An accurate algorithm for evaluating rational functions

Stef Graillat

An accurate algorithm for evaluating rational functions

Stef Graillat*

August 29, 2017

Abstract

Several different techniques intend to improve the accuracy of results computed in floating-point precision. Here, we focus on a method to improve the accuracy of the evaluation of rational functions. We present a compensated algorithm to evaluate rational functions. This algorithm is accurate and fast. The accuracy of the computed result is similar to the one given by the classical algorithm computed in twice the working precision and then rounded to the current working precision. This algorithm runs much more faster than existing implementation producing the same output accuracy.

Keywords: floating-point, error-free transformation, rational function, Horner scheme, accuracy, rounding errors

AMS Subject Classifications: 15-04, 65G99, 65-04

1 Introduction

Evaluating a polynomial or a rational function is ubiquitous in computational sciences and their applications. For example, in signal processing, transfer functions are very often rational functions. Moreover, real functions are often approximated by polynomials or rational functions.

In this paper, we present fast and accurate algorithms to compute the evaluation of a rational function. Our aim is to increase the accuracy at a fixed precision. We show that the results have the same error estimates as if computed in twice the working precision and then rounded to working precision. This paper was motivated by papers [14, 7, 6], where similar approaches are used to compute summation, dot product, and polynomial evaluation.

This outline of this article is as follows. In Section 2, we quickly recall some information on floating-point arithmetic and we give some definitions and notations used in the sequel. In Section 3, we recall the compensated Horner scheme [7, 6]. This algorithm makes it possible to evaluate a polynomial whose accuracy of the computed result is similar to the one given by the classical algorithm computed in twice the working precision and then rounded to the current working precision. Section 4 is devoted to the study of the accuracy of the classic algorithm to evaluate a rational function with Horner scheme. We also define and compute

*Sorbonne Universités, UPMC Univ Paris 06, CNRS, UMR 7606, LIP6, F-75005 Paris, France (stef.graillat@upmc.fr, http://www-pequan.lip6.fr/~graillat).

a closed formula for the condition number of rational function evaluation. A compensated algorithm for evaluating rational functions is presented in Section 5. This algorithm evaluates a fractional function and gives an accuracy of the computed result that is similar to the one given by the classical algorithm computed in twice the working precision and then rounded to the current working precision. Finally, numerical experiments showing the accuracy and the performance of our new compensated algorithm to evaluate fractional functions are presented in Section 6.

2 Floating-point arithmetic

Throughout the paper, we assume to work with a floating-point arithmetic adhering to IEEE 754 floating-point standard [9]. We assume that no overflow nor underflow occur. The set of floating-point numbers is denoted by $\mathbb{F}$, the relative rounding error by $\mathbf{u}$. For IEEE 754 double precision, we have $\mathbf{u} = 2^{-53}$ and for single precision $\mathbf{u} = 2^{-24}$.

We denote by $\mathrm{fl}(\cdot)$ the result of a floating-point computation, where all operations inside parentheses are done in floating-point working precision. Floating-point operations in IEEE 754 satisfy [8]

$$\mathrm{fl}(a \circ b) = (a \circ b)(1+\varepsilon_1) = (a \circ b)/(1+\varepsilon_2) \text{ for } \circ = \{+,-,\cdot,/\} \text{ and } |\varepsilon_v| \le \mathbf{u}.$$

This implies that

$$|a \circ b - \mathrm{fl}(a \circ b)| \le \mathbf{u}|a \circ b| \text{ and } |a \circ b - \mathrm{fl}(a \circ b)| \le \mathbf{u}|\mathrm{fl}(a \circ b)| \text{ for } \circ = \{+,-,\cdot,/\}. \tag{2.1}$$

We use standard notation for error estimations. The quantities γ_n are defined as usual [8] by

$$\gamma_n := \frac{n\mathbf{u}}{1-n\mathbf{u}} \quad \text{for } n \in \mathbb{N},$$

where we implicitly assume that $n\mathbf{u} \le 1$.

Following [8], we also use the following classic properties in error analysis (we always assume that $nu < 1$): $\gamma_k < \gamma_{k+1}$ and $(1+\mathbf{u})\gamma_k \le \gamma_{k+1}$.

One can notice that $a \circ b \in \mathbb{R}$ and $a \circledcirc b := \mathrm{fl}(a \circ b) \in \mathbb{F}$ but in general we do not have $a \circ b \in \mathbb{F}$. It is known that for the basic operations $+, -, \times$, the rounding error of a floating-point operation is still a floating-point number (see for example [3]):

$$\begin{array}{rcll} x = a \oplus b & \Rightarrow & a+b = x+y & \text{with } y \in \mathbb{F}, \\ x = a \ominus b & \Rightarrow & a-b = x+y & \text{with } y \in \mathbb{F}, \\ x = a \otimes b & \Rightarrow & a \times b = x+y & \text{with } y \in \mathbb{F}. \end{array} \tag{2.2}$$

These are *error-free* transformations of the pair (a,b) into the pair (x,y).

Fortunately, the quantities x and y in (2.2) can be computed exactly in floating-point arithmetic. For the algorithms, we use Matlab-like notations. For addition, we can use the following algorithm by Knuth [12, Thm B. p.236].

Algorithm 2.1 (Knuth [12])**.** Error-free transformation of the sum of two floating-point numbers

function $[x,y] = \texttt{TwoSum}(a,b)$
  $x = a \oplus b$
  $z = x \ominus a$
  $y = (a \ominus (x \ominus z)) \oplus (b \ominus z)$

Another algorithm to compute an error-free transformation is the following algorithm from Dekker [3]. The drawback of this algorithm is that we have $x + y = a + b$ provided that $|a| \geq |b|$.

Algorithm 2.2 (Dekker [3]). Error-free transformation of the sum of two floating-point numbers.

function $[x, y] = \texttt{FastTwoSum}(a, b)$
  $x = a \oplus b$
  $y = (a \ominus x) \oplus b$

For the error-free transformation of a product, we first need to split the input argument into two parts. Let p be given by $\mathbf{u} = 2^{-p}$ and define $s = \lceil p/2 \rceil$. For example, if the working precision is IEEE 754 double precision, then $p = 53$ and $s = 27$. The following algorithm by Dekker [3] splits a floating-point number $a \in \mathbb{F}$ into two parts x and y such that

$$a = x + y \quad \text{and} \quad x \text{ and } y \text{ nonoverlapping with } |y| \leq |x|.$$

Algorithm 2.3 (Dekker [3]). Error-free split of a floating-point number into two parts

function $[x, y] = \texttt{Split}(a)$
  $\texttt{factor} = 2^s + 1$
  $c = \texttt{factor} \otimes a$
  $x = c \ominus (c \ominus a)$
  $y = a \ominus x$

With this function, an algorithm from Veltkamp (see [3]) makes it possible to compute an error-free transformation for the product of two floating-point numbers. This algorithm returns two floating point numbers x and y such that

$$a \times b = x + y \quad \text{with } x = a \otimes b.$$

Algorithm 2.4 (Veltkamp [3]). Error-free transformation of the product of two floating-point numbers

function $[x, y] = \texttt{TwoProduct}(a, b)$
  $x = a \otimes b$
  $[a_1, a_2] = \texttt{Split}(a)$
  $[b_1, b_2] = \texttt{Split}(b)$
  $y = a_2 \otimes b_2 \ominus (((x \ominus a_1 \otimes b_1) \ominus a_2 \otimes b_1) \ominus a_1 \otimes b_2)$

The `TwoProduct` algorithm can be re-written in a very simple way if a Fused-Multiply-and-Add (`FMA`) operator is available on the targeted architecture [13, 2]. This means that for $a, b, c \in \mathbb{F}$, the result of $\texttt{FMA}(a, b, c)$ is the nearest floating-point number of $a \cdot b + c \in \mathbb{R}$. The `FMA` satisfies

$$\texttt{FMA}(a, b, c) = (a \cdot b + c)(1 + \varepsilon_1) = (a \cdot b + c)/(1 + \varepsilon_2) \text{ with } |\varepsilon_\nu| \leq \mathbf{u}.$$

Algorithm 2.5 (Ogita, Rump and Oishi [14]). Error-free transformation of the product of two floating-point numbers using an `FMA`.

function $[x, y] = \texttt{TwoProductFMA}(a, b)$
  $x = a \otimes b$
  $y = \texttt{FMA}(a, b, -x)$

3 Compensated Horner scheme

We recall hereafter the compensated algorithm for Horner scheme. One can find a more detailed description of the algorithm in [7, 6]. We first recall the classic algorithm for Horner scheme and give an error bound. We then present the compensated Horner scheme together with an error bound.

The classical method for evaluating a polynomial

$$p(x) = \sum_{i=0}^{n} a_i x^i$$

is the Horner scheme which consists in the following algorithm.

Algorithm 3.1. Polynomial evaluation with Horner's scheme

function res = Horner(p, x)
  $s_n = a_n$
  for $i = n-1 : -1 : 0$
    $s_i = s_{i+1} \otimes x \oplus a_i$
  end
  res $= s_0$

A forward error bound for the result of Algorithm 3.1 is (see [8, p.95]):

$$|p(x) - \texttt{res}| \leq \gamma_{2n} \sum_{i=0}^{n} |a_i||x|^i = \gamma_{2n} \widetilde{p}(|x|) \tag{3.3}$$

where $\widetilde{p}(x) = \sum_{i=0}^{n} |a_i| x^i$. It is very interesting to express and interpret this result in terms of the condition number of the polynomial evaluation defined by

$$\operatorname{cond}(p, x) := \lim_{\varepsilon \to 0} \sup \left\{ \frac{|p(x) - \widehat{p}(x)|}{\varepsilon |p(x)|} : |a_i - \widehat{a}_i| \leq \varepsilon |a_i|, i = 0, \ldots, n \right\}.$$

It is well-known that

$$\operatorname{cond}(p, x) = \frac{\sum_{i=0}^{n} |a_i||x|^i}{|p(x)|} = \frac{\widetilde{p}(|x|)}{|p(x)|}. \tag{3.4}$$

Thus we have

$$\frac{|p(x) - \texttt{res}|}{|p(x)|} \leq \gamma_{2n} \operatorname{cond}(p, x). \tag{3.5}$$

We can modify the Horner scheme to compute the rounding error at each elementary operation that are a sum and a product. This is done in Algorithm 3.2.

Algorithm 3.2 (Graillat, Langlois, Louvet [7, 6])**.** Polynomial evaluation with a compensated Horner's scheme

function res = CompHorner(p, x)
  $s_n = a_n$
  $r_n = 0$
  for $i = n-1 : -1 : 0$
    $[p_i, \pi_i]$ = TwoProduct(s_{i+1}, x)
    $[s_i, \sigma_i]$ = TwoSum(p_i, a_i)
    $r_i = r_{i+1} \otimes x \oplus (\pi_i \oplus \sigma_i)$
  end
  res $= s_0 \oplus r_0$

If we denote by p_π and p_σ the two following polynomials

$$p_\pi = \sum_{i=0}^{n-1} \pi_i x^i, \qquad p_\sigma = \sum_{i=0}^{n-1} \sigma_i x^i,$$

then one can show, thanks to error-free transformations that

$$p(x) = s_0 + p_\pi(x) + p_\sigma(x).$$

If one looks at the previous algorithm closely, it is then clear that $s_0 = \texttt{Horner}(p, x)$. As a consequence, we can derive a new error-free transformation for polynomial evaluation

$$p(x) = \texttt{Horner}(p, x) + p_\pi(x) + p_\sigma(x).$$

The compensated Horner scheme first computes $p_\pi(x) + p_\sigma(x)$ which corresponds to the rounding errors and then adds the obtained value to the result of the classic Horner scheme $\texttt{Horner}(p, x)$.

We will show that the results computed by Algorithm 3.2 admits significantly better error-bounds than those computed with the classical Horner scheme. We argue that Algorithm 3.2 provides results as if they were computed using twice the working precision. This is summed up in the following theorem.

Theorem 3.1 (Graillat, Langlois, Louvet[7, 6]). *Consider a polynomial p of degree n with floating-point coefficients, and a floating-point value x. The forward error in the compensated Horner algorithm is such that*

$$|\texttt{CompHorner}(p, x) - p(x)| \leq \mathbf{u}|p(x)| + \gamma_{2n}^2 \widetilde{p}(x). \tag{3.6}$$

It is interesting to interpret the previous theorem in terms of the condition number of the polynomial evaluation of p at x. Combining the error bound (3.6) with the condition number (3.4) for polynomial evaluation gives

$$\frac{|\texttt{CompHorner}(p, x) - p(x)|}{|p(x)|} \leq \mathbf{u} + \gamma_{2n}^2 \operatorname{cond}(p, x). \tag{3.7}$$

In other words, the bound for the relative error of the computed result is essentially γ_{2n}^2 times the condition number of the polynomial evaluation, plus the unavoidable term $\mathbf{u}$ for rounding the result to the working precision. In particular, if $\operatorname{cond}(p, x) < \gamma_{2n}^{-1}$, then the relative accuracy of the result is bounded by a constant of the order of $\mathbf{u}$. This means that the compensated Horner algorithm computes an evaluation accurate to the last few bits as long as the condition number is smaller than $\gamma_{2n}^{-1} \approx (2n\mathbf{u})^{-1}$. Besides that, (3.7) tells us that the computed result is as accurate as if computed by the classic Horner algorithm with twice the working precision, and then rounded to the working precision.

4 Classic evaluation of rational functions

In this Section, we present a classic algorithm to evaluate a rational function. It is based on the evaluation of the numerator and denominator (which are polynomials) with the Horner scheme. We then give a definition of the condition number of the rational functions evaluation and give an explicit formula to compute it. We then study the numerical stability of the algorithm in terms of the condition number.

Let p, q two polynomials of degree n (it is not complicated to deal with polynomials with different degrees but for simplicity, we assume they both have the same degree). They are denoted by

$$p(x) = \sum_{i=0}^{n} a_i x^i \qquad \text{and} \qquad q(x) = \sum_{i=0}^{n} b_i x^i.$$

The rational fraction $f(x)$ is $f(x) = p(x)/q(x)$. A classic way to compute $f(x)$ is via the evaluation of $p(x)$ and $q(x)$ with Horner scheme as explained in Algorithm 4.1

Algorithm 4.1. Rational function evaluation with Horner scheme

function res = RatEval(p, q, x)
  res = Horner$(p, x) \oslash$ Horner(q, x)

The condition number of the evaluation of a rational function measures the sensitivity of the evaluation with respect to perturbations on the coefficients of the rational function. It can be defined as follows.

Definition 4.1. Let $f(x) = p(x)/q(x)$ be a rational function. The condition number of the evaluation of f in x is defined by

$$\operatorname{cond}(f, x) := \lim_{\varepsilon \to 0} \sup \left\{ \left| \frac{(\widehat{p}/\widehat{q})(x) - (p/q)(x)}{\varepsilon (p/q)(x)} \right| : |\widehat{a}_i - a_i| \le \varepsilon |a_i|, |\widehat{b}_j - b_j| \le \varepsilon |b_j| \text{ for } i = 1, \ldots, n, j = 1, \ldots, m \right\}.$$

where $\widehat{a}_i$ and $\widehat{b}_i$ are respectively the coefficients of $\widehat{p}$ and $\widehat{q}$.

It is possible to obtain an explicit expression for this condition number.

Theorem 4.1. *Let $f(x) = p(x)/q(x)$ be a rational function. The condition number of the evaluation of f at x satisfies*

$$\operatorname{cond}(f, x) = \operatorname{cond}(p, x) + \operatorname{cond}(q, x).$$

Proof. It is easy to show that

$$\begin{aligned} \frac{(\widehat{p}/\widehat{q})(x) - (p/q)(x)}{\varepsilon (p/q)(x)} &= \frac{\widehat{p}(x)(q(x) - \widehat{q}(x)) - \widehat{q}(x)(p(x) - \widehat{p}(x))}{\varepsilon p(x) \widehat{q}(x)}, \\ &= \frac{1}{\varepsilon} \left[\frac{\widehat{p}(x)}{p(x)} \cdot \frac{q(x) - \widehat{q}(x)}{\widehat{q}(x)} - \frac{p(x) - \widehat{p}(x)}{p(x)} \right]. \end{aligned}$$

As a consequence, we have

$$\left| \frac{(\widehat{p}/\widehat{q})(x) - (p/q)(x)}{\varepsilon (p/q)(x)} \right| \le \frac{1}{\varepsilon} \left[\left| \frac{\widehat{p}(x)}{p(x)} \right| \cdot \frac{|q(x) - \widehat{q}(x)|}{|\widehat{q}(x)|} + \frac{|p(x) - \widehat{p}(x)|}{|p(x)|} \right].$$

By definition of p and $\widehat{p}$, we have

$$|p(x) - \widehat{p}(x)| \le \varepsilon \widetilde{p}(|x|) \quad \text{and} \quad |q(x) - \widehat{q}(x)| \le \varepsilon \widetilde{q}(|x|).$$

It follows that

$$\left| \frac{(\widehat{p}/\widehat{q})(x) - (p/q)(x)}{\varepsilon (p/q)(x)} \right| \le \left| \frac{\widehat{p}(x) q(x)}{p(x) \widehat{q}(x)} \right| \cdot \operatorname{cond}(q, x) + \operatorname{cond}(p, x).$$

By taking the supremum and the limit for $\varepsilon \to 0$, we obtain

$$\operatorname{cond}(f, x) \le \operatorname{cond}(p, x) + \operatorname{cond}(q, x).$$

To show that we, indeed, have equality, let us define $\widehat{p}(x) = p(x) + \varepsilon \sum_{i=0}^{n} \operatorname{sign}(a_i x^i) a_i x^i$ and $\widehat{q}(x) = q(x) - \varepsilon \sum_{i=0}^{n} \operatorname{sign}(b_i x^i) b_i x^i$. In that case, we have

$$\frac{(\widehat{p}/\widehat{q})(x) - (p/q)(x)}{\varepsilon (p/q)(x)} = \frac{\widehat{p}(x) q(x)}{p(x) \widehat{q}(x)} \cdot \frac{\widetilde{q}(|x|)}{|q(x)|} + \frac{\widetilde{p}(|x|)}{|p(x)|}.$$

Taking the supremum and the limit for $\varepsilon \to 0$ proves that it is possible to construct a sequence of polynomials such the limit in the definition of the condition number converges to $\operatorname{cond}(p, x) + \operatorname{cond}(q, x)$. □

It is now possible to evaluate the numerical stability of the classic algorithm. It is shown in Theorem 4.2 that this algorithm is backward-stable.

Theorem 4.2. *Let $f(x) = p(x)/q(x)$ be a rational function with floating-point coefficients, and x be a floating-point value. Then if no underflow occurs, and* `res` $=$ `RatEval`(p, q, x),

$$\frac{|\texttt{res} - f(x)|}{|f(x)|} \leq \mathbf{u} + [\gamma_{2n} + \mathcal{O}(\mathbf{u}^2)]\,\mathrm{cond}(f, x).$$

Proof. We have

$$\left|\mathrm{fl}\left(\frac{p(x)}{q(x)}\right) - \frac{p(x)}{q(x)}\right| = \left|(1+\varepsilon)\frac{\mathrm{fl}(p(x))}{\mathrm{fl}(q(x))} - \frac{p(x)}{q(x)}\right|$$

with $|\varepsilon| \leq \mathbf{u}$. As a consequence,

$$\left|\mathrm{fl}\left(\frac{p(x)}{q(x)}\right) - \frac{p(x)}{q(x)}\right| \leq \mathbf{u}\left|\frac{\mathrm{fl}(p(x))}{\mathrm{fl}(q(x))}\right| + \left|\frac{\mathrm{fl}(p(x))}{\mathrm{fl}(q(x))} - \frac{p(x)}{q(x)}\right|. \tag{4.8}$$

Thanks to Equation (3.3), we know that

$$\mathrm{fl}(p(x)) \leq p(x) + \gamma_{2n}\widetilde{p}(|x|) \quad \text{and} \quad q(x) - \gamma_{2n}\widetilde{q}(|x|) \leq \mathrm{fl}(q(x)).$$

We can then deduce that

$$\begin{aligned}\left|\frac{\mathrm{fl}(p(x))}{\mathrm{fl}(q(x))}\right| &\leq \left|\frac{p(x) + \gamma_{2n}\widetilde{p}(|x|)}{q(x) - \gamma_{2n}\widetilde{q}(|x|)}\right| = |f(x)|\left|\frac{1 + \gamma_{2n}\,\mathrm{cond}(p, x)}{1 - \gamma_{2n}\,\mathrm{cond}(q, x)}\right|, \\ &\leq |f(x)|(1 + \gamma_{2n}\,\mathrm{cond}(f, x) + \mathcal{O}(\mathbf{u}^2)).\end{aligned} \tag{4.9}$$

Moreover, using Equation (3.5), we have

$$\begin{aligned}\left|\frac{\mathrm{fl}(p(x))}{\mathrm{fl}(q(x))} - \frac{p(x)}{q(x)}\right| &= \left|\frac{q(x)\,\mathrm{fl}(p(x)) - p(x)\,\mathrm{fl}(q(x))}{q(x)\,\mathrm{fl}(q(x))}\right|, \\ &= \left|\frac{[q(x) - \mathrm{fl}(q(x))]\,\mathrm{fl}(p(x)) - [p(x) - \mathrm{fl}(q(x))]\,\mathrm{fl}(q(x))}{q(x)\,\mathrm{fl}(q(x))}\right|, \\ &\leq \left|\frac{\mathrm{fl}(p(x))}{\mathrm{fl}(q(x))}\right| \cdot \left|\frac{q(x) - \mathrm{fl}(q(x))}{q(x)}\right| + \left|\frac{p(x)}{q(x)}\right| \cdot \left|\frac{p(x) - \mathrm{fl}(p(x))}{p(x)}\right|, \\ &\leq \left|\frac{\mathrm{fl}(p(x))}{\mathrm{fl}(q(x))}\right| \gamma_{2n}\,\mathrm{cond}(q, x) + \left|\frac{p(x)}{q(x)}\right| \gamma_{2n}\,\mathrm{cond}(p, x)|.\end{aligned}$$

Now, using Equations (4.8) and (4.9), we can deduce that

$$\begin{aligned}|\,\mathrm{fl}(f(x)) - f(x)| \leq \mathbf{u}|f(x)|(1 + \gamma_{2n}\,\mathrm{cond}(f, x) + \mathcal{O}(\mathbf{u}^2)) + \\ |f(x)|(1 + \gamma_{2n}\,\mathrm{cond}(f, x) + \mathcal{O}(\mathbf{u}^2))\gamma_{2n}\,\mathrm{cond}(q, x) + |f(x)|\gamma_{2n}\,\mathrm{cond}(p, x),\end{aligned}$$

that can be simplified into

$$|\,\mathrm{fl}(f(x)) - f(x)| \leq |f(x)| \cdot [\mathbf{u} + (\gamma_{2n} + \mathcal{O}(\mathbf{u}^2))\,\mathrm{cond}(f, x)],$$

which concludes the proof. □

5 A compensated algorithm for evaluating rational functions

In this Section, we present a compensated version of the classic algorithm for evaluating rational functions. The idea is to replace the Horner scheme used to evaluate the numerator and denominator by the compensated Horner scheme followed by a floating-point division. This is Algorithm 5.1.

Algorithm 5.1. Rational function evaluation with compensated Horner scheme

function res = CompRatEval(p, q, x)
  res = CompHorner$(p, x) \oslash$ CompHorner(q, x)

The following Theorem shows that the error bound on the accuracy of the computed result given by the compensated algorithm is improved compared to the one for the classic algorithm.

Theorem 5.1. *Let $f(x) = p(x)/q(x)$ be a rational function with floating-point coefficients, and x be a floating-point value. Then if no underflow occurs, and* res = CompFracEval(p, q, x)*,*

$$\frac{|\texttt{res} - f(x)|}{|f(x)|} \leq 3\mathbf{u} + \mathcal{O}(\mathbf{u}^2) + [2\gamma_{2n+1}^2 + \mathcal{O}(\mathbf{u}^3)]\,\mathrm{cond}(f, x). \tag{5.10}$$

Proof. For better readability, we will denote $\mathrm{CH}(p, x)$ the computed result of CompHorner(p, x). By definition we have

$$|\texttt{CompRatEval}(p, q, x) - f(x)| = \left|\mathrm{fl}\left(\frac{\mathrm{CH}(p, x)}{\mathrm{CH}(q, x)}\right) - f(x)\right| = \left|(1 + \varepsilon)\frac{\mathrm{CH}(p, x)}{\mathrm{CH}(q, x)} - f(x)\right|$$

with $|\varepsilon| \leq \mathbf{u}$. As a consequence,

$$|\texttt{CompRatEval}(p, q, x) - f(x)| \leq \mathbf{u}\left|\frac{\mathrm{CH}(p, x)}{\mathrm{CH}(q, x)}\right| + \left|\frac{\mathrm{CH}(p, x)}{\mathrm{CH}(q, x)} - f(x)\right|. \tag{5.11}$$

Moreover, using Equation (3.7), we have

$$|\,\mathrm{CH}(p, x)| \leq |p(x)|(1 + \mathbf{u}) + \gamma_{2n}^2\widetilde{p}(|x|) \quad \text{and} \quad (1 - \mathbf{u})|q(x)| - \gamma_{2n}^2\widetilde{q}(|x|) \leq |\,\mathrm{CH}(q, x)|.$$

As a consequence, we can bound

$$\begin{aligned}\left|\frac{\mathrm{CH}(p, x)}{\mathrm{CH}(q, x)}\right| &\leq \left|\frac{p(x)}{q(x)}\right| \cdot \frac{1 + \mathbf{u} + \gamma_{2n}\,\mathrm{cond}(p, x)}{1 - \mathbf{u} - \gamma_{2n}\,\mathrm{cond}(q, x)}, \\ &\leq |f(x)| \cdot [1 + \mathbf{u} + \gamma_{2n}\,\mathrm{cond}(p, x)] \cdot [1 + \mathbf{u} + \gamma_{2n}\,\mathrm{cond}(p, x) + \mathcal{O}(\mathbf{u}^3)], \\ &\leq |f(x)| \cdot [(1 + \mathbf{u})^2 + \gamma_{2n}^2\,\mathrm{cond}(f, x) + \mathcal{O}(\mathbf{u}^3)]. \end{aligned} \tag{5.12}$$

Moreover,

$$\begin{aligned}\left|\frac{\mathrm{CH}(p, x)}{\mathrm{CH}(q, x)} - f(x)\right| &= \left|\frac{\mathrm{CH}(p, x)[q(x) - \mathrm{CH}(q, x)] - \mathrm{CH}(q, x)[p(x) - \mathrm{CH}(p, x)]}{q(x)\,\mathrm{CH}(q, x)}\right|, \\ &\leq \left|\frac{\mathrm{CH}(p, x)}{\mathrm{CH}(q, x)}\right| \cdot \left|\frac{q(x) - \mathrm{CH}(q, x)}{q(x)}\right| + \left|\frac{p(x)}{q(x)}\right| \cdot \left|\frac{p(x) - \mathrm{CH}(p, x)}{p(x)}\right|, \\ &\leq \left|\frac{\mathrm{CH}(p, x)}{\mathrm{CH}(q, x)}\right| (\mathbf{u} + \gamma_{2n}^2\,\mathrm{cond}(q, x)) + |f(x)|(\mathbf{u} + \gamma_{2n}^2\,\mathrm{cond}(p, x)).\end{aligned}$$

Using Equation (5.12), we deduce that

$$\left|\frac{\mathrm{CH}(p,x)}{\mathrm{CH}(q,x)} - f(x)\right| \le |f(x)| \cdot [(1+\mathbf{u})^2 + \gamma_{2n}^2 \operatorname{cond}(f,x) + \mathcal{O}(\mathbf{u}^3)] \cdot (\mathbf{u} + \gamma_{2n}^2 \operatorname{cond}(q,x)) + |f(x)|(\mathbf{u} + \gamma_{2n}^2 \operatorname{cond}(p,x)).$$

which can be simplified into

$$\left|\frac{\mathrm{CH}(p,x)}{\mathrm{CH}(q,x)} - f(x)\right| \le |f(x)| \cdot [2\mathbf{u}(1+\mathbf{u})^2 + \gamma_{2n+1}^2 \operatorname{cond}(f,x) + \mathcal{O}(\mathbf{u}^3)]. \tag{5.13}$$

Finally combining Equations (5.13) and (5.12) with Equation (5.11), we obtain

$$|\texttt{CompRatEval}(p,q,x) - f(x)| \le \mathbf{u}|f(x)| \cdot [(1+\mathbf{u})^2 + \gamma_{2n}^2 \operatorname{cond}(f,x) + \mathcal{O}(\mathbf{u}^3)] + |f(x)| \cdot [2\mathbf{u}(1+\mathbf{u})^2 + \gamma_{2n+1}^2 \operatorname{cond}(f,x) + \mathcal{O}(\mathbf{u}^3)].$$

that can be rewritten as

$$|\texttt{CompRatEval}(p,q,x) - f(x)| \le |f(x)|[3\mathbf{u}(1+\mathbf{u})^2 + (2\gamma_{2n+1}^2 + \mathcal{O}(\mathbf{u}^3))\operatorname{cond}(f,x)].$$

and so concludes the proof. □

In other words, the bound for the relative error of the computed result is essentially $2\gamma_{2n+1}^2$ times the condition number of the rational function evaluation, plus the term $3\mathbf{u}$ for rounding the result to the working precision. In particular, if $\operatorname{cond}(f,x) < (2\gamma_{2n+1})^{-1}$, then the relative accuracy of the result is bounded by a constant of the order of $3\mathbf{u}$. This means that the compensated algorithm computes an evaluation accurate to the last few bits as long as the condition number is smaller than $(2\gamma_{2n+1})^{-1} \approx ((4n+2)\mathbf{u})^{-1}$. Besides that, (5.10) tells us that the computed result is as accurate as if computed by the classic rational function evaluation algorithm with twice the working precision, and then rounded to the working precision.

6 Numerical experiments

The numerical experiments have been done on a laptop with an Intel Core i5 processor at 2.9 GHz with 16 Gb of RAM. We used MATLAB R2016b.

We test the rational function $f_n(x) = p_n(x)/q_n(x)$ where p_n is a random polynomial of degree n and q_n is the expand form of the polynomial $(x-1)^n$. The argument x is chosen near to the unique real root 1 of q_n, and with many significant bits so that a lot of rounding errors occur during the evaluation of $q_n(x)$. We increment the degree n from 1 until a sufficiently large range has been covered by the condition number $\operatorname{cond}(f_n,x)$. Here we have

$$\operatorname{cond}(f_n,x) = \operatorname{cond}(p_n,x) + \frac{\widetilde{q_n}(|x|)}{|q_n(x)|} = \operatorname{cond}(p_n,x) + \left|\frac{1+x}{1-x}\right|^n,$$

for x close to 1 and $\operatorname{cond}(f_n,x)$ grows exponentially with respect to n. In the experiments reported on Figure 1, $\operatorname{cond}(f_n,x)$ varies from 10^2 to 10^{35} (for $x = \mathrm{fl}(1.333)$, that corresponds to the degree range $n = 3,\ldots,42$). These huge condition numbers have some meaning since here the coefficients of p_n and q_n and the value x are chosen to be exact floating-point numbers.

We experiment both `RatEval` and `CompRatEval`. For each rational function f_n, the exact value $f_n(x)$ is approximate with a high accuracy thanks to the Symbolic Math Toolbox of MATLAB. Figure 1 presents the relative accuracy $|y - f_n(x)|/|f_n(x)|$ of the evaluation y computed by the two algorithms.

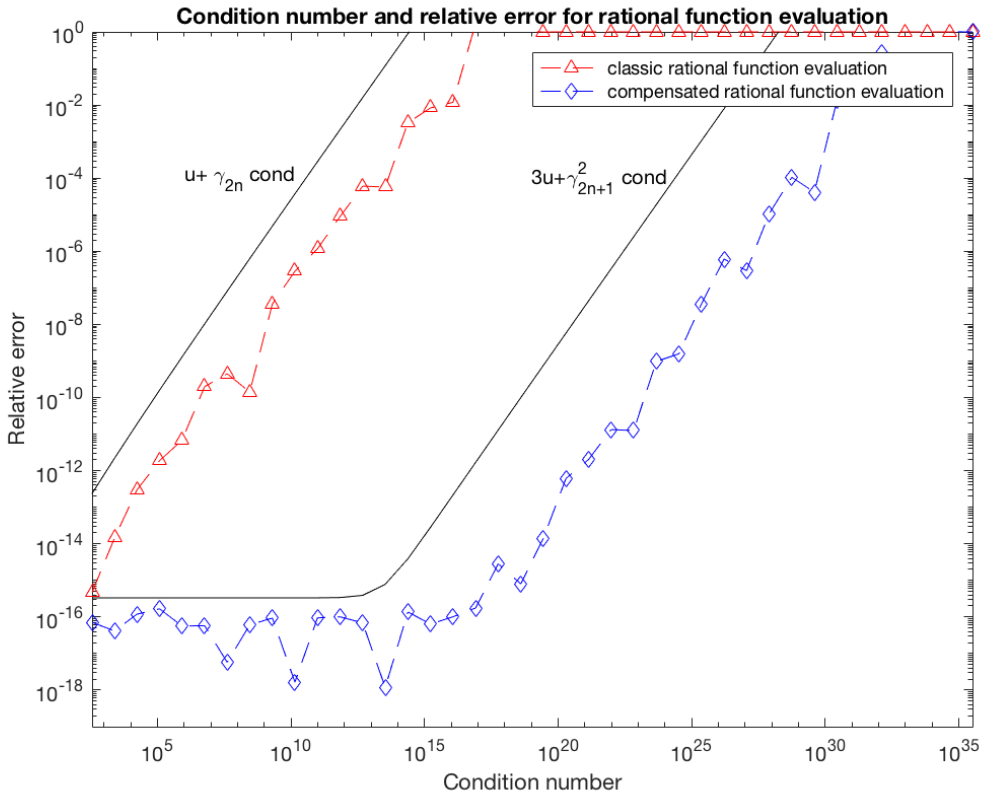

Figure 1: Comparison of the accuracy of `RatEval` and `CompRatEval`

We observe that the compensated algorithm exhibits the expected behavior. The full precision solution is computed as long as the condition number is smaller than $\mathbf{u}^{-1} \approx 10^{16}$. Then, for condition numbers between $\mathbf{u}^{-1}$ and $\mathbf{u}^{-2} \approx 10^{32}$, the relative error degrades to no accuracy at all.

We now demonstrate the practical efficiency in terms of running time comparing our algorithm and up-to-date challenger.

Since Bailey's double-double are usually considered as the most efficient portable library to double the IEEE-754 double precision, we consider it as a reference in the comparisons. Double-double numbers are represented as an unevaluated sum of a leading double and a trailing double. More precisely, a double-double number a is the pair (a_h, a_l) of floating-point numbers with $a = a_h + a_l$ and $|a_l| \leq \mathbf{u}|a_h|$.

In the sequel, we present two algorithms to compute product of two double-double or a double times a double-double. Those algorithms are taken from [1].

Algorithm 6.1. Product of the double-double number (a_h, a_l) by the double number b

function $[c_h, c_l] = \texttt{prod\_dd\_d}(a_h, a_l, b)$
  $[s_h, s_l] = \texttt{TwoProduct}(a_h, b)$
  $[t_h, t_l] = \texttt{FastTwoSum}(s_h, (a_l \otimes b))$
  $[c_h, c_l] = \texttt{FastTwoSum}(t_h, (t_l \oplus s_l))$

Algorithm 6.2. Addition of the double number b to the double-double number (a_h, a_l)

function $[c_h, c_l] = \texttt{add\_dd\_d}(a_h, a_l, b)$
  $[t_h, t_l] = \texttt{TwoSum}(a_h, b)$
  $[c_h, c_l] = \texttt{FastTwoSum}(t_h, (t_l \oplus a_l))$

The double-double library can be used to implement an Horner scheme in quadruple precision like `DDHorner`.

Algorithm 6.3. Horner scheme with internal double-double computations

function `res` = `DDHorner`(p, x)
  $s_h = a_n$
  $s_l = 0$
  for $i = n - 1 : -1 : 0$
    $[p_h, p_l]$ = `prod_dd_d`(s_h, s_l, x)
    $[s_h, s_l]$ = `add_dd_d`(p_h, p_l, a_i)
  end
  `res` = s_h

With `DDHorner`, we can evaluate a rational function in quadruple precision.

Algorithm 6.4. Rational function evaluation with double-double Horner scheme

function `res` = `DDRatEval`(p, q, x)
  `res` = `DDHorner`$(p, x) \oslash$ `DDHorner`(q, x)

We have implemented the three algorithms `RatEval`, `CompRatEval`, and `DDRatEval` in a C code to measure their overhead compared to the `RatEval` algorithm. We have programmed these tests straightforwardly with no other optimization than the ones performed by the compiler.

For each algorithm, we measured the ratio of its computing time over the computing time of the classic rational function evaluation algorithm. It turned out that our compensated algorithm `CompRatEval` is about 3 times slower than the classic Horner scheme. The same slowdown factor is about 10 for algorithm `DDRatEval`. From a practical point of view, we can state that our algorithm is about 5 times faster than the algorithm with double-doubles.

We compared `RatEval`, `CompRatEval` and `DDRatEval` in term of measured computing time. We tested with random rational functions where the degree of numerator and denominator vary from 100 to 100000.

Table 1: Measured computing times with `RatEval` normalised to 1.0

n	`RatEval`	`CompRatEval`	`DDRatEval`
100	1.0	2.0	10.0
500	1.0	1.6	7.9
1000	1.0	1.7	8.2
10000	1.0	1.6	8.2
100000	1.0	1.6	8.5

7 Conclusion

We presented a fast algorithm for the evaluation of rational functions in floating-point arithmetic. We have proved that the accuracy of the result computed by our compensated algorithm is similar to the one given by the classic algorithm performed in doubled working precision. The only assumption we made is that the floating-point arithmetic available on the computer is conformed to the IEEE-754 Standard. Its low requirement make it highly portable, and our compensated algorithm could be easily integrated into numerical

libraries. Our algorithm uses only basic floating-point operations, and only the same working precision as the data. Finally, our compensated algorithm runs much more faster than existing implementation producing the same output accuracy. This approach can easily be generalized to rational functions where numerators and denominators are written in other basis and for bivariate rational functions (see [5, 4, 10, 11] for example).

Acknowledgement

This work was partly supported by the project FastRelax ANR-14-CE25-0018-01.

References

[1] D. H. Bailey. *A Fortran-90 double-double library*, 2001. Available at URL = http://crd-legacy.lbl.gov/~dhbailey/mpdist/.

[2] 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.

[3] T. J. Dekker. A floating-point technique for extending the available precision. *Numer. Math.*, 18:224–242, 1971.

[4] P. Du, H. Jiang, and L. Cheng. Accurate evaluation of polynomials in legendre basis. *J. Applied Mathematics*, 2014:742538:1–742538:13, 2014.

[5] P. Du, H. Jiang, H. Li, L. Cheng, and C. Yang. Accurate evaluation of bivariate polynomials. In *17th International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT 2016, Guangzhou, China, December 16-18, 2016*, pages 51–56, 2016.

[6] S. Graillat, Ph. Langlois, and N. Louvet. Algorithms for accurate, validated and fast polynomial evaluation. *Japan J. Indust. Appl. Math.*, 2-3(26):191–214, 2009. Special issue on State of the Art in Self-Validating Numerical Computations.

[7] S. Graillat, N. Louvet, and Ph. Langlois. Compensated Horner scheme. Research Report 04, Équipe de recherche DALI, Laboratoire LP2A, Université de Perpignan Via Domitia, France, 52 avenue Paul Alduy, 66860 Perpignan cedex, France, July 2005.

[8] N. J. Higham. *Accuracy and stability of numerical algorithms*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second edition, 2002.

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

[10] H. Jiang, R. Barrio, H. Li, X. Liao, L. Cheng, and F. Su. Accurate evaluation of a polynomial in chebyshev form. *Applied Mathematics and Computation*, 217(23):9702–9716, 2011.

[11] H. Jiang, S. Li, L. Cheng, and F. Su. Accurate evaluation of a polynomial and its derivative in bernstein form. *Computers & Mathematics with Applications*, 60(3):744–755, 2010.

[12] D. E. Knuth. *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*. Addison-Wesley, Reading, MA, USA, third edition, 1998.

[13] Y. Nievergelt. Scalar fused multiply-add instructions produce floating-point matrix arithmetic provably accurate to the penultimate digit. *ACM Trans. Math. Software*, 29(1):27–48, 2003.

[14] T. Ogita, S. M. Rump, and S. Oishi. Accurate sum and dot product. *SIAM J. Sci. Comput.*, 26(6):1955–1988, 2005.