Applied Mathematics and Computation 337 (2018) 494–503

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

An accurate algorithm for evaluating rational functions

Stef Graillat

Sorbonne Université, CNRS, Laboratoire d'informatique de Paris 6, LIP6, F-75005 Paris, France

ARTICLE INFO

MSC:
15-04
65G99
65-04

Keywords:
floating-point
Error-free transformation
Rational function
Horner scheme
Accuracy
Rounding errors

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.

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 [6,7,14], 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 [6,7]. 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 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.

E-mail addresses: stef.graillat@sorbonne-universite.fr, stef.graillat@lip6.fr
URL: http://lip6.fr/Stef.Graillat

https://doi.org/10.1016/j.amc.2018.05.039
0096-3003/

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]

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

This implies that

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

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 $n\mathbf{u} \leq 1$.

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

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]):

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

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] = TwoSum(a, b)
x = a ⊕ b
z = x ⊖ a
y = (a ⊖ (x ⊖ z)) ⊕ (b ⊖ 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] = FastTwoSum(a, b)
x = a ⊕ b
y = (a ⊖ x) ⊕ 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] = Split(a)
factor = 2^s + 1
c = factor ⊗ a
x = c ⊖ (c ⊖ a)
y = a ⊖ 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