On the computation of the reciprocal of floating point expansions using an adapted Newton-Raphson iteration

Mioara Joldes
CNRS, lab. LAAS
7 Avenue du Colonel Roche,
31077 Toulouse, France,
joldes@laas.fr

Jean-Michel Muller
CNRS, lab. LIP,
46 Allée d'Italie,
69364 Lyon, France
jean-michel.muller@ens-lyon.fr

Valentina Popescu
Lab. LAAS-CNRS
7 Avenue du Colonel Roche,
31077 Toulouse, France,
valentina.popescu@univ-tlse3.fr

***Abstract*—Many numerical problems require a higher computing precision than that offered by common floating point (FP) formats. One common way of extending the precision is to represent numbers in a *multiple component* format. With so-called *floating point expansions*, numbers are represented as the unevaluated sum of standard machine precision FP numbers. This format offers the simplicity of using directly available and highly optimized FP operations and is used by multiple-precisions libraries such as Bailey's QD or the analogue Graphics Processing Units tuned version, GQD. In this article we present a new algorithm for computing the reciprocal FP expansion a^{-1} of a FP expansion a. Our algorithm is based on an adapted Newton-Raphson iteration where we use "truncated" operations (additions, multiplications) involving FP expansions. The thorough error analysis given shows that our algorithm allows for computations of very accurate quotients. Precisely, after $i \geqslant 0$ iterations, the computed FP expansion $x = x_0 + \ldots + x_{2^i-1}$ satisfies the relative error bound: $\left|\frac{x-a^{-1}}{a^{-1}}\right| \leqslant 2^{-2^i(p-3)-1}$, where $p > 2$ is the precision of the FP representation used ($p = 24$ for single precision and $p = 53$ for double precision).**

Keywords*—*floating-point arithmetic, floating-point expansions, high precision arithmetic, multiple-precision arithmetic, division, Newton-Raphson iteration

I. Introduction

Many numerical problems in dynamical systems or planetary orbit dynamics, such as the long-term stability of the solar system [1], finding sinks in the Henon Map [2], iterating Lorenz attractor [3], etc., require higher precisions than the standard double precision (now called *binary64* [4]). Quad or higher precision is rarely implemented in hardware, and the most common solution is to use software emulated higher precision libraries, also called arbitrary precision libraries. There are mainly two ways of representing numbers in extended precision. The first one is the *multiple-digit representation*: numbers are represented with a sequence of digits coupled with a single exponent. An example is the representation used in GNU MPFR [5] which is an open-source C library, which provides besides arbitrary precision, also correct rounding for each atomic operation (for basic operations and functions). The second way is *multiple terms representation*: a number is expressed as an unevaluated sum of several standard FP numbers. This sum is usually called an *FP expansion*. Bailey's library QD [6] supports double-double (DD) and quad-double (QD) computations, that is, numbers are represented as the unevaluated sum of 2 or 4 double-precision FP numbers. It is known [7], however, that the DD and QD formats and operations implemented in this library are not compliant with the IEEE 754-2008 standard, and do not provide correctly rounded operations. However, this multiple term format offers the simplicity of using directly available and highly optimized FP operations and most multiple terms algorithms are straightforwardly portable to highly parallel architectures, such as GPUs. In consequence, there is a demand for providing error bounds for algorithms based on the multiple-term format. Addition and multiplication have been previously studied [7], [6, Thm. 44, Chap.

14]. In this article we consider the case of computing reciprocals of FP expansions. There are two classes of algorithms for performing division: the so-called *digit-recurrence algorithms* [8], that generalize the paper-and-pencil method, and the algorithms based on the Newton-Raphson iteration [9], [10]. While the algorithms suggested so far for dividing expansions belong to the former class, here we will be interested in studying the possible use of the latter class, since its very fast, quadratic, convergence is appealing when high precision is at stake.

The outline of the paper is the following: in Section I-A we recall some basic notions about FP expansions and in Section I-B we present existing algorithms based on long classical division on expansions. Then, in Section II the new algorithm for computing reciprocals of expansions is given and its correctness is proved. Finally, in Section III we assess the performance of our algorithm – in terms of number of FP operations, proven accuracy bounds and timings –.

A. Floating-point expansions

A normal binary precision-p floating-point (FP) number is a number of the form

$$x = M_x \cdot 2^{e_x - p + 1},$$

with $2^{p-1} \leqslant |M_x| \leqslant 2^p - 1$. The integer e_x is called the *exponent* of x, and $M_x \cdot 2^{-p+1}$ is called the *significand* of x. We denote accordingly to Goldberg's definition $\mathrm{ulp}(x) = 2^{e_x - p + 1}$ [7, Chap. 2].

A natural extension of the notion of DD or QD is the notion of *floating-point expansion*. A *floating-point expansion* u with n terms is the unevaluated sum of n floating-point numbers $u_0, u_1, \ldots, u_{n-1}$, in which all nonzero terms are ordered by magnitude (i.e., $u_i \neq 0 \Rightarrow |u_i| \geqslant |u_{i+1}|$). Arithmetics on FP expansions have been introduced by Priest [11], and later on by Shewchuk [12].

To make sure that such an expansion carries significantly more information than one FP number only, it is required that the u_i's do not "overlap". This notion of overlapping varies depending on the authors. We give here the definition of Priest and Bailey that we consider in the following, using the above-introduced notation.

Definition I.1. Assuming x and y are normal numbers with representations $M_x \cdot 2^{e_x - p + 1}$ and $M_y \cdot 2^{e_y - p + 1}$ (with $2^{p-1} \leqslant M_x, M_y \leqslant 2^p - 1$), they are *$\mathcal{P}$-nonoverlapping* (that is, nonoverlapping according to Priest's definition [13]) if $|e_y - e_x| \geqslant p$.

Definition I.2. An expansion is *$\mathcal{P}$-nonoverlapping* (that is, nonoverlapping according to Priest's definition [13]) if all of its components are mutually $\mathcal{P}$-nonoverlapping.

A slightly stronger sense of *nonoverlapping* was introduced by Hida, Li and Bailey [6]:

Definition I.3. An expansion $u_0, u_1, \ldots, u_{n-1}$ is *$\mathcal{B}$-nonoverlapping* (that is, nonoverlapping according to Bailey's definition [6]) if for all $0 < i < n$, we have $|u_i| \leqslant \frac{1}{2}\,\mathrm{ulp}(u_{i-1})$.

Remark I.4. Note that for *$\mathcal{P}$-nonoverlapping* expansions we have $|u_i| \leqslant \frac{2^p - 1}{2^p}\,\mathrm{ulp}(u_{i-1})$.

The majority of algorithms performing arithmetic operations on expansions are based on the so-called *error-free transforms* (such as the algorithms 2Sum, Fast2Sum, Dekker product and 2MultFMA presented for instance in [7]), that make it possible to compute both the result and the error of a FP addition or multiplication. This implies that in general, each such *error-free transform* applied to two FP numbers, returns *still two* FP numbers. So, an algorithm that performs addition of two expansions x and y with n and respectively m terms, returns an FP expansion with at most $n+m$ terms. Similarly, for multiplication, the product is an FP expansion with at most $2nm$ terms [11]. So-called normalization algorithms are used to render the result non-overlapping, which implies also a potential reduction in the number of components.

In what follows we denote by *RoundAddE*($x[0 : n-1]$, $y[0 : m-1]$, k), an algorithm for expansions addition, which given two ($\mathcal{P}-$ or $\mathcal{B}-$) nonoverlapping expansions, returns the k most significant terms of the exact normalized ($\mathcal{P}-$ or $\mathcal{B}-$) nonoverlapping sum. If no request is made on the number of terms to be returned, then we denote simply by *AddE*($x[0 : n-1]$, $y[0 : m-1]$). Similarly, we denote by *RoundMulE*, *MulE*, *RoundSubE*, *SubE*, *RenormalizeE* algorithms for multiplication, subtraction, normalization. While many variants exist for these algorithms, division and hence, *reciprocal* are less studied in literature.

B. Algorithms using classical long division on expansions

In [11] division is done using the classical long division algorithm, which is recalled in Al-

gorithm 1.

Figure 1. Priest's [11] division algorithm. We denote by $f[0:\ldots]$ and expansion f whose number of terms is not known in advance.

Input: FP expansion $a = a_0 + \ldots + a_{n-1}$; $b = b_0 + \ldots + b_{m-1}$; length of output quotient FP expansion d.

Output: FP expansion $q = q_0 + \ldots$ with at most d terms s.t. $\left|\frac{q-a/b}{a/b}\right| < 2^{1-\lfloor (p-4)d/p \rfloor}$.

1: $q_0 = \mathrm{RN}(a_0/b_0)$
2: $r^{(0)}[0:n-1] \leftarrow r[0:n-1]$
3: **for** $i \leftarrow 1$ to $d-1$ **do**
4: $f[0:\ldots] \leftarrow \mathit{MulE}(q_{i-1}, b[0:m-1])$
5: $r^{(i)}[0:\ldots] \leftarrow \mathit{RenormalizeE}(\mathit{SubE}(r^{(i-1)}[0:\ldots], f[0:\ldots]))$
6: $q_i = \mathrm{RN}(r_0^{(i)}/b_0)$
7: **end for**
8: $q[0:\ldots] \leftarrow \mathit{RenormalizeE}(q[0:d-1])$
9: **return** FP expansion $q = q_0 + \ldots$.

Bailey's division algorithm [6] is very similar. For instance, let $a = a_0 + a_1 + a_2 + a_3$ and $b = b_0 + b_1 + b_2 + b_3$ be QD numbers. First, one approximates the quotient $q_0 = a_0/b_0$, then compute the remainder $r = a - q_0 b$ in quad-double. The next correction term is $q_1 = r_0/b_0$. Subsequent terms q_i are obtained by continuing this process. Note that at each step when computing r full quad-double multiplication and subtraction must be performed since most of the bits will be cancelled out when computing q_3 and q_4. A renormalization step is performed only at the end, on $q_0+q_1+q_2+...$ in order to ensure non-overlapping. No error bound is given in [6].

Note that in Algorithm 1 [11] a renormalization step is performed after each computation of $r = r - q_i b$ and an error bound given:

Proposition I.5. *Consider two* $\mathcal{P}$-nonoverlapping *expansions:* $a = a_0 + \ldots + a_{n-1}$ *and* $b = b_0 + \ldots + b_{m-1}$, *Priest division algorithm [11] computes a quotient expansion* $q = q_0 + \ldots + q_{d-1}$ *s.t.*

$$\left|\frac{q - a/b}{a/b}\right| < 2^{1-\lfloor (p-4)d/p \rfloor}. \tag{1}$$

In Daumas and Finot's paper [14], Priest's division algorithm is improved by using only estimates of the most significant component of the remainder r_0 and storing the less significant components of the remainder and the terms $-q_i b$ unchanged in a set that is managed with a priority queue. While the asymptotic complexity of this algorithm is better, in practical simple cases Priest's algorithm is faster due to the control overhead of the priority queue [14]. The error bound obtained with Daumas' algorithm is (using the same notations as above):

$$\left|\frac{q - a/b}{a/b}\right| < 2^{-d(p-1)} \prod_{i=0}^{d-1} (4i+6). \tag{2}$$

II. Reciprocal of expansions with an adapted Newton-Raphson iteration

The classical Newton-Raphson iteration for computing reciprocals is briefly recalled in what follows [9], [10], [7, Chap. 2]. It is based on the classical Newton iteration for computing the roots of a given function f, which is $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$. When x_0 is close to a root α, $f'(\alpha) \neq 0$, the iteration converges quadratically. For computing $1/a$ we look for roots of the function $f(x) = 1/x - a$ which implies using the iteration $x_{n+1} = x_n(2 - a x_n)$. The iteration converges to $1/a$ for all $x_0 \in (0, 2/a)$, but for a fast convergence we need x_0 to be close to $1/a$. The quadratic convergence is proven by $x_{n+1} - \frac{1}{a} = -a(x_n - \frac{1}{a})^2$. This iteration is *self-correcting* because minor errors, like rounding errors, do not modify the limit value of the convergence.

While the iteration is classical, in Algorithm 2 we use an adaptation to computing reciprocals of FP expansions, with truncated operations involving FP expansions. Our algorithm works with both $\mathcal{B}$- *and* $\mathcal{P}$-*nonoverlapping* FP expansions. For the sake of clarity we consider first the case of $\mathcal{B}$-*nonoverlapping* FP expansions, and then make the necessary adjustments for $\mathcal{P}$-*nonoverlapping* expansions in Proposition II.3.

A. Error analysis of Algorithm 2

In the following, let $a = a_0 + \ldots + a_{2^k-1}$ be a $\mathcal{B}$-*nonoverlapping* FP expansion with 2^k terms and $q \geqslant 0$. We will prove that our algorithm returns an approximation $x = x_0 + \ldots + x_{2^q-1}$ of $\frac{1}{a}$, in the form of a $\mathcal{B}$-*nonoverlapping* FP expansion with 2^q terms, such that

$$\left|x - \frac{1}{a}\right| \leqslant \frac{2^{-2^q(p-3)-1}}{|a|}. \tag{3}$$

We will first prove the following proposition:

Proposition II.1. *Consider a* $\mathcal{B}$-nonoverlapping *expansion* $u = u_0 + u_1 + \ldots + u_k$ *with* $k > 0$ *normal binary FP terms of precision* p. *Denote* $u^{(i)} = u_0 + u_1 + \cdots + u_i, i \geqslant 0$, *i.e. "a truncation" of* u *to* $i+1$ *terms. The following inequalities hold for* $0 \leqslant i \leqslant k$:

$$|u_i| \leqslant 2^{-ip} |u_0|, \tag{4}$$

$$\left|u - u^{(i)}\right| \leqslant 2^{-ip} |u| \frac{\eta}{1-\eta}, \tag{5}$$

$$\left(1 - 2^{-ip}\frac{\eta}{1-\eta}\right)|u| \leqslant \left|u^{(i)}\right| \leqslant \left(1 + 2^{-ip}\frac{\eta}{1-\eta}\right)|u|, \tag{6}$$

$$\left|\frac{1}{u} - \frac{1}{u_0}\right| \leqslant \frac{1}{|u|}\eta, \tag{7}$$

where

$$\eta = \sum_{j=0}^{\infty} 2^{(-j-1)p} = \frac{2^{-p}}{1-2^{-p}}.$$

Proof: By definition of a $\mathcal{B}$-*nonoverlapping* expansion and since for any normal binary FP number u_i, $\operatorname{ulp}(u_i) \leqslant 2^{-p+1}|u_i|$ we have $|u_i| \leqslant \frac{1}{2}\operatorname{ulp}(u_{i-1}) \leqslant 2^{-p}|u_{i-1}|$ and (4) follows by induction.

Consequently we have $|u - u_0| = |u_1 + u_2 + \cdots + u_k| \leqslant 2^{-p}|u_0| + 2^{-2p}|u_0| + \cdots + 2^{-kp}|u_0| \leqslant |u_0|\,\eta$. One easily observes that u and u_0 have the same sign. One possible proof is by noticing that $1 - \eta > 0$ and $-|u_0|\,\eta \leqslant u - u_0 \leqslant |u_0|\,\eta$. Suppose $u_0 > 0$, then $-u_0\eta \leqslant u - u_0 \leqslant u_0\eta$, and hence $u_0(1-\eta) \leqslant u \leqslant u_0(1+\eta)$ which implies $u > 0$. The case $u_0 < 0$ is similar. It follows that

$$\frac{|u|}{1+\eta} \leqslant |u_0| \leqslant \frac{|u|}{1-\eta}. \tag{8}$$

For (5) we use (8) together with:

$$\left|u - u^{(i)}\right| \leqslant \sum_{j=0}^{\infty} 2^{(-i-j-1)p}|u_0| \leqslant 2^{-ip}\eta\,|u_0|,$$

and (6) is a simple consequence of (5). Similarly, (7) follows from $\left|\frac{1}{u} - \frac{1}{u_0}\right| = \frac{1}{|u|}\left|\frac{u_0 - u}{u_0}\right| \leqslant \frac{1}{|u|}\eta$. ∎

Figure 2. Truncated Newton iteration based algorithm for reciprocal of an FP expansion.

Input: FP expansion $a = a_0 + \ldots + a_{2^k-1}$; length of output FP expansion 2^q.

Output: FP expansion $x = x_0 + \ldots + x_{2^q-1}$ s.t. $\left|x - \frac{1}{a}\right| \leqslant \frac{2^{-2^q(p-3)-1}}{|a|}$.

1: $x_0 = \mathrm{RN}(1/a_0)$
2: **for** $i \leftarrow 0$ to $q-1$ **do**
3: $\hat{v}[0 : 2^{i+1}-1] \leftarrow$ *MulRoundE*($x[0 : 2^i - 1]$, $a[0 : 2^{i+1}-1]$, 2^{i+1})
4: $\hat{w}[0 : 2^{i+1}-1] \leftarrow$ *SubRoundE*(2, $\hat{v}[0 : 2^{i+1}-1]$, 2^{i+1})
5: $x[0 : 2^{i+1}-1] \leftarrow$ *MulRoundE*($x[0 : 2^i - 1]$, $\hat{w}[0 : 2^{i+1}-1]$, 2^{i+1})
6: **end for**
7: **return** FP expansion $x = x_0 + \ldots + x_{2^q-1}$.

Proposition II.2. *Algorithm 2 is correct when run with* $\mathcal{B}$-nonoverlapping *expansions.*

Proof: The algorithm's input is $a = a_0 + a_1 + \cdots + a_{2^k-1}$ a non-overlapping FP expansion in which every term a_i is a normal binary FP number of precision p. Let $f_i = 2^{i+1} - 1$ and $a^{(f_i)} = a_0 + a_1 + \ldots + a_{f_i}$ i.e. "a truncation" of a to $f_i + 1$ terms, with $0 \leqslant i$.

For computing $1/a$ we use Newton iteration: $x_0 = \mathrm{RN}(1/a_0)$, $x_{i+1} = x_i(2 - a^{(f_i)}x_i)), i \geqslant 0$ by truncating each operation involving FP expansions in the following way:

- let $v_i := (a^{(f_i)} \cdot x_i)$ be the exact product represented as a non-overlapping FP expansion on $2^{2(i+1)}$ terms, we compute $\hat{v}_i := v_i^{(2^i)}$ i.e. v_i "truncated to" 2^{i+1} terms;
- let $w_i := 2 - \hat{v}_i$ be the exact result of the subtraction represented as a non-overlapping FP expansion on $2^{i+1}+1$ terms, we compute $\hat{w}_i := w_i^{(2^i)}$ i.e. v_i "truncated to" 2^{i+1} terms;
- let $\tau_i := x_i \cdot \hat{w}_i$ be the exact product represented as a non-overlapping FP expansion on $2 \cdot 2^i(2^{i+1})$ terms, we compute $x_{i+1} := \tau_i^{(2^{i+1}-1)}$ i.e. τ_i "truncated to" to 2^{i+1} terms.

Let us first prove a simple upper bound for the approximation error in x_0:

$$\varepsilon_0 = \left|x_0 - \frac{1}{a}\right| \leqslant \frac{2\eta}{|a|}. \tag{9}$$

Since $x_0 = \mathrm{RN}(1/a_0)$, then $\left|x_0 - \frac{1}{a_0}\right| \leqslant 2^{-p}\left|\frac{1}{a_0}\right|$, so $\left|x_0 - \frac{1}{a}\right| \leqslant 2^{-p}\left|\frac{1}{a_0}\right| + \left|\frac{1}{a} - \frac{1}{a_0}\right| \leqslant \frac{(1+\eta)2^{-p}+\eta)}{|a|} \leqslant \frac{2\eta}{|a|}$ (from (8)).

Let us deduce an upper bound for the approximation error in x at step $i+1$, $\varepsilon_{i+1} = \left|x_{i+1} - \frac{1}{a}\right|$. For this, we will use a chain of triangular inequalites that make the transition from our "truncated" Newton error, to the "untruncated" one. Let $\gamma_i = 2^{-(2^{i+1}-1)p}\frac{\eta}{1-\eta}$. We have from Proposition II.1, eq. (5):

$$|x_{i+1} - \tau_i| \leqslant \gamma_i |x_i \cdot \hat{w}_i|, \tag{10}$$

$$|w_i - \hat{w}_i| \leqslant \gamma_i |w_i| \leqslant \gamma_i |2 - \hat{v}_i|, \tag{11}$$

$$|v_i - \hat{v}_i| \leqslant \gamma_i \left|a^{(f_i)} \cdot x_i\right|, \tag{12}$$

$$\left|a - a^{(f_i)}\right| \leqslant \gamma_i |a|. \tag{13}$$

From (10) we have:

$$\begin{aligned}
\varepsilon_{i+1} &\leqslant |x_{i+1} - \tau_i| + \left|\tau_i - \frac{1}{a}\right| \\
&\leqslant \gamma_i |x_i \cdot \hat{w}_i| + \left|x_i \cdot \hat{w}_i - \frac{1}{a}\right| \\
&\leqslant \gamma_i |x_i(w_i - \hat{w}_i)| + \gamma_i |x_i w_i| + \left|x_i \cdot \hat{w}_i - \frac{1}{a}\right| \\
&\leqslant (1+\gamma_i)|x_i||w_i - \hat{w}_i| + \gamma_i |x_i w_i| \\
&+ \left|x_i \cdot w_i - \frac{1}{a}\right|.
\end{aligned}$$

Using (11) and (12):

$$\begin{aligned}
\varepsilon_{i+1} &\leqslant \left|x_i \cdot w_i - \frac{1}{a}\right| + ((1+\gamma_i)\gamma_i + \gamma_i)|x_i w_i| \\
&\leqslant \left|x_i \cdot (2 - v_i) - \frac{1}{a}\right| + |x_i| \cdot |(v_i - \hat{v}_i)| \\
&+ (\gamma_i(1+\gamma_i) + \gamma_i)|x_i|(|(2 - v_i)| + |v_i - \hat{v}_i|) \\
&\leqslant \left|x_i \cdot (2 - a^{(f_i)} \cdot x_i) - \frac{1}{a}\right| \\
&+ \gamma_i(1+\gamma_i)^2 \left|x_i^2\right| \left|a^{(f_i)}\right| \\
&+ (\gamma_i(1+\gamma_i) + \gamma_i)\left|x_i(2 - a^{(f_i)} \cdot x_i)\right|.
\end{aligned}$$

By (13), we have:

$$\left|x_i \cdot (2 - a^{(f_i)} \cdot x_i) - \frac{1}{a}\right| \leqslant |a|\left|x_i - \frac{1}{a}\right|^2 + \gamma_i |x_i|^2 |a|,$$

$$|x_i|^2 \left|a^{(f_i)}\right| \leqslant (1+\gamma_i)|x_i|^2|a|,$$

and

$$\left|x_i \cdot (2 - a^{(f_i)} \cdot x_i)\right| \leqslant |a|\left|x_i - \frac{1}{a}\right|^2 + \gamma_i |x_i|^2 |a| + \frac{1}{|a|}.$$

Hence we have:

$$\begin{aligned}
\varepsilon_{i+1} &\leqslant (1+\gamma_i)^2 |a| \left|x_i - \frac{1}{a}\right|^2 \\
&+ \gamma_i(1+\gamma_i)^2(2+\gamma_i)\left|x_i^2\right||a| \\
&+ \gamma_i(2+\gamma_i)\frac{1}{|a|}.
\end{aligned} \tag{14}$$

We now prove by induction that for all $i \geqslant 0$:

$$\varepsilon_i = \left|x_i - \frac{1}{a}\right| \leqslant \frac{2^{-2^i(p-3)-1}}{|a|}. \tag{15}$$

For $i = 0$, this holds from (9) and the fact that $\eta = \frac{1}{2^p - 1} \leqslant 2^{-p+1}$. For the induction step, we have from (14):

$$\begin{aligned}
\varepsilon_{i+1} &\leqslant (1+\gamma_i)^2 |a| |\varepsilon_i|^2 \\
&+ \gamma_i(1+\gamma_i)^2(2+\gamma_i)(1 \pm \varepsilon_i |a|)^2 \frac{1}{|a|} \\
&+ \gamma_i(2+\gamma_i)\frac{1}{|a|},
\end{aligned} \tag{16}$$

which implies

$$\begin{aligned}
\frac{|a|\,\varepsilon_{i+1}}{2^{-2^{i+1}(p-3)}} &\leqslant \frac{(1+\gamma_i)^2}{4} + \frac{(1+2^{-p+2})(2+\gamma_i)}{64} \\
&\cdot \left(1 + (1+\gamma_i)^2\left(1 + 2^{-2^i(p-3)-1}\right)^2\right) \\
&\leqslant \frac{1}{2}.
\end{aligned} \tag{17}$$

This completes our proof. ∎

Proposition II.3. *Algorithm 2 is correct when run with* $\mathcal{P}$-nonoverlapping *expansions.*

Proof: It is easy to see that the previous analysis holds, provided that we use Remark I.4. This implies mainly the following changes $\eta' = \frac{2}{2^p - 3}$,

$\gamma'_i = \left(\frac{2}{2^p-1}\right)^{2^{i+1}-1} \frac{\eta'}{1-\eta'}$. With this change it is easy to verify that equations (9)–(16) hold as soon as $p > 2$. Note that for the induction step at $i = 1$, a tighter bound is needed for $\varepsilon'_0 \leqslant \frac{2^{-p}(1+\eta')+\eta'}{|a|} \leqslant \frac{2\eta'}{|a|}\frac{3-2^{-p}}{4}$, but the rest of the proof is identical, safe for some tedious computations. ∎

B. Complexity analysis

As presented before, our algorithm has the feature of using "truncated" expansions, while some variants of *RoundAddE* and *RoundMulE* need to compute the result fully and only then truncate. However, for the sake of completeness, we first analyze the theoretical complexity of our algorithm in terms of number of FP operations, using Priest's multiplication and addition. These are not tuned for obtaining "truncated" expansions on the fly – and thus penalize our algorithm–, but are more general than Bailey's ones, which restrict to 2 or 4 terms. Secondly, we compare the actual implementation performance of our algorithm using Bailey's addition and multiplication.

Complexity analysis based on Priest's multiplication and addition: We recall (and slightly update by taking 6 FP operations for *2Sum* [7]) the original operation count of Priest [11]:
– Priest addition [11] of two $\mathcal{P}$-*nonoverlapping* expansions with n and respectively m terms, requires $n + m$ *2Sum* calls and $n + m + 1$ FP comparisons in worst case. This accounts for $A(n, m) = 7(n + m) + 1$ FP operations.
– Priest renormalization [11] of a $\mathcal{P}$-*nonoverlapping* expansion x with k terms, requires $3(k - 1)$ *2Sum* calls and $2(k - 1)$ comparisons in worst case. This accounts for $R(k) = 20(k - 1)$ FP operations.
– hence, Priest addition followed by renormalization [11] of two $\mathcal{P}$-*nonoverlapping* expansions with n and respectively m terms, requires $(n + m) + 3(n + m - 1)$ *2Sum* calls and $(n+m+1)+2(n+m-1)$ FP comparisons in worst case. This accounts for $AR(n, m) = 27(n+m)-19$ FP operations.
– Priest multiplication [11] of two $\mathcal{P}$-*nonoverlapping* expansions with n and respectively m terms, requires $n + 2m$ Veltkamp's "splits" [7, Chap.4], $6nm$ multiplications, followed by $\sum_{i=1}^{n}(6R(m) + \sum_{j=1}^{5} AR(jm, m) + AR(6m, 6(i - 1)m))$ calls. This accounts for $M(n, m) = 81\,mn^2 + 747\,nm + 2\,m - 233\,n$ FP operations.
– Using Priest algorithms for addition and multiplication of FP expansions, Priest division Algorithm 1 requires d divisions and $(d-1)(M(1, m))+\sum_{i=0}^{d-1} AR(2m, n+2m(i-1))+R(d)$ function calls in worst case. This accounts for $D(d, n, m) = 27\,d^2m + 803\,dm + 27\,dn - 231\,d - 830\,m + 213$ FP operations.

Proposition II.4. *Using Priest algorithms for addition and multiplication of FP expansions, Algorithm 2 requires* $-\frac{4461}{7} + 8\,q + 996 \cdot 4^q + \frac{324}{7} \cdot 8^q - 404 \cdot 2^q$ *FP operations.*

Proof: We observe that the ith iteration the following operations with expansions are performed: 2 multiplications $M(2^i, 2^{i+1})$; one addition and renormalization $AR(2^{i+1}, 1)$. Since q iterations are done, the total number of FP operations is: $1 + \sum_{i=0}^{q-1}(2M(2^i, 2^{i+1}) + AR(2^{i+1}, 1)) = -\frac{4461}{7} + 8\,q + 996 \cdot 4^q + \frac{324}{7} \cdot 8^q - 404 \cdot 2^q$ FP operations.

∎

Remark II.5. Division is simply performed with Algorithm 2 followed by a multiplication $M(2^q, n)$ where the numerator expansion has n terms.

III. Comparison and Discussion

Concerning the error bounds provided (when the same number of terms in the division are requested) our algorithm performs better. In Table I we show effective values for the bounds provided by our error analysis compared with those of Priest (see eq. (1)) and Daumas (see eq. (2)). Moreover, our algorithm provides a unified error bound with quadratic convergence independent of using underlying $\mathcal{P}$- or $\mathcal{B}$-*nonoverlapping* expansions.

Table I. Effective values for the bounds provided by our error analysis compared with those of Priest (see eq. (1)) and Daumas (see eq. (2))

Prec, iteration	Eq. (1)	Eq. (2)	Eq. (15)
$p = 53, i = 0$	2	2^{-49}	2^{-51}
$p = 53, i = 1$	1	2^{-98}	2^{-101}
$p = 53, i = 2$	2^{-2}	2^{-195}	2^{-201}
$p = 53, i = 3$	2^{-6}	2^{-387}	2^{-401}
$p = 53, i = 4$	2^{-13}	2^{-764}	2^{-801}
$p = 24, i = 0$	2	2^{-20}	2^{-22}
$p = 24, i = 1$	1	2^{-40}	2^{-43}
$p = 24, i = 2$	2^{-2}	2^{-79}	2^{-85}
$p = 24, i = 3$	2^{-5}	2^{-155}	2^{-169}
$p = 24, i = 4$	2^{-12}	2^{-300}	2^{-337}

The complexity analysis performed with $\mathcal{P}$-*nonoverlapping* expansions shows that when at

most d components of the quotient are needed and no error bound is requested, classical division performs better. Some effective values of the worst case operation count for the two algorithms, for the same number of expansion terms are given in Table II. Note that Priest algorithms are known to be slow in practice due to the renormalization process applied very often. For example, the multiplication of two FP expansions of length n and m is proportional to n^2m which is mainly the reason why our algorithm is slower.

Table II. Effective values of the worst case FP operation count for Algorithm 1 and Algorithm 2 used for reciprocal, when an expansion of at most d terms is obtained.

d	Flops Algorithm 1	Flops Algorithm 2
2	1573	2917
4	10653	16661
8	57157	83597
16	299829	437493

On the other hand, if a relative error is requested, then it is preferable to use Algorithm 2. For instance, to guarantee an error bound of $2^{-d(p-3)-1}$, Priest's algorithm (based on the bound given in Prop I.5) needs at least $(dp-3d+2)p/(p-4)$ terms, which entails a very poor complexity. This implies that Daumas' algorithm might be a good compromise in this case, provided that the priority queue used there can be efficiently implemented.

In practice however, Bailey's DD/QD libraries are very often used, so we compare timings in the setting of $\mathcal{B}$-*nonoverlapping expansions*. In Table III and IV we present effective timings for several precisions for reciprocal and division obtained with our algorithm vs. the QD library. For manipulating double-double numbers, the QD library is faster, but for quad-double and higher precisions our algorithm performs better. This confirms our hypothesis that for higher precisions the Newton-Raphson iteration is preferable than classical division. In Table V timings for precisions greater than quad are recorded. All tests were performed on an Intel(R) Core(TM) i7 CPU 3820, 3.6GHz computer.

As a future work we intend to generalize the theoretical analysis of DD and QD addition/multiplication algorithms and thus to be able to perform a more clinching complexity analysis for reciprocal of $\mathcal{B}$-*nonoverlapping* expansions also.

Table III. Timings in MFlops/s for Algorithm 2 vs. QD implementation for reciprocal of expansions. The input expansion has d_i and the output expansion has d_o double FP terms.

d_i	d_o	Algorithm 2	QD
1	1	10^7	10^7
2	2	62	70
4	4	10	3.6
1	2	62	86.2
2	4	10.7	3.7
4	2	61	86.2
1	4	12.6	7.36

Table IV. Timings in MFlops/s for Algorithm 2 vs. a QD implementation, where the numerator expansion has d_o terms, the denominator expansion has d_i and the output has d_o terms.

d_i	d_o	Algorithm 2	QD
1	1	10^7	10^7
2	2	46.3	70
4	4	6.8	3.6
1	2	46.7	86.2
2	4	7	3.7
4	2	46.1	86.2
1	4	7.7	7.36

Table V. Timings in MFlops/s for Algorithm 2 implementation for reciprocal of expansions. The input expansion has d_i and the output expansion has d_o double FP terms.

d_i	d_o	Algorithm 2
1	2	62
1	4	12.6
1	8	2
2	2	62
2	4	10.7
2	8	1.7
4	8	1.4
8	8	1.3
1	16	0.3
2	16	0.27
4	16	0.22
8	16	0.19
16	16	0.17

Acknowledgment

The authors would like to warmly thank Warwick Tucker for his great interest and many useful suggestions on this topic.

References

[1] J. Laskar and M. Gastineau, "Existence of collisional trajectories of Mercury, Mars and Venus with the Earth," *Nature*, vol. 459, no. 7248, pp. 817–819, Jun. 2009. [Online]. Available: http://dx.doi.org/10.1038/nature08096

[2] M. Joldes, V. Popescu, and W.Tucker, "Searching for sinks of Hénon map using a multiple-precision GPU arithmetic library," LAAS-CNRS No. 13492, LAAS, Tech. Rep., Nov 2013.

[3] A. Abad, R. Barrio, and A. Dena, "Computing periodic orbits with arbitrary precision," *Phys. Rev. E*, vol. 84, p. 016701, Jul 2011. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.84.016701

[4] 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.

[5] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann, "MPFR: A Multiple-Precision Binary Floating-Point Library with Correct Rounding," *ACM Transactions on Mathematical Software*, vol. 33, no. 2, 2007, available at http://www.mpfr.org/.

[6] Y. Hida, X. S. Li, and D. H. Bailey, "Algorithms for quad-double precision floating-point arithmetic," in *Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH-16)*, N. Burgess and L. Ciminiera, Eds., Vail, CO, Jun. 2001, pp. 155–162.

[7] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010, ACM G.1.0; G.1.2; G.4; B.2.0; B.2.4; F.2.1., ISBN 978-0-8176-4704-9.

[8] M. D. Ercegovac and T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Kluwer Academic Publishers, Boston, MA, 1994.

[9] T. J. Ypma, "Historical development of the Newton-Raphson method," *SIAM Rev.*, vol. 37, no. 4, pp. 531–551, Dec. 1995. [Online]. Available: http://dx.doi.org/10.1137/1037125

[10] M. Cornea, R. A. Golliver, and P. Markstein, "Correctness proofs outline for Newton–Raphson-based floating-point divide and square root algorithms," in *Proceedings of the 14th IEEE Symposium on Computer Arithmetic (Adelaide, Australia)*, Koren and Kornerup, Eds. IEEE Computer Society Press, Los Alamitos, CA, Apr. 1999, pp. 96–105.

[11] D. M. Priest, "Algorithms for arbitrary precision floating point arithmetic," in *Proceedings of the 10th IEEE Symposium on Computer Arithmetic (Arith-10)*, P. Kornerup and D. W. Matula, Eds. IEEE Computer Society Press, Los Alamitos, CA, Jun. 1991, pp. 132–144.

[12] J. R. Shewchuk, "Adaptive precision floating-point arithmetic and fast robust geometric predicates," *Discrete Computational Geometry*, vol. 18, pp. 305–363, 1997. [Online]. Available: http://link.springer.de/link/service/journals/00454/papers97/18n3p305.pdf

[13] D. M. Priest, "On properties of floating-point arithmetics: Numerical stability and the cost of accurate computations," Ph.D. dissertation, University of California at Berkeley, 1992.

[14] M. Daumas and C. Finot, "Division of floating point expansions with an application to the computation of a determinant," *Journal of Universal Computer Science*, vol. 5, no. 6, pp. 323–338, jun 1999.