignment in Algorithm 3, it is enough to prove that

$$\mathrm{RN}_{p(k)}(S_h(k)) = \mathrm{RN}_{p(k)}(\tau(k)) \text{ as } k \to \infty. \tag{15}$$

Since e_f denotes the exponent of f, $1 \leqslant |F| < \beta$. Moreover, from Property 7 and $a_1 > a_2$, we have $|z(k)| < \beta^{e_2(k)-e_1(k)+1} < 1/\beta$ as $k \to \infty$. We deduce that $1/\beta \leqslant |\tau(k)| < \beta^2$ as $k \to \infty$. The Taylor series expansion of $1/(1 - z(k))$ as $k \to \infty$ gives $\tau(k) = S_h(k) + S_\ell(k) + \tau(k)z(k)^{N+1}$.

We now prove that

$$|\tau(k) - S_h(k)| < \beta^{M(k)} \text{ as } k \to \infty. \tag{16}$$

We have the two following inequalities:

- $|S_\ell| \leqslant \beta^{M-n}$ from Property 3;
- $|\tau(k)z(k)^{N+1}| < \beta^{2+(N+1)(e_2(k)-e_1(k)+1)}$ as $k \to \infty$.

Moreover, comparing the slopes, $2 + (N + 1)(e_2(k) - e_1(k) + 1) \leqslant M(k) - n$ as $k \to \infty$, and (16) follows.

To conclude, we define $e_\tau(k)$ as the (numerical) exponent of $\tau(k)$ whenever $\tau(k)$ is defined; we proved earlier that $-1 \leqslant e_\tau(k) \leqslant 1$ as $k \to \infty$. We split the analysis depending on whether or not $\tau(k)$ is a midpoint in precision $p(k)$.

If $\tau(k)$ is a midpoint, then we have $\tau(k) = (T + 1/2) \times \beta^{e_\tau(k)-p(k)+1}$, with $T \in \mathbb{Z}$. Since β is even and $M(k) < -p(k) - 1 \leqslant e_\tau - p(k)$, we deduce that $\tau(k) = T'\beta^{M(k)}$ with $T' \in \mathbb{Z}$. The equality $S_h(k) = \tau(k)$ then follows from (16) and Property 3, so that (15) holds.

If $\tau(k)$ is not a midpoint, then we know from [8] in base 2, and more generally from [9] in base $\beta \geqslant 2$, that the minimal distance between $\tau(k)$ and a midpoint is greater than $\beta^{-2p(k)-2} = \beta^{M(k)}$. Hence, if m_1 and m_2 are the two consecutive midpoints such that $\tau(k) \in (m_1, m_2)$, then $S_h(k) \in (m_1, m_2)$ and (15) follows. □

4 Implementation and experiments

In sections 2 and 3, we defined the arithmetic over the symbolic data in $\mathbb{S}$ and $\mathbb{SF}_p$ as $k \to \infty$. In practice, we want to compute a lower bound for k from which this asymptotic behavior is reached. Here, we explain how we implemented this aspect. We then describe the features of our library, and give two examples to illustrate its use.

4.1 Practical handling of the asymptotic behavior

To compute a lower bound from which the asymptotic behavior is reached, every element of $\mathbb{S}$ is given as in (2a), with an additional $k_0 \in \mathbb{N}$ realizing (2c), that is,

$$e_1(k) > e_2(k) > \cdots > e_n(k) \quad \text{for all } k \geqslant k_0.$$

Moreover, for any operation on elements of $\mathbb{S}$, given with their own k_0, the result is also given with a new k_0 realizing the expected behavior of the function.

We also chose in our implementation to force the new value of k_0, coming with the result, to be greater than or equal to any k_0 given as input. For example, if f and k_0 are given by (2) with $n \geqslant 1$, the exponent function computes a pair $(e, k_0') \in \mathbb{L} \times \mathbb{N}$ such that $k_0' \geqslant k_0$ and $\beta^{e(k)} \leqslant |f(k)| < \beta^{e(k)+1}$ for all $k \geqslant k_0'$. Hence, we ensure a nondecreasing behavior for the k_0's, which implies that the final k_0 obtained after a sequence of operations guarantees the asymptotic behavior to be reached for every operation in this sequence.

All the updates of k_0 come from the algorithm comparing two exponents in $\mathbb{L}$. It is then the building block for the computation of a valid k_0, together with the nondecreasing behavior of k_0 we enforce. When comparing two exponents $a_1k + b_1$ and $a_2k + b_2$ in $\mathbb{L}$, either $a_1 = a_2$ and the ordering is valid for every value of k, or $a_1 \neq a_2$ and $k_0 = \lfloor (b_2 - b_1)/(a_1 - a_2) \rfloor + 1$ realizes the ordering.

4.2 Description of the library

In our Maple library, every element of $\mathbb{S}$ (and thus also of its subset $\mathbb{SF}_p$) is represented by a structure of the form $\mathrm{EP}(L, \beta, k, k_0)$ where:

- L is a list of pairs (c_i, e_i), sorted in decreasing order with respect to the e_i's. As in (2) and (5), the c_i's are integers such that $|c_i| \in \{0, \ldots, \beta - 1\}$, and the e_i's are Maple expressions of the form $a_ik + b_i$.
- β is the basis in which the element of $\mathbb{S}$ is expressed.
- k is the symbolic variable.
- k_0 is a nonnegative integer used to ensure that the asymptotic behavior is reached.

The library contains the functions listed below.

- `neg`, `add`, `sub`, `mul`: perform the arithmetic operations on the ring $\mathbb{S}$, as described in subsection 2.2.
- `fma`: computes the fused multiply-add exactly; given $a, b, c \in \mathbb{S}$, it returns $a \times b + c$.
- `sign`, `abs`, `exponent`: compute the sign, absolute value, and exponent as presented in 2.3.
- `cmp`: given $f_1, f_2 \in \mathbb{S}$, returns $\mathrm{sign}(f_1 - f_2)$, which enables the comparisons between elements of $\mathbb{S}$.
- `round`: implements the round-to-nearest function from $\mathbb{S}$ to $\mathbb{SF}_p$ according to Algorithm 2.

- `div`: performs floating-point division using Algorithm 3.

We also implemented some conversion functions between our data type (EP), a Maple expression (expr) as in (2a), and the representation by a Laurent polynomial (poly).

- `EP2expr` and `expr2EP`: conversion from the EP data type to a Maple expression is straightforward; the reciprocal operation requires decompositions in base β to convert terms such as 5×2^p into $2^{p+2} + 2^p$.
- `poly2expr` and `expr2poly`: implement the change of variable $X = \beta^k$ in both ways. They take as additional parameters the base β and the variables k and X.

4.3 Examples

We give below two examples to illustrate the use of our library in Maple. The first example is about the computation of a two-by-two determinant with Kahan's algorithm (see Algorithm 1); we run this algorithm on a set of inputs parametrized by the precision p in (1) for which the relative error is equivalent to $2u$ as $u \to 0$ (or $p \to \infty$ while β is fixed). It provides a certificate of asymptotic optimality for the relative error bound $2u$.

The second example concerns the algorithm CompDivS given in [4] to compute a complex floating-point division, that is an approximation $\widehat{x} + i\,\widehat{y}$ of $(a + ib)/(c + id)$ where all variables are taking floating-point values. The authors are interested in the componentwise relative error. Example 8 in that paper provides inputs parametrized by the precision for which the computation of the real part leads to a relative error equivalent to the a priori bound of $5u$. In this example, we use the division algorithm and an even precision.

4.3.1 Kahan's algorithm

The algorithm is implemented as follows, using the add, sub, mul, and round functions from our Maple library:

```
> Kahan := proc(a, b, c, d, p)
>   wh := round(mul(b, c), p);
>   e := fma(neg(b), c, wh);
>   fh := round(fma(a, d, neg(wh)), p);
>   xh := round(add(fh, e), p);
>   return xh;
> end proc:
```

Then we set the value of β, the input variables as Maple expressions parametrized by p as in (1), and we compute an expression for the exact result `x`:

```
> beta := 10:
> a := beta^(p-1)+1:
> b := a:
> c := 2*beta^(p-1)+(beta/2)*beta^(p-2):
> d := beta^(p-1)+(beta/2)*beta^(p-2):
> x := simplify(a*d - b*c);
  x := 10^(2p-2) + 10^(p-1)
```

The inputs are then converted into the data structure using the expr2EP function. For instance, for a, this gives:

```
> A := expr2EP(a, beta, p);
  A := EP([[1,p-1], [1,0]], 10, p, 2)
```

Next we apply the symbolic version of Kahan's algorithm to get the approximate result `Xh`, and we convert it into a Maple expression `xh`:

```
> Xh := Kahan(A, B, C, D, p);
> xh := EP2expr(Xh);
  Xh := EP([[1,2p-2]], 10, p, 3)
  xh := 10^(2p-2)
```

The computation of `xh` by hand is rather tedious and occupies half a page in [3]. Moreover, `Xh` indicates that the computed result is valid for $p \geqslant 3$, and using a numerical evaluation, it can be checked that for $p = 2$ the algorithm returns 120 while `xh` $= 100$. We now convert `x` and `xh` into their polynomial representations using the change of variable $X = \beta^p$, and we express the error with respect to u, using $X = \beta/(2u)$.

```
> xp := expr2poly(x, beta, p, X);
> xhp := expr2poly(xh, beta, p, X);
> err := simplify(subs(X=beta/(2u), (xp-xhp)/xp));
  xp := X^2/100 + X/10
  xhp := X^2/100
  err := 2*u/(1+2*u)
```

These computations double-check the certificate of optimality for the relative error bound of Kahan's algorithm for $\beta = 10$.

4.3.2 Complex floating-point division

This example illustrates the use of Algorithm 3 and how to handle the case of an even precision. Since p is even, we set $p = 2k$:

```
> beta := 2: p := 2*k:
> a := beta^p-5*beta^(p/2-1):
> b := -beta^(p/2) + 5/beta - 3*beta^(-p/2):
> c := beta^p - beta:
> d := beta^(3*p/2)+beta^p:
```

As previously mentioned, a, b, c, d are converted respectively into A, B, C, D using expr2EP. We compute the exact result `r` and the approximate one `Rh` using the algorithm provided in [4]; `Rh` is then converted to a Maple expression.

```
> r := (a*c+b*d)/(c^2+d^2):
> Dh := round(fma(C, C, round(mul(D, D), p)), p):
> Gh := Kahan(A, neg(B), D, C, p):
> Rh := div(Gh, Dh, p);
> rh := EP2expr(Rh);
  Rh := EP([[-1, -3*k], [-1, -4*k-1]], 2, k, 8)
  rh := -8^(-k)-(1/2)*16^(-k)
```

The pen-and-paper computation of `rh` is more involved than the previous case, especially for rounding the quotient. We know from `Rh` that the computed result is valid for $k \geqslant 8$. In fact, it can be checked by numerical evaluations for $k = 3, \ldots, 7$ that the algorithm and the expression `rh` produce the same values. Hence, `rh` is valid for all $k \geqslant 3$.

Next, we compute the relative error and express it as a rational function in $X = \beta^k$.

```
> rel := (rh - r)/r:
> num := expr2poly(numer(rel), beta, k, X):
> den := expr2poly(denom(rel), beta, k, X):
> err := num/den;
  err := (10*X^5+2*X^4-8*X^3-4*X^2+8*X+4)
         /(2*X^7+5*X^6-4*X^5)
```

Now we express this error with respect to the unit roundoff u using the change of variable $X = \sqrt{\beta/(2u)}$ and compute an equivalent as $u \to 0$.

```
> erru := subs(X=sqrt(beta/(2u)), err):
> assume(u > 0):
> series(subs(X = sqrt(beta/(2*u)), errX), u=0, 3);
```

This gives $5u - \frac{23}{2}u^{3/2} + \mathcal{O}(u^2)$ when $u \to 0$, as in [4].

5 Conclusion and perspectives

As we are writing these lines, our library is still at an early development stage, but we plan to make it publicly available soon. Although the code is not designed with high performance as the primary target, the library checks 27 examples similar to the ones given in subsection 4.3, taken from [2, 3, 4, 10] (23 in base 2, and 4 in base 10) in less than 1.5 second of CPU time on a recent laptop, which we consider as sufficiently fast for our purposes. Other rounding modes can be easily be implemented, and they will be available in future versions of our library. We have shown that it is possible to perform the division of two symbolic floating point numbers when the denominator has a well suited shape. A natural extension of this work will be to extend the range of denominators for which we are able to give a result. Another extension will be to incorporate the square root function into the library, at least in some particular cases. We hope that this work will make easier the analysis of small yet important building blocks of numerical computing.

References

[1] IEEE Computer Society, *IEEE Standard for Floating-Point Arithmetic*. IEEE Standard 754-2008, Aug. 2008, available at http://ieeexplore.ieee.org/servlet/opac?punumber=4610933.

[2] R. Brent, C. Percival, and P. Zimmermann, "Error bounds on complex floating-point multiplication," *Mathematics of Computation*, vol. 76, pp. 1469–1481, 2007.

[3] C.-P. Jeannerod, N. Louvet, and J.-M. Muller, "Further analysis of Kahan's algorithm for the accurate computation of 2×2 determinants," *Mathematics of Computation*, vol. 82, pp. 2245–2264, 2013.

[4] ——, "On the componentwise accuracy of complex floating-point division with an FMA," in *21st IEEE Symposium on Computer Arithmetic, Austin, TX, USA*, 2013, pp. 83–90.

[5] G. Everest, A. van der Poorten, I. Shparlinski, and T. Ward, *Recurrence sequences*, ser. Mathematical Surveys and Monographs. Providence, RI: American Mathematical Society, 2003, vol. 104.

[6] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Transactions on Electronic Computers*, vol. 10, pp. 389–400, 1961.

[7] P. Markstein, *IA-64 and elementary functions: speed and precision.* Prentice Hall, 2000.

[8] C. Iordache and D. W. Matula, "On infinitely precise rounding for division, square root, reciprocal and square root reciprocal," in *14th IEEE Symposium on Computer Arithmetic, Adelaide, Australia*, 1999, pp. 233–240.

[9] N. Brisebarre and J.-M. Muller, "Correct rounding of algebraic functions," *Theoretical Informatics and Applications*, vol. 41, pp. 71–83, 2007.

[10] J.-M. Muller, "On the error of computing $ab + cd$ using Cornea, Harrison and Tang's method," *ACM Transactions on Mathematical Software*, vol. 41, no. 2, pp. 7:1–7:8, Feb. 2015.