Validated Evaluation of Special Mathematical Functions

Franky Backeljauw[a], Stefan Becuwe[a], Annie Cuyt[a], Joris Van Deun[a], Daniel W. Lozier[b]

[a] *Universiteit Antwerpen, Department of Mathematics and Computer Science, Middelheimlaan 1, B-2020 Antwerpen, Belgium*
[b] *National Institute of Standards and Technology 100 Bureau Drive, Stop 8910, Gaithersburg, MD 20899-8910, USA*

Abstract

Because of the importance of special functions, several books and a large collection of papers have been devoted to their use and computation, the most well-known being the Abramowitz and Stegun handbook [1] and its successor [2]. But up to this date, no environment offers routines for the provable correct multiprecision and radix-independent evaluation of these special functions.
We point out how we make good use of series and limit-periodic continued fraction representations in a package that is being developed at the university of Antwerp. Our scalable precision technique is mainly based on the use of sharpened a priori truncation and round-off error upper bounds for real arguments. The implementation is validated in the sense that it returns a sharp interval enclosure for the requested function evaluation, at the same cost as the evaluation.

Keywords: Validated software, special functions.

1. Introduction

Special functions are pervasive in all fields of science. The most well-known application areas are in physics, engineering, chemistry, computer science and statistics. Because of their importance, several books and websites and a large collection of papers have been devoted to these functions. Of the standard work on the subject, the *Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables* edited by Milton Abramowitz and Irene Stegun [1] and published nearly 50 years ago, the American National Institute of Standards and Technology (NIST) claims to have sold over 700 000 copies (over 150 000 directly and more than fourfold that number through commercial publishers)! This old handbook became obsolete in 2010 when NIST released the online DLMF: *NIST Digital Library of Mathematical Functions* edited by Frank W. J. Olver, Daniel W. Lozier, Ronald F. Boisvert and Charles W. Clark [2]. The DLMF updates, completely rewrites, and greatly expands the material contained in

Email addresses: `franky.backeljauw@ua.ac.be` (Franky Backeljauw), `stefan.becuwe@ua.ac.be` (Stefan Becuwe), `annie.cuyt@ua.ac.be` (Annie Cuyt), `daniel.lozier@nist.gov` (Daniel W. Lozier)

the old handbook. Together with its simultaneously published print edition, the DLMF is receiving steadily increasing usage (measured by citations).

Due to their regular appearance in mathematical models of scientific problems, special functions are also pervasive in numerical computations. Consequently, there is no shortage of numerical routines for evaluating many of the special functions in widely used mathematical software packages, systems and libraries such as Maple, Mathematica, MATLAB, IMSL, CERN and NAG. But up to this date, none of these contains *reliable*, or *validated* routines. In this paper, a routine must do more than just compute an accurate approximation. In addition to this, it must provide a *guaranteed bound* on the error of the computed numerical value. In the case of a real-valued function, this error bound determines an interval within which the exact function value is guaranteed to lie.

Constructing algorithms that are accurate and stable, even without guaranteeing bounds on errors, is difficult enough. The following quotes from 1998 and 2002 are still relevant today:

> *"Algorithms with strict bounds on truncation and rounding errors are not generally available for special functions. These obstacles provide an opportunity for creative mathematicians and computer scientists."* — Dan Lozier, general editor of the DLMF project [3];
>
> *"The decisions that go into these algorithm designs — the choice of reduction formulae and interval, the nature and derivation of the approximations — involve skills that few have mastered. The algorithms that MATLAB uses for gamma functions, Bessel functions, error functions, Airy functions, and the like are based on Fortran codes written 20 or 30 years ago."* — Cleve Moler, founder of MATLAB [4].

When we compute, we have come to expect absolute reliability of arithmetic operations, input-output procedures, and certain very elementary functions such as the square root, and we count upon the results of these operations, procedures and functions to lie strictly within certain bounds. Our confidence in these matters is the result of the current IEEE floating-point standard [5].

Many books and papers have been published addressing the need to extend the scope of absolute reliability to more areas of scientific computing. Examination of these sources indicates that attention until now has been directed mainly to processes involving finite sequences of arithmetic operations such as those arising in matrix computations and standard numerical methods for solving polynomials, differential equations, optimization problems, and such. But elementary functions are barely covered, and special functions even less. On the whole, not enough attention is paid to the reliable evaluation of functions, with the exception of [6] and the coverage of multiprecision special and elementary functions in MPFR [7].

As it stands now, the DLMF puts a wealth of mathematical information at the fingertips of scientists who use special functions. No provision is made for a computational component. Abramowitz and Stegun included voluminous numerical tables that are omitted in the DLMF in favor of references to existing mathematical software. We feel the addition of a built-in facility for computing function values on the fly would be found useful, for example by software developers. Therefore, the university of Antwerp authors have begun an active collaboration with NIST to leverage the DLMF as a way to bring the value of reliable computing to the attention of a wider audience. To this end we are developing a validated, multiprecision and radix-independent library of special (and elementary) functions.

Our goal in this paper is to present our approach to the development of general algorithms that are applicable to the reliable arbitrary-precision computation of functions. Convergent and asymptotic series expansions, differential equations, recurrence relations, continued fractions and numerical quadrature are applicable approaches that have been employed successfully in a variety of multiple-precision packages such as [8, 9, 10, 7]. Among these we have chosen to use convergent power series, for which truncation errors are well understood and sharply bounded, and continued fractions, for which error-theoretic improvements in recent years [11, 12] have led to similarly sharp bounds. Together the convergence domains of these representations often cover the full area of interest for users of these functions. In the following sections we describe how a combination of both techniques, with the new results from Sections 3, 6, 7, 8 and 9, leads to validated special function software. The current implementation builds on a complete a priori error analysis, in contrast to the partial implementation described in [13], which makes use of a running error analysis (also see Section 8).

In Table 1 we indicate which functions and argument ranges (on the real line) are covered by our implementation. For the definition of the special functions we refer to [14]. A similar implementation in the complex plane is the subject of future research.

special function	series	continued fraction
$\gamma(a,x)$		$a > 0, x \neq 0$ (1)
$\Gamma(a,x)$		$a \in \mathbb{R}, x \geq 0$
$\mathrm{erf}(x)$	$\lvert x\rvert \leq 1$	identity via $\mathrm{erfc}(x)$
$\mathrm{erfc}(x)$	identity via $\mathrm{erf}(x)$	$\lvert x\rvert > 1$
$\mathrm{dawson}(x)$	$\lvert x\rvert \leq 1$	$\lvert x\rvert > 1$
Fresnel $S(x)$	$x \in \mathbb{R}$ (2)	
Fresnel $C(x)$	$x \in \mathbb{R}$ (2)	
$E_n(x),\ n > 0$		$n \in \mathbb{N}, x > 0$ (3)
${}_2F_1(a,n;c;x)$		$a \in \mathbb{R}, n \in \mathbb{Z}$, $c \in \mathbb{R} \setminus \mathbb{Z}_0^-, x < 1$
${}_1F_1(n;c;x)$		$n \in \mathbb{Z}$, $c \in \mathbb{R} \setminus \mathbb{Z}_0^-, x \in \mathbb{R}$
$I_n(x)$	$n = 0, x \in \mathbb{R}$	$n \in \mathbb{N}, x \in \mathbb{R}$
$J_n(x)$	$n = 0, x \in \mathbb{R}$	$n \in \mathbb{N}, x \in \mathbb{R}$
$I_{n+1/2}(x)$	$n = 0, x \in \mathbb{R}$	$n \in \mathbb{N}, x \in \mathbb{R}$
$J_{n+1/2}(x)$	$n = 0, x \in \mathbb{R}$	$n \in \mathbb{N}, x \in \mathbb{R}$

Table 1: Overview of the special functions that are covered by our implementation.

2. Round-off error accumulation

Let us assume to have at our disposal a scalable precision, IEEE 754-854 compliant [5], floating-point implementation of the basic operations, comparisons, base and type conversions, in the rounding modes upward, downward, truncation and round-to-nearest. Such an implementation is characterized by four parameters: the internal base β, the precision p and the exponent range $[L, U]$. The IEEE 754-854 standard was revised in 2008 [15] but most of this work was carried out in line with the floating-point standard that was valid for the last 20 years. For a concise analysis of the differences we refer to [16].

Here we aim at least at implementations for $\beta = 2$ with precisions $p \geq 53$, and at implementations for use with $\beta = 2^i$ or $\beta = 10^i$ where $i > 1$. Our internal exponent range $[L, U]$ is determined by the largest available integer hardware format (when this turns out to be insufficient, and underflow or overflow results are being generated, then an error message is returned because the subsequent error analysis breaks down).

Rounding a value to the nearest base β precision p floating-point number (with tie break even when $\beta/2$ is odd and odd when $\beta/2$ is even) is indicated by the operation $\bigcirc_p(\cdot)$. Further we denote by $\oplus, \ominus, \otimes, \oslash$ the exactly rounded (to the nearest, with appropriate tiebreak) floating-point implementation of the basic operations $+, -, \times, \div$ in the chosen base β and precision p. Sometimes the generic notation $\circledast$ is used where $* \in \{+, -, \times, \div\}$. Hence, for floating-point numbers x and y, following the IEEE standards, the basic operations are carried out, in the absence of underflow and overflow, with a relative error of at most

$$u(p) := {}^1\!/\!{}_2\, \beta^{-p+1} \tag{1}$$

which is also called half a unit-in-the-last-place in precision p:

$$\left| \frac{(x \circledast y) - (x * y)}{x * y} \right| \leq u(p), \quad * \in \{+, -, \times, \div\},$$

or

$$x \circledast y = (x * y)(1 + \delta), \quad |\delta| \leq u(p), \quad * \in \{+, -, \times, \div\}.$$

Formulated differently, $x \circledast y = \bigcirc_p(x * y)$ where $\bigcirc_p(x * y)$ indicates the nearest base β precision p floating-point neighbour of the exact value $x * y$. The same holds for the square root, the remainder and the conversions between formats, from integers and (since the revision of the IEEE standard in 2008, and also in our implementation) from and to decimal.

In order to compute a relative error bound for a sequence of operations, it is necessary to keep track of all these error terms $1 + \delta$. A basic result, given in [17, p. 63], says that if all $|\delta_i| \leq u(p)$ and $\rho_i = \pm 1$, and if also $nu(p) < 1$, then

$$\prod_{i=1}^{n} (1 + \delta_i)^{\rho_i} = 1 + \theta_n, \tag{2a}$$

$$|\theta_n| \leq \frac{nu(p)}{1 - nu(p)} =: \gamma(n, p). \tag{2b}$$

[1] For $a > 0, a > x$ a faster implementation making use of series is under development.
[2] When $|x| > 1$ the implementation is slow. An improvement thereof is planned.
[3] When $0 < x \leq 1$ the implementation is slow. A faster series version is planned.

In practice, (2) means that n errors δ_i each of at most $u(p)$ in absolute value and combined with one another in products and quotients, cannot simply be stacked together into one larger error of at most $nu(p)$, but are glued together with some thick glue resulting in a slightly larger $[n/(1-nu(p))]\,u(p)$. This result is very convenient, as it allows us to rewrite any number of products and quotients of factors $1+\delta_i$ in an error analysis. Note that the reverse does not hold, meaning that not any expression of the form $1+\theta_n$ with θ_n bounded above in absolute value by $\gamma(n,p)$, can be rewritten as a product of n factors $(1+\delta_i)^{\rho_i}$.

It is also possible to combine factors of the form $1+\theta_i$. For the multiplication and division the following relations hold [17]:

$$(1+\theta_k)(1+\theta_\ell) = 1+\theta_{k+\ell}, \tag{3a}$$

$$\frac{1+\theta_k}{1+\theta_\ell} = \begin{cases} 1+\theta_{k+\ell}, & \text{if } \ell \le k, \\ 1+\theta_{k+2\ell}, & \text{if } \ell > k. \end{cases} \tag{3b}$$

In the division $(1+\theta_k)/(1+\theta_\ell)$, the case $\ell > k$ only applies when nothing is known about the nature of θ_ℓ, meaning that θ_ℓ is not of the form (2b). If $1+\theta_\ell$ comes from products and quotients of factors $1+\delta_i$, which is usual, then the rule $(1+\theta_k)/(1+\theta_\ell) = 1+\theta_{k+\ell}$ applies for all k and ℓ [17, p. 67].

For the addition and subtraction with $x, y \ge 0$ it is easy to verify that:

$$x(1+\theta_k) + y(1+\theta_\ell) = (x+y)\left(1+\theta_{\max(k,\ell)}\right),$$
$$x(1+\theta_k) - y(1+\theta_\ell) = (x-y)\left(1+\theta_{j\max(k,\ell)}\right),$$
$$j \ge \frac{|x|+|y|}{|x-y|}, \quad j \in \mathbb{N}.$$

Note that the order in which one rewrites the error terms influences the sharpness of the final bound: $(x(1+\theta_k) - y(1+\theta_\ell))\,(1+\theta_m)$ can be rewritten as $(x-y)\left(1+\theta_{m+j\max(k,\ell)}\right)$ and as $(x-y)\left(1+\theta_{jm+j\max(k,\ell)}\right)$.

The perturbations θ_n and bounds $\gamma(n,p)$ keep track of the accumulation of round-off errors in numerical algorithms. In the sequel of our analysis, the errors δ_i, which combine into θ_n where the subscript n acts as a tally, relate to the accumulation of floating-point errors incurred in some working precision $\hat{p}$. Working precisions, which are usually somewhat larger than destination precisions in which a computed result is delivered, are topped by a $\hat{\ }$. The accompanying bounds $\gamma(n,\hat{p})$ are then also expressed in terms of this working precision $\hat{p}$.

In the end we want to control this accumulated error and guarantee a threshold for it. We now explain how to distribute such a threshold over individual subexpressions in products, quotients, additions and subtractions.

3. Round-off error control

Let us take a look at the floating-point expression

$$\tilde{Y} = \bigcirc_p\left(\tilde{Y}_1 \circledast \tilde{Y}_2 \circledast \ldots \circledast \tilde{Y}_n\right), \quad * \in \{\times, \div\}$$

consisting of $n-1$ multiplications or divisions, say in precision $\hat{p}$, and an additional rounding of the precision $\hat{p}$ right hand side to the precision $p \le \hat{p}$ memory destination $\tilde{Y}$. Moreover, let $\tilde{Y}_i$ be a floating-point representation of the mathematical quantity Y_i with a relative error

$$\frac{\tilde{Y}_i - Y_i}{Y_i} = \epsilon_i.$$

If we want for $Y = Y_1 * Y_2 * \ldots * Y_n$, that

$$\tilde{Y} = Y(1 + \eta_n), \quad |\eta_n| \le \nu u(p) =: \kappa(\nu, p),$$

in other words, that the computed $\tilde{Y}$ is an approximation for the exact quantity Y within a relative error of at most $\kappa(\nu, p)$ which is usually a small multiple of the half unit-in-the-last-place $u(p)$, then we must find out how the following values relate to one another,

$$\begin{gathered}
\tilde{Z}_i = \tilde{Y}_1 \circledast \ldots \circledast \tilde{Y}_i, \quad i = 2, \ldots, n, \\
\tilde{Z}_i \circledast \tilde{Y}_{i+1} = (\tilde{Z}_i * \tilde{Y}_{i+1})(1 + \delta_i), \quad i = 2, \ldots, n-1, \\
|\delta_i| \le u(\hat{p}), \quad i = 1, \ldots, n-1, \\
\tilde{Y} = \bigcirc_p(\tilde{Z}_n) = \tilde{Z}_n(1 + \delta_n), \\
|\delta_n| \le u(p) = \beta^{\hat{p}-p} u(\hat{p}), \\
\tilde{Y}_i = Y_i(1 + \epsilon_i), \quad i = 1, \ldots, n, \\
\left| \prod_{i=1}^{n} (1 + \epsilon_i)^{\sigma_i} \prod_{i=1}^{n} (1 + \delta_i)^{\rho_i} \right| \le 1 + \kappa(\nu, p), \quad \sigma_i, \rho_i = \pm 1.
\end{gathered}$$

We write

$$\prod_{i=1}^{n-1} (1 + \delta_i)^{\rho_i} = 1 + \theta_{n-1}, \quad |\theta_{n-1}| \le \gamma(n-1, \hat{p})$$

as in Section 2, and we denote

$$1 + \epsilon_0 = (1 + \theta_{n-1})(1 + \delta_n), \quad |\epsilon_0| \le \gamma(n - 1 + \beta^{\hat{p}-p}, \hat{p}). \tag{4}$$

The errors ϵ_i combine into η_n and relate to more general approximation errors than the mere round-off errors δ_i which combine into subscripted θ-values. The η_n are bound by some $\kappa(\nu, p)$ expressed in terms of the precision carried by the memory destination for $\tilde{Y}$ or $\tilde{Y}_i$, which is in its turn denoted by an appropriately subscripted (or superscripted, if necessary) letter p. Here ν need not be integer, but we assume $\nu \ge 2$.

To be more precise, the operation $\tilde{Y}_1 \circledast \tilde{Y}_2 \circledast \ldots \circledast \tilde{Y}_n$ is carried out in precision $\hat{p}$ and then stored as $\tilde{Y}$ in a precision p format, and in order to achieve this each subexpression $\tilde{Y}_i$ is computed in its own working precision $\hat{p}_i$ and subsequently stored in a precision p_i format where obviously $p_i = \hat{p}$ for $i = 1, \ldots, n$. This process can be repeated recursively.

How the threshold $\kappa(\nu, p)$ is to be distributed over the individual $|\epsilon_i|, i = 0, \ldots, n$, is proved in Section 9. Theorem 1 tells us how accurate the operands $\tilde{Y}_i$ must be, in other words how small ϵ_i should be, in order to guarantee

$$\left| \frac{\tilde{Y} - Y}{Y} \right| \le \kappa(\nu, p).$$

We find that if

$$|\epsilon_i| \le \mu_i \frac{\kappa(\nu,p)}{1+\kappa(\nu,p)}, \quad i = 0,\ldots,n, \tag{5a}$$

$$\sum_{i=0}^{n} \mu_i = 1, \tag{5b}$$

then

$$1 + |\eta_n| \le 1 + \kappa(\nu,p).$$

Similar to (2a), formula (5) tells us that the bound $\kappa(\nu,p)$ cannot simply be cut into fractions $\mu_i\kappa(\nu,p)$ but is cut apart with some thick scissors resulting into some slightly smaller pieces $[\mu_i/(1+\kappa(\nu,p))]\,\kappa(\nu,p)$. For $1 \le i \le n$ each of the pieces $[\mu_i/(1+\kappa(\nu,p))]\,\kappa(\nu,p)$ plays the role of a bound $\kappa(\nu_i,p_i)$ on ϵ_i with $\nu_i = \mu_i\nu/(1+\kappa(\nu,p))u(p)/u(\hat{p})$ and expressed as a multiple of $u(p_i) = u(\hat{p})$. It must in its turn be cut into a number of pieces if Y_i is again a composite expression. Here, because of (4),

$$\mu_0 \frac{\kappa(\nu,p)}{1+\kappa(\nu,p)} \ge \gamma\left(n-1+\beta^{\hat{p}-p},\hat{p}\right), \tag{6}$$

and

$$\min_{i=1,\ldots,n} \mu_i \frac{\kappa(\nu,p)}{1+\kappa(\nu,p)} > u(\hat{p}).$$

These inequalities define the range for the working precision $\hat{p}$ for the computation of $\tilde{Y}_1 \circledast \tilde{Y}_2 \circledast \ldots \circledast \tilde{Y}_n$, which is at the same time the destination precision of the individual $\tilde{Y}_i$. The fraction $\mu_0\kappa(\nu,p)/(1+\kappa(\nu,p))$ leaves room for the rounding error to precision p and the $n-1$ multiplications or divisions of the $\tilde{Y}_i$. The bounds $\mu_i\kappa(\nu,p)/(1+\kappa(\nu,p))$ should leave enough room for the rounding error involved in representing the computed $\tilde{Y}_i$ in precision $\hat{p}$, while accommodating at the same time the relative error ϵ_i. Unless n is rather large while β is small, $\hat{p}$ is only slightly larger than p (when β is large and n is small, $\hat{p}$ may exceptionally be slightly less than p). In practice, for given p,β,n and ν, the working precision is obtained by iteratively increasing (exceptionally decreasing) $\hat{p}$ until μ_0 drops (not too far) below an acceptable threshold between 0 and 1 (in our library we have chosen $\mu_0 \le 0.751$). The remaining weights $\mu_i, i = 1,\ldots,n$ are chosen depending on the difficulty with which the operands $\tilde{Y}_i$ in the expression for $\tilde{Y}$ are obtained numerically and add up to $\mu_1 + \ldots + \mu_n = 1 - \mu_0$.

For the addition of n operands $\tilde{Y}_i$ in precision $\hat{p}$ followed by a rounding to the precision p destination $\tilde{Y}$,

$$\tilde{Y} = \bigcirc_p\left(\tilde{Y}_1 \oplus \tilde{Y}_2 \oplus \ldots \oplus \tilde{Y}_n\right),$$

it is easy to see that with $\mu = \mu_0$ from (6) the bounds

$$\max\left(|\epsilon_1|,\ldots,|\epsilon_n|\right) \le (1-\mu)\frac{\kappa(\nu,p)}{1+\kappa(\nu,p)}, \quad 0 < \mu < 1 \tag{7}$$

and

$$|\epsilon_0| \le \mu \frac{\kappa(\nu,p)}{1+\kappa(\nu,p)}$$

deliver the same guarantee. In a sequence of additions and subtractions we gather the positive and negative contributions before carrying out the subtraction. For one precision $\hat{p}$ subtraction, rounded to precision p,

$$\tilde{Y} = \bigcirc_p \left(\tilde{Y}_1 \ominus \tilde{Y}_2 \right),$$

we find that with $\mu = \mu_0$ from (6),

$$\max(|\epsilon_1|, |\epsilon_2|) \frac{|Y_1| + |Y_2|}{|Y_1 - Y_2|} \le (1 - \mu) \frac{\kappa(\nu, p)}{1 + \kappa(\nu, p)}, \quad 0 < \mu < 1$$

should be satisfied.

When $n = 1$, then $\hat{p} = p$ and both (5a) and (7) simplify to $|\epsilon_1| \le \kappa(\nu, p)$.

4. Validated function evaluation

Now we focus on the computation of a single real-valued subexpression Y_i in $Y = Y_1 * Y_2 * \ldots * Y_n$. In our case Y_i is a special mathematical quantity such as $\exp(-x^2)$ or $\sqrt{\pi}$ or $\Gamma(1/x)$. For example

$$Y = \operatorname{erf}(x) = Y_1 \times Y_2, \qquad Y_1 = \frac{2}{\sqrt{\pi}}, \quad Y_2 = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)n!}.$$

In this section we refer to Y_i as $f_i(z_{i,x})$, or briefly $f(z_x)$, where $z_{i,x}$ or z_x is the argument built from an exact argument x (in base β and precision p) passed by a user, and f is the mathematical function which is to be evaluated in $z_{i,x}$ or z_x to yield Y_i.

The realization of a machine implementation $\tilde{Y}_i = \tilde{F}(\tilde{z}_x)$ of a function $Y_i = f(z_x)$ in a floating-point environment is essentially a three-step procedure (we abbreviate the threshold $\kappa(\nu_i, p_i)$ from Section 3 by Δ):

1. From a given (presumed exact) argument x, the actual argument z_x passed to f is computed and an error analysis is made. In the case of the transcendental functions, z_x often results from x by some reduction of x to an argument lying within specified bounds [6]. For the more complicated mathematical functions, z_x results from the use of some simple identities mapping the argument x into a specific half-line or interval.
2. After determining the argument, a mathematical model $F(z_x)$ for $f(z_x)$ is constructed and a truncation error comes into play, which needs to be bounded:
$$\left| \frac{f(z_x) - F(z_x)}{f(z_x)} \right| \le \phi_1 \Delta. \tag{8}$$
In the sequel we systematically denote the model $F \approx f$ by a capital italic letter.
3. When implemented, in other words, when evaluated as $\tilde{F}(\tilde{z}_x)$, this mathematical model $F(z_x)$ is subject to an accumulated round-off error, which also needs to be controlled:
$$\left| \frac{F(z_x) - \tilde{F}(\tilde{z}_x)}{f(z_x)} \right| \le \phi_2 \Delta. \tag{9}$$
We systematically denote the implementation of the model F by $\tilde{F}$.

By bounding both the truncation error (8) and the round-off error (9), we obtain a bound for the total relative error for the computation of $f(z_x)$ using the machine implementation $\tilde{F}(\tilde{z}_x)$, since for $\phi_1 + \phi_2 \leq 1$, by the triangle inequality,

$$\left| \frac{f(z_x) - \tilde{F}(\tilde{z}_x)}{f(z_x)} \right| \leq \Delta. \tag{10}$$

The technique to provide a mathematical model $F(z_x)$ of a function $f(z_x)$ differs substantially when going from a fixed finite precision context to a finite scalable precision context. In the former, the aim is to provide one best mathematical model per fixed precision as in [18]. The model is of minimal complexity with respect to the truncation error bound requested in the fixed finite precision. In the latter, the goal is to provide a generic technique, from which a mathematical model yielding the imposed accuracy, is deduced at runtime. Hence best approximants are not an option since these models have to be recomputed every time the precision is altered and a function evaluation is requested. Despite this the generic technique should generate an approximant of as low complexity as possible.

Our aim is the development of a generic technique, suitable for use in a multiprecision context. However, we want the technique to be efficient enough so that it can compete with the traditional hardware algorithms when used in an environment of 53 binary or (approximately) 16 decimal digits accuracy. So while genericity and accuracy are our primary goals, we also watch over the efficiency of the technique. That is why we use different mathematical models in different subdomains of the function. In addition, we want our implementation to be reliable, in other words, that a sharp interval enclosure for the requested function evaluation is returned without any additional cost (the argument x is sill presumed to be exact)!

Besides series representations, as presented in Section 5, continued fraction representations of functions can be very helpful in the multiprecision context. A lot of well-known constants in mathematics, physics and engineering, as well as elementary and special functions enjoy very nice and rapidly converging continued fraction representations. In addition, many of these fractions are limit-periodic. Both series and continued fraction representations are classical techniques to approximate functions and there's a lot of literature describing implementations that make use of them [19]. But so far, no attempt was made at an efficient yet provable correct implementation.

It is well-known that the tail or remainder term of a convergent Taylor series expansion converges to zero. It is less well-known that the tail of a convergent continued fraction representation does not necessarily converge to zero. It does not even need to converge at all. A suitable approximation of the usually disregarded continued fraction tail may speed up the convergence of the continued fraction approximants. This idea is elaborated in Section 6.

In Section 7 mathematical models $F_i(z_{i,x})$ for $f_i(z_{i,x})$ and their implementations $\tilde{F}_i(\tilde{z}_{i,x})$ are combined to compute an approximation $\tilde{Y}$ for Y within the relative error $\kappa(\nu, p)$. Hence the error analysis of Section 3 applies before (8) and (9).

From the bound (10) the guaranteed enclosure

$$\left[\frac{\tilde{F}(\tilde{z}_x)}{1+\Delta}, \frac{\tilde{F}(\tilde{z}_x)}{1-\Delta} \right], \quad f(z_x) > 0$$

or

$$\left[\frac{\tilde{F}(\tilde{z}_x)}{1-\Delta}, \frac{\tilde{F}(\tilde{z}_x)}{1+\Delta}\right], \quad f(z_x) < 0$$

is obtained. Precision p interval endpoints are obtained by performing the additions $1 \pm \Delta$ and the divisions $\tilde{F}(\tilde{z}_x)/(1 \pm \Delta)$ in appropriately rounded precision p arithmetic.

In the Sections 5 and 6 we deal with the case $n = 1$ for Y and simplify the notation of the calling argument z_x to z. It remains that the argument x passed by a user is considered to be exact, while the calling argument z in $F(z) \approx f(z)$ may not be exactly representable anymore.

5. Taylor series development

For simplicity, but without loss of generality, we assume that the Taylor series of $f(z)$ is given at the origin:

$$f(z) = \sum_{i=0}^{\infty} a_i z^i. \tag{11}$$

If we want to bound the total relative error using (10), we must determine N such that for $F(z) = T_N(z)$, the partial sum of degree N of (11), the truncation error is bounded by (8), and evaluate $T_N(z)$, in a working precision $\hat{p}$ possibly slightly larger than the destination precision p, such that for the computed value $\tilde{F}(\tilde{z}) = \tilde{T}_N(\tilde{z})$, the round-off error satisfies (9).

An upper bound for the absolute truncation error $|f(z) - T_N(z)|$ is obtained from the sequence of terms $\{a_i z^i\}_i$ of the series (11). If the quotients $a_i z/a_{i-1}$ of consecutive terms are positive and strictly decreasing, and if there exists an i_0 such that

$$\frac{a_i z}{a_{i-1}} \le R < 1, \quad i \ge i_0,$$

then, for $N \ge i_0$,

$$\sum_{i=N+1}^{\infty} a_i z^i \le a_{N+1} z^{N+1} \sum_{i=0}^{\infty} R^i = \frac{a_{N+1} z^{N+1}}{1-R}.$$

On the other hand, if the terms in the series are alternating with $-a_i z/a_{i-1}$ positive and decreasing, then for any odd N,

$$\sum_{i=N+1}^{\infty} a_i z^i \le a_{N+1} z^{N+1}.$$

If furthermore $|q(z)|$ is a (tight) lower bound for $|f(z)|$, then we can replace the bound for the truncation error in (8) by

$$\left|\frac{f(z) - T_N(z)}{q(z)}\right| \le \phi_1 \Delta, \tag{12}$$

from which we can determine the degree N.

Since in both cases $|f(z)| \ge |T_N(z)|$, we can also replace the round-off error in (9) by

$$\left|\frac{T_N(z) - \tilde{T}_N(\tilde{z})}{f(z)}\right| \le \left|\frac{T_N(z) - \tilde{T}_N(\tilde{z})}{T_N(z)}\right| \le \phi_2 \Delta. \tag{13}$$

A standard method for the evaluation of the partial sum $T_N(z)$ is Horner's scheme, which consists of the rearrangement

$$T_N(z) = a_0 + z(a_1 + z(a_2 + z(\ldots + za_N))). \tag{14}$$

Since, for series of elementary and special functions, the coefficients a_i are often related by a simple ratio $r_i \coloneqq a_i/a_{i-1}$, Horner's scheme can be rewritten as

$$T_N(z) = a_0(1 + zr_1(1 + zr_2(1 + zr_3(\ldots + zr_N)))), \tag{15}$$

leading to the following recursive computation scheme:

$$\begin{aligned} T_{N,N}(z) &= 1, \\ T_{N,i}(z) &= 1 + zr_{i+1}T_{N,i+1}(z), \quad i = N-1, \ldots, 0, \\ T_N(z) &= a_0 T_{N,0}(z). \end{aligned}$$

We now derive a round-off error bound for the computation of the scheme in the working precision $\hat{p}$.

Let $\tilde{z}$, $\tilde{a}_0$ and $\tilde{r}_i$ denote the machine representations of z, a_0 and r_i respectively, computed at the working precision $\hat{p}$, such that

$$\begin{aligned} \tilde{z} &= z\left(1 + \theta_{n(z)}\right), \\ \tilde{a}_0 &= a_0\left(1 + \theta_{n(a_0)}\right), \\ \tilde{r}_i &= r_i\left(1 + \theta_{n(r_i)}\right), \quad i = 1, \ldots, N. \end{aligned}$$

and let the accumulated relative errors $\theta_{n(z)}$, $\theta_{n(a_0)}$ and $\theta_{n(r_i)}$ be bounded according to (2b), expressed in function of $u(\hat{p})$. Inserting these data error terms and adding round-off error terms for the basic operations into the nested scheme given above, results in

$$\begin{aligned} \tilde{T}_{N,N}(\tilde{z}) &= 1, \\ \tilde{T}_{N,i}(\tilde{z}) &= 1 \oplus \tilde{z} \otimes \tilde{r}_{i+1} \otimes \tilde{T}_{N,i+1}(\tilde{z}), \quad i = N-1, \ldots, 0, \\ &= (1 + \delta_3) + z\left(1 + \theta_{n(z)}\right) r_{i+1}\left(1 + \theta_{n(r_{i+1})}\right) \\ &\quad \tilde{T}_{N,i+1}(\tilde{z})(1 + \delta_1)(1 + \delta_2)(1 + \delta_3), \\ \tilde{T}_N(\tilde{z}) &= \tilde{a}_0 \otimes \tilde{T}_{N,0}(\tilde{z}) \\ &= a_0\tilde{T}_{N,0}(\tilde{z})\left(1 + \theta_{n(a_0)}\right)(1 + \delta_4). \end{aligned}$$

Collecting all round-off error terms gives

$$\begin{aligned} \tilde{T}_N(\tilde{z}) = \sum_{i=0}^{N} \Bigg(&(1 + \theta_{n(a_0)})(1 + \delta_3)(1 + \delta_4) \times \\ &\prod_{\ell=0}^{i-1} (1 + \theta_{n(z)})(1 + \theta_{n(r_{\ell+1})})(1 + \delta_1)(1 + \delta_2)(1 + \delta_3) \Bigg) a_i z^i, \end{aligned}$$

which can be rearranged, using (2b) and (3a), into the following expression for the final value:

$$\tilde{T}_N(\tilde{z}) = \sum_{i=0}^{N} \left(1 + \theta_{k(i)}\right) a_i z^i,$$

with

$$\begin{aligned} k(i) &= 2 + n(a_0) + i\,(3 + n(z)) + \sum_{j=0}^{i-1} n(r_{j+1}) \\ &\le 2 + n(a_0) + i\left(3 + n(z) + \max_{j=1}^{i} n(r_j)\right). \end{aligned}$$

Further rewriting gives

$$\begin{aligned} \left|T_N(z) - \tilde{T}_N(\tilde{z})\right| &= \left|\sum_{i=0}^{N} \theta_{k(i)} a_i z^i\right| \\ &\le \max_{i=0}^{N} \left|\theta_{k(i)}\right| \sum_{i=0}^{N} \left|a_i z^i\right| \\ &\le \gamma(k(N), \hat{p}) T_N^+(|z|), \end{aligned}$$

where

$$T_N^+(z) = \sum_{i=0}^{N} |a_i|\, z^i.$$

Hence, the round-off error is bounded by

$$\left|\frac{T_N(z) - \tilde{T}_N(\tilde{z})}{T_N(z)}\right| \le \gamma(n(T_N), \hat{p}) \frac{T_N^+(|z|)}{|T_N(z)|}, \tag{16}$$

where $\gamma(n(T_N), \hat{p})$ is expressed in function of $u(\hat{p})$ as in (2b) with $n(T_N) \coloneqq k(N)$.

Note that the factor

$$\frac{T_N^+(|z|)}{|T_N(z)|} \ge 1. \tag{17}$$

It equals 1 if $a_i \ge 0$ for all i and $z \ge 0$, or if $(-1)^i a_i \ge 0$ for all i and $z \le 0$. Otherwise, this factor can be arbitrarily large. As such, it might be necessary to limit the domain for z in order to obtain a reasonable upper bound for this factor.

Bounding (16) by $\phi_2\Delta$, as required in (13), gives a condition on the working precision $\hat{p}$,

$$\frac{n(T_N)\, u(\hat{p})}{1 - n(T_N)\, u(\hat{p})} \frac{T_N^+(|z|)}{|T_N(z)|} \le \phi_2\Delta.$$

This can be rewritten as

$$u(\hat{p}) \le \frac{\phi_2\Delta}{n(T_N)\left(\frac{T_N^+(|z|)}{|T_N(z)|} + \phi_2\Delta\right)},$$

which results in the following bound for the working precision,

$$\begin{aligned} \hat{p} \ge 1 &- \log_\beta(2) - \log_\beta(\phi_2\Delta) + \log_\beta(n(T_N)) \\ &+ \log_\beta\left(\frac{T_N^+(|z|)}{|T_N(z)|}\right) + \log_\beta(1 + \phi_2\Delta). \end{aligned}$$

Here, the last two terms come from the fact that

$$\frac{T_N^+(|z|)}{|T_N(z)|} + \phi_2\Delta \le \frac{T_N^+(|z|)}{|T_N(z)|}\left(1 + \phi_2\Delta\right).$$

6. Continued fraction representation

Let us consider a continued fraction representation of the form

$$f(z) = \cfrac{a_1}{1 + \cfrac{a_2}{1 + \dots}} = \frac{a_1 \rfloor}{\lceil 1} + \frac{a_2 \rfloor}{\lceil 1} + \dots = \sum_{i=1}^{\infty} \frac{a_i \rfloor}{\lceil 1}, \tag{18}$$

where $a_i := a_i(z)$ and $a_i \geq -1/4$. Here a_i is called the i-th partial numerator. For the restriction $a_i \geq -1/4$ we refer to [14, p. 55]. The case where $a_i < -1/4$ can also be dealt with but is much trickier [20, p. 159], [21].

The continued fraction is said to be limit-periodic if the limit $\lim_{i\to\infty} a_i$ exists (it is allowed to be $+\infty$). We respectively denote by the N-th approximant $f_N(z; w_N)$ and N-th tail $t_N(z)$ of (18), the values

$$f_N(z; w_N) = \sum_{i=1}^{N-1} \frac{a_i \rfloor}{\lceil 1} + \frac{a_N \rfloor}{\lceil 1 + w_N},$$
$$t_N(z) = \sum_{i=N+1}^{\infty} \frac{a_i \rfloor}{\lceil 1}.$$

We restrict ourselves to the case where a sequence $\{w_i\}_i, w_i \neq 0$ can be chosen such that $\lim_{i\to\infty} f_i(z; w_i) = \lim_{i\to\infty} f_i(z; 0)$.

The tails $t_N(z)$ of a convergent continued fraction can behave quite differently compared to the tails of a convergent series, which always go to zero. We illustrate the different cases with an example. Take for instance the continued fraction expansion

$$\frac{\sqrt{1+4z}-1}{2} = \sum_{i=1}^{\infty} \frac{z \rfloor}{\lceil 1}, \quad z \geq -\frac{1}{4}.$$

Each tail $t_N(z)$ converges to the value $(\sqrt{1+4z}-1)/2$ as well and hence the sequence of tails is a constant sequence. More remarkable is that the even-numbered tails of the convergent continued fraction

$$\sqrt{2}-1 = \sum_{i=1}^{\infty} \left(\frac{(3+(-1)^i)/2 \rfloor}{\lceil 1} \right) = \frac{1 \rfloor}{\lceil 1} + \frac{2 \rfloor}{\lceil 1} + \frac{1 \rfloor}{\lceil 1} + \frac{2 \rfloor}{\lceil 1} + \dots$$

converge to $\sqrt{2}-1$ while the odd-numbered tails converge to $\sqrt{2}$ (hence the sequence of tails does not converge), and that the sequence of tails $\{t_N(z)\}_N = \{N+1\}_N$ of

$$1 = \sum_{i=1}^{\infty} \frac{i(i+2) \rfloor}{\lceil 1}$$

converges to $+\infty$. When carefully monitoring the behaviour of these continued fraction tails, very accurate approximants $f_N(z; w_N)$ for $f(z)$ can be computed by making an appropriate choice for w_N [12]. In our implementation we take w_N to be an exactly representable number in the floating-point set under consideration. The relative truncation error $|f(z) - f_N(z; w_N)|/|f(z)|$ is bounded by the so-called interval sequence theorem [11].

The following example illustrates that it makes quite a difference to use no tail ($w_N = 0$), or the mathematical limit value [14] ($w_N = \lim_{N\to\infty} t_N(z)$), or an accurate estimate of the N-th tail ($w_N \approx t_N(z)$). Consider the continued fraction representation (22). Evaluating the 13-th approximant at $z = 6.5$ using $w_{13} = 0$ and exact arithmetic, delivers an approximant with 28 significant decimal digits (relative error bounded above by 5×10^{-27}). Plugging in the limit of the tail sequence for w_N, results in a loss of two significant digits rather than a gain! But making use of a double precision estimate of the actual tail value $t_{13}(z)$ results in almost 40 significant decimal digits (relative error bounded above by 8×10^{-41}).

Another way to view the influence of an appropriate choice for the tail is the following. At $z = 6.5$ the 24-th approximant of (22) with $w_N = 0$ guarantees a relative error bounded above by $2u(p), p = 40, \beta = 10$. But N can easily be reduced from 24 to 19 by using $w_{19} = -9.2861$, and even further to 14 by plugging in $w_{14} = -5.5909501809$. These experiments can be verified by the reader at `cfsf.ua.ac.be`.

Let the partial numerators $a_i(z)$ be represented by the base β precision $\hat{p}$ floating-point approximation $\tilde{a}_i := \tilde{a}_i(\tilde{z})$ satisfying

$$\left|\frac{\tilde{a}_i - a_i}{a_i}\right| = |\theta_{n(a_i)}| \le \gamma(n(a_i), \hat{p}), \quad i = 1, 2, \ldots.$$

We denote for $a_i > 0$

$$b_i := \frac{\tilde{a}_i}{(1 + \gamma(n(a_i), \hat{p}))},$$
$$c_i := \frac{\tilde{a}_i}{(1 - \gamma(n(a_i), \hat{p}))},$$

and for $a_i < 0$

$$b_i := \frac{\tilde{a}_i}{(1 - \gamma(n(a_i), \hat{p}))},$$
$$c_i := \frac{\tilde{a}_i}{(1 + \gamma(n(a_i), \hat{p}))},$$

such that

$$b_i \le a_i \le c_i$$

and assume

$$-1/4 \le b_i, \quad 0 \le b_i c_i.$$

The continued fraction (18) is most stably evaluated using the backward algorithm:

$$\begin{aligned}
\tilde{F}_{N+1}^{(N)} &= w_N, \\
\tilde{F}_i^{(N)} &= \tilde{a}_i \oslash (1 \oplus \tilde{F}_{i+1}^{(N)}), \quad i = N, \ldots, 1, \\
\tilde{F}(z) &= \tilde{f}_N(\tilde{z}; w_N) = \tilde{F}_1^{(N)}.
\end{aligned}$$

Let $[D_N, U_N]$ denote a reliable enclosure of the N-th tail $t_N(z)$. For the special mathematical functions these are easy to obtain [11]. Then a reliable enclosure of $f_N(z; w_N)$ with $w_N \in [D_N, U_N]$ is obtained from the computation of

$$[D_i, U_i] = \frac{[b_{i+1}, c_{i+1}]}{1 + [D_{i+1}, U_{i+1}]}, \quad i = N - 1, \ldots, 0. \qquad (19)$$

Here, for $0 \le b_{i+1} \le c_{i+1}$

$$\begin{aligned} D_i &= \frac{b_{i+1}}{1+U_{i+1}}, \\ U_i &= \frac{c_{i+1}}{1+D_{i+1}}, \end{aligned}$$

and for $b_{i+1} \le c_{i+1} \le 0$

$$\begin{aligned} D_i &= \frac{b_{i+1}}{1+D_{i+1}}, \\ U_i &= \frac{c_{i+1}}{1+U_{i+1}}. \end{aligned}$$

With $-1/4 \le b_i \le a_i \le c_i$, we have $-1/2 \le D_i \le t_i(z) \le U_i$. Then for the mathematical approximation $F(z) = f_N(z; w_N)$, the truncation error satisfies

$$\begin{aligned} \left| \frac{f(z) - f_N(z; w_N)}{f(z)} \right| &\le \frac{U_N - D_N}{1 + D_N} \prod_{i=1}^{N-1} M_i, \\ M_i &= \max \left\{ \left| \frac{D_i}{1+D_i} \right|, \left| \frac{U_i}{1+U_i} \right| \right\}, \quad D_N \le w_N \le U_N. \end{aligned}$$

From the condition

$$\frac{U_N - D_N}{1 + D_N} \prod_{i=1}^{N-1} M_i \le \phi_1 \Delta,$$

N is deduced, following the ideas in [11]. Depending on the sign of the partial numerators a_i and their monotonicity behaviour (ascending, descending, oscillatory), even/odd approximants of $t_i(z), i = 1, \dots, N$, provide interval representations for each of the values D_i and U_i. We found it most useful to aim for 12-13 decimal significant digits in $w_N \in]D_N, U_N[$.

A proper working precision $\hat{p} \ge p$ is obtained from the upper bound for the round-off error in the backward algorithm for $f_N(z; w_N)$. It is proved in [22] that for

$$\begin{aligned} &|\theta_{n(a_i)}| \le \gamma(n(a_i), \hat{p}), \\ &\Gamma_N = \max_{i=1,\dots,N} \frac{n(a_i)}{1 - n(a_i) u(\hat{p})}, \\ &M = \max_{i=1,\dots,N} M_i, \end{aligned}$$

the round-off error in the backward algorithm, when carried out in base β precision $\hat{p}$ floating-point arithmetic, satisfies

$$\left| \frac{f_N(z; w_N) - \tilde{f}_N(\tilde{z}; w_N)}{f_N(z; w_N)} \right| \le (4 + \Gamma_N) \frac{1 - M^N}{1 - M} u(\hat{p}).$$

From the upper bound

$$(4 + \Gamma_N) \frac{1 - M^N}{1 - M} u(\hat{p}) \le \phi_2 \Delta,$$

we can solve for the precision $\hat{p}$ as follows. Let $\check{p}$ denote the approximate precision defined by

$$u(\check{p}) = \frac{\phi_2 \Delta}{\dfrac{1-M^N}{1-M}\left(4 + \max_{i=1}^{N} n(a_i)\right)}.$$

Then with

$$\begin{aligned} a &= \max_{i=1}^{N} n(a_i), \\ b &= \frac{1-M^N}{1-M}, \\ c &= \phi_2 \Delta, \end{aligned}$$

the function $f(a,b,c)$ in Lemma 3 equals $f(a,b,c) = u(\hat{p})/u(\check{p})$. It follows from this lemma that $\check{p} \le \hat{p} < \check{p}+1$ (because $\beta \ge 2$). Since the exact solution $\hat{p}$ is difficult to compute for small Δ, we may safely bound the working precision by $\check{p}+1$, giving

$$\begin{aligned} \check{p} &\ge 2 - \log_\beta(2) - \log_\beta(\phi_2\Delta) \\ &\quad + \log_\beta\left(\frac{1-M^N}{1-M}\right) + \log_\beta\left(4 + \max_{i=1}^{N} n(a_i)\right). \end{aligned}$$

7. Example: the error and complementary error functions

We consider the error function and the complementary error function

$$\begin{aligned} \operatorname{erf}(x) &= \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\,dt, \\ \operatorname{erfc}(x) &= \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2}\,dt \end{aligned}$$

for $x \in \mathbb{R}$. These functions are closely related to one another through

$$\operatorname{erfc}(x) = 1 - \operatorname{erf}(x). \tag{20}$$

Furthermore, we can limit the discussion to $x > 0$ since

$$\begin{aligned} \operatorname{erf}(0) &= 0, \\ \operatorname{erf}(-x) &= -\operatorname{erf}(x), \\ \operatorname{erfc}(-x) &= 2 - \operatorname{erfc}(x). \end{aligned}$$

In addition we implement the special cases

$$\begin{aligned} \operatorname{erf}(-\infty) &= -1, \\ \operatorname{erf}(+\infty) &= 1, \\ \operatorname{erf}(\texttt{not-a-number}) &= \texttt{not-a-number}. \end{aligned}$$

For the implementation of $\operatorname{erf}(x)$ and $\operatorname{erfc}(x)$ we have chosen the representations (series and continued fractions) with the highest rate of convergence on each of the two regions $[0,1]$ and $(1,+\infty)$. For a comparison of the different series and continued fraction representations of several special functions, we refer to the many numerical tables in [14, Chapter 13].

7.1. Series implementation of erf(x) for $0 < x \le 1$

The Maclaurin series of $\operatorname{erf}(x)$ is defined by

$$\operatorname{erf}(x) = f_1(x) \times f_2(x) = \frac{2}{\sqrt{\pi}} \times \sum_{i=0}^{\infty} \frac{(-1)^i x^{2i+1}}{(2i+1)i!}.$$

Suppose that for the corresponding floating-point expression

$$\widetilde{\operatorname{erf}}(x) = \bigcirc_p(\tilde{F}_1(\tilde{z}_{1,x}) \otimes \tilde{F}_2(\tilde{z}_{2,x})),$$

we require

$$\left|\frac{\operatorname{erf}(x) - \widetilde{\operatorname{erf}}(x)}{\operatorname{erf}(x)}\right| \le \kappa(\nu, p) = 2u(p), \quad \nu = 2, \quad n = 2.$$

Using (6) with

$$\mu_0 = \frac{1 + \nu u(p)}{\nu} \; \frac{1 + \dfrac{n-1}{\beta^{\hat{p}-p}}}{1 - \left(1 + \dfrac{n-1}{\beta^{\hat{p}-p}}\right) u(p)}, \tag{21}$$

we can compute the precision $\hat{p}$, in which both $\tilde{F}_1(\tilde{z}_{1,x})$ and $\tilde{F}_2(\tilde{z}_{2,x})$ need to be delivered and in which the floating-point product $\tilde{F}_1(\tilde{z}_{1,x}) \otimes \tilde{F}_2(\tilde{z}_{2,x})$ needs to be computed. For $\beta = 10$ this results in $\hat{p} = p + 1$ with $\mu_0 \le 0.55$.

To comply with (5b), we take $\mu_1 = \mu_2 = \frac{1}{2}(1 - \mu_0)$. The values $\tilde{F}_1(\tilde{z}_{1,x})$ and $\tilde{F}_2(\tilde{z}_{2,x})$ are themselves computed in respective working precisions $\hat{p}_1$ and $\hat{p}_2$ which are determined separately using the same principles. The bounds for the relative truncation errors ϵ_1 and ϵ_2 associated with $\tilde{F}_1(\tilde{z}_{1,x})$ and $\tilde{F}_2(\tilde{z}_{2,x})$ are given by (5a). In the remainder of this section, we focus on the evaluation of $f_2(x)$.

Before determining the degree N of the partial sum $T_N(x)$ of the series $f_2(x)$ using (12), we note that a sufficient lower bound for $f_2(x)$ is given by

$$q(x) = x - \frac{x^3}{3},$$

for which $f_2(x)/q(x) \le 1.121$, for $0 < x \le 1$. Since the series is alternating, we take the minimal odd degree N for which

$$\frac{x^{2N+3}}{(2N+3)(N+1)!} \frac{1}{q(x)} \le \frac{1}{2}\kappa(\nu_2, p).$$

Note that the value of N depends on x and increases for growing values of x.

We can compute the partial sum $F_2(x) = T_N(x)$ using (15) by replacing x by $z = x^2$ and a_0 by x. Since the input x is assumed to be exactly representable, we get

$$n(z) = 1,$$
$$n(a_0) = 0.$$

Otherwise, the tally $n(z)$ is increased accordingly. Furthermore, note that the coefficients of the series given by $f_2(x)$ are related by the ratio

$$r_n = -\frac{2n-1}{n(2n+1)}.$$

x	p	n	$\hat{p}$	x	p	N	$\hat{p}$
0.125	50	19	56	0.625	50	33	56
	100	34	106		100	57	106
	250	77	256		250	121	257
0.250	50	23	56	0.750	50	36	56
	100	42	106		100	62	106
	250	91	256		250	129	257
0.375	50	27	56	0.875	50	39	56
	100	47	106		100	66	106
	250	102	257		250	137	257
0.500	50	30	56	1.000	50	41	56
	100	52	106		100	70	106
	250	112	257		250	144	257

Table 2: For $\beta = 10$, given p and x, the N-th partial sum of $\underset{\sim}{f_2}(x)$ evaluated in precision $\hat{p}$, guarantees a total relative error of at most $2u(p)$ for $\left|\mathrm{erf}(x) - \widetilde{\mathrm{erf}}(x)\right| / |\mathrm{erf}(x)|$.

If we assume that N is such that $N(2N+1)$ remains exactly representable (which is, for instance, the case for $N \leq 67108863$ when computing $N(2N+1)$ in IEEE double precision floating-point arithmetic), then computing this ratio involves only one floating-point division, so we have

$$\max_{n=1}^{N} n(r_n) = 1.$$

Taken together, the value of $n(T_N)$ is given by

$$n(T_N) = 1 + 5N.$$

Finally, we note that for $0 < x \leq 1$, the factor (17) is bounded by 2. We choose $\phi_1 = 1/2 = \phi_2$ to obtain the respective values for N and $\hat{p}$ listed in *Table 2* for different x and p and $\beta = 10$.

7.2. Continued fraction implementation on $1 < x$

A rapidly converging continued fraction representation for $\mathrm{erfc}(x)$ is given by

$$\mathrm{erfc}(x) = \frac{f_1(x)}{f_2(x)} \times f_3(x) = \frac{e^{-x^2}}{\sqrt{\pi}} \times \tag{22}$$

$$\left(\frac{2x/(2x^2+1)\,\big|}{\big|\,1} + \sum_{i=2}^{\infty} \frac{\frac{-(2i-3)(2i-2)}{(2x^2+4i-7)(2x^2+4i-3)}\,\big|}{\big|\,1} \right). \tag{23}$$

We expect the floating-point implementation

$$\widetilde{\mathrm{erfc}}(x) = \bigcirc_p(\tilde{F}_1(\tilde{z}_{1,x}) \oslash \tilde{F}_2(\tilde{z}_{2,x}) \otimes \tilde{F}_3(\tilde{z}_{3,x})),$$

to satisfy

$$\left| \frac{\mathrm{erfc}(x) - \widetilde{\mathrm{erfc}}(x)}{\mathrm{erfc}(x)} \right| \leq \kappa(\nu, p) = 2u(p), \quad \nu = 2, \quad n = 3.$$

Since n is larger, the fraction μ_0 given by (21) is slightly larger. Furthermore we take $\mu_1 = \mu_2 = \frac{1}{4}(1 - \mu_0)$ and $\mu_3 = \frac{1}{2}(1 - \mu_0)$. Again we find that for $\beta = 10$ the floating-point division and product $\tilde{F}_1(\tilde{z}_{1,x}) \oslash \tilde{F}_2(\tilde{z}_{2,x}) \otimes \tilde{F}_3(\tilde{z}_{3,x})$ need to be computed in precision $\hat{p} = p + 1$, now guaranteeing $\mu_0 \le 0.6$. So the individual $\tilde{F}_i(\tilde{z}_{i,x}), i = 1, 2, 3$ are delivered in memory destinations of size $p_i = p+1, i = 1, 2, 3$. They are in their turn computed in working precisions $\hat{p}_i, i = 1, 2, 3$ which are determined in the same way. The bounds for the relative truncation errors ϵ_1, ϵ_2 and ϵ_3 associated with $\tilde{F}_1(\tilde{z}_{1,x})$, $\tilde{F}_2(\tilde{z}_{2,x})$ and $\tilde{F}_3(\tilde{z}_{3,x})$ are obtained from (5a) as above. In the remainder of this section we focus on $f_3(x)$.

When computing

$$\begin{aligned} a_1(x) &= \frac{2x}{2x^2 + 1}, \\ a_i(x) &= \frac{-(2i-3)(2i-2)}{(2x^2 + 4i - 7)(2x^2 + 4i - 3)}, \quad i \ge 2, \end{aligned}$$

in floating-point arithmetic, we find

$$\begin{aligned} n(a_1) &= 5, \\ n(a_i) &= 9, \quad i \ge 2. \end{aligned}$$

Here we assume that $2N-3$, $2N-2$, $4N-7$ and $4N-3$ are exactly representable in IEEE double precision arithmetic (which is satisfied for $N \le 2.25 \times 10^{15}$). If this is not the case, the tally $n(a_i)$ can be adapted accordingly. Also x is again assumed to be an exactly representable number in the destination precision p (and the higher working precisions). For the sake of being radix-independent, the multiplication $2x$ is not assumed to be an exact operation.

Double precision interval enclosures $[b_i, c_i]$ for the partial numerators a_i and the inner bounds

$$\begin{aligned} D_i \le \sum_{j=i+1}^{i+\ell-1} \frac{b_j \rfloor}{\lceil 1} + \frac{b_{i+\ell} \rfloor}{\lceil 1 + (-1 + \sqrt{4b_{i+\ell+1} + 1})/2} &\le t_i(x), \\ t_i(x) \le \sum_{j=i+1}^{i+\ell-1} \frac{c_j \rfloor}{\lceil 1} + \frac{c_{i+\ell} \rfloor}{\lceil 1 - 1/2} &\le U_i, \end{aligned}$$

and outer bounds

$$\begin{aligned} \sum_{j=i+1}^{i+\ell-1} \frac{b_j \rfloor}{\lceil 1} + \frac{b_{i+\ell} \rfloor}{\lceil 1 - 1/2} &\le D_i, \\ U_i \le \sum_{j=i+1}^{i+\ell-1} \frac{c_j \rfloor}{\lceil 1} + \frac{c_{i+\ell} \rfloor}{\lceil 1 + (-1 + \sqrt{4c_{i+\ell+1} + 1})/2}&, \end{aligned}$$

lead to upper bounds for $M_1, \ldots, M_{N-1}$ and $U_N - D_N$ and a lower bound for D_N [11, 13]. Upper bounds for $M_1, \ldots, M_{N-1}$ and D_N can be obtained with $\ell = 1$ while larger values for ℓ are recommended for an accurate bound on $U_N - D_N$.

We choose $\phi_1 = 1/2 = \phi_2$ to find the values for N and $\check{p}$ given in *Table 3* for a variety of x- and p-values and $\beta = 10$.

x	p	N	$\check{p}$	w_N
1.750	50	165	54	$-4.329496127779110e-01$
	100	871	104	$-4.705213176840440e-01$
	250	6242	255	$-4.889463306700787e-01$
2.500	50	84	54	$-3.681442608479621e-01$
	100	438	104	$-4.408225669101029e-01$
	250	3088	255	$-4.775605158021344e-01$
3.250	50	55	54	$-2.961145010800355e-01$
	100	271	104	$-4.030228242779376e-01$
	250	1851	254	$-4.623699926667059e-01$
4.000	50	40	54	$-2.240298230456090e-01$
	100	182	104	$-3.568083173196760e-01$
	250	1240	254	$-4.435599088170887e-01$
4.750	50	28	54	$-1.491872106512259e-01$
	100	138	104	$-3.096699530828058e-01$
	250	900	254	$-4.216730831698096e-01$
5.500	50	24	54	$-1.046709780247154e-01$
	100	102	104	$-2.545017655429691e-01$
	250	685	254	$-3.967538546425260e-01$
6.250	50	18	54	$-6.056715517294565e-02$
	100	87	104	$-2.115853181648406e-01$
	250	549	254	$-3.702016868974058e-01$
7.000	50	13	54	$-3.029929019566408e-02$
	100	70	104	$-1.640969910050955e-01$
	250	450	254	$-3.415466979650395e-01$

Table 3: For $\beta = 10$, given p and x, the N-th approximant of $f_3(x)$ evaluated in precision $\check{p}$ with double precision tail estimate w_N, guarantees a total relative error of at most $2u(p)$ for $|\mathrm{erfc}(x) - \widetilde{\mathrm{erfc}}(x)|/|\mathrm{erfc}(x)|$.

7.3. *Further implementation details*

A transcript of the documented code that implements the Sections 7.1 and 7.2, completely parallelling the description in this paper, can be downloaded from the webpages `cant.ua.ac.be/publications` or `cfsf.ua.ac.be`. The variable names in the code equal the mathematical notations used in the paper.

8. Tips and tricks of the trade

8.1. *Multiprecision arithmetic*

In the above analysis we assume that the scalable precision base β arithmetic follows the IEEE 754-854 standard. The main reason is that our error analysis is based on this model. But typically, multiprecision software implementations, as opposed to fixed precision hardware implementations, support some additional features. Some of these may even bring the accumulated error down. We point out which features in our own library `MpIeee` qualify as such:

- Where the 1985 standard requires the base conversions (between decimal and the internal base β) to be exactly rounded only for a limited range of real decimal numbers, in the `MpIeee` library the conversions are exactly rounded for all real decimal numbers. Thanks to the use of C++, all input, output and arithmetic operations are simply expressed by overriding intrinsic operators.

- All operations accept multiprecision numbers with different precisions for the (one or two) operands and the result. As such, `MpIeee` allows for mixed-precision arithmetic. Making use of the technique of delayed evaluation when (non-compound) operations or function evaluations are assigned to a destination, an expression of the form $Y = \bigcirc_p(Y_1 \circledast Y_2)$ where the precisions p, p_1 and p_2 may differ, is carried out with one single rounding error of at most $u(p)$.

- In a multiprecision implementation the bound $\kappa(\nu, p)$ is best passed as $\log_\beta(\kappa(\nu, p))$, rounded down. Otherwise (double precision) underflow may result when β and p are large and the bound is stored in double precision (it is not very sensible to not use a hardware type for it).

8.2. *Running error analysis*

Instead of a straightforward implementation of the continued fraction technique above, a faster implementation as described in [13] can be used. The latter makes use of a running error analysis rather than a fully a priori error analysis. Both are built on the same mathematical continued fraction toolbox though.

Besides double precision and multiprecision interval enclosures for the partial numerators a_i, it also needs a crude but guaranteed interval enclosure for the continued fraction tail $t_i(z)$, if necessary only from a certain i on. The only requirement is that this enclosure is not so crude that it unnecessarily stretches beyond -1 because then the arithmetic breaks down.

To avoid the validated computation of the intervals $[D_i, U_i]$ for the truncation error bound, and $]D_i, U_i[$ for the choice of w_i, the running error technique introduces, besides the true relative error $r := |y - \tilde{y}|/|y|$ for a magnitude y approximated by a computed value $\tilde{y}$, a computable relative error $\tilde{r} := |y - \tilde{y}|/|\tilde{y}|$. Both are intimately related by

$$\frac{\tilde{r}}{1+\tilde{r}} \le r \le \frac{\tilde{r}}{1-\tilde{r}}$$
$$\frac{r}{1+r} \le \tilde{r} \le \frac{r}{1-r}$$

whenever both r and $\tilde{r}$ are less than 1. But a bound for the latter is more easily computable at runtime: it suffices to have a bound for $|y-\tilde{y}|$. In order to guarantee that $r \le \kappa(\nu,p)$ one must impose the stricter $\tilde{r} \le \kappa(\nu,p)/(1+\kappa(\nu,p))$.

9. Lemmas and theorems

Lemma 1. *Assume $a > 0$ and $a+b > 0$ and let there be given n positive numbers μ_i, $i = 1,\dots,n$, that satisfy*

$$\sum_{i=1}^{n} \mu_i = 1. \tag{24}$$

Then

$$(a+b)a^{n-1} \le \prod_{i=1}^{n}(a+\mu_i b) \le \left(a + \frac{b}{n}\right)^n. \tag{25}$$

The lower bound is reached when exactly one of the μ_i is equal to 1 and the others are zero. The upper bound is reached when all μ_i are equal to $1/n$.

Proof. Define the function

$$f(\mu_1,\dots,\mu_n) = -\sum_{i=1}^{n} \log(a+\mu_i b).$$

Because of the conditions on a and b, the argument of each logarithm is strictly positive so f is a bounded, continuous function. The product in (25) reaches a maximum when f reaches a minimum and vice versa. Furthermore, the Hessian of f is a diagonal matrix with positive entries $b^2/(a+\mu_i b)^2$, so f is a convex function.

To minimise f with the equality constraint (24), we consider the function

$$g(\mu_1,\dots,\mu_n,\lambda) = f(\mu_1,\dots,\mu_n) + \lambda(\sum_{i=1}^{n}\mu_i - 1)$$

where λ is a Lagrange multiplier. It is easily checked that the gradient of g is zero only when

$$\mu_1 = \ldots = \mu_n = \frac{1}{n}, \quad \lambda = \frac{b}{a+b/n}$$

and since f is convex and the constraint is linear, this corresponds to a global minimum. This proves the upper bound in (25).

Since g has only one critical point (corresponding to a global minimum), the maximum must occur at the boundary, where one or more of the μ_i are zero. Let us assume, without loss of generality, that $\mu_n = 0$. Minimising the product in (25) then corresponds to minimising a similar product with $n-1$ factors instead of n, so we have reduced the dimension of the problem by 1. Repeating this argument $n-2$ times, we find that $\mu_1 = 1$ and $\mu_2 = \ldots = \mu_n = 0$ (or, since the problem is symmetric in the μ_i, that exactly one of the μ_i is 1 and the others are zero). This proves the lower bound of (25).

Note that the upper bound in (25) is a variation on the well-known fact that the maximum volume of an n-dimensional rectangular parallelepiped whose edges cannot exceed a certain length is that of a hypercube.

Lemma 2. *Let $a \geq 0$. Then*

$$\log(1+a) \geq \frac{a}{1+a} .$$

PROOF. Define $f(a) = \log(1+a) - a/(1+a)$. Then $f'(a) = a/(1+a)^2 \geq 0$ and $f(0) = 0$. So f only increases, starting from zero, which proves the lemma.

Using the previous lemmas it is not so difficult to prove the following result. Before proceeding with the proof, we point out that $1+\kappa$ with $\kappa \geq 0$, also bounds the worst case scenario

$$1 \leq \frac{\prod_{i=1}^{m}(1+|\epsilon_i|)}{\prod_{i=m+1}^{n}(1-|\epsilon_i|)} \leq 1+\kappa.$$

Theorem 1. *Let $\kappa \geq 0$ and n real numbers ϵ_i be given with $n > 1$. Assume that*

$$|\epsilon_i| \leq \frac{\mu_i \kappa}{1+\kappa}, \quad i = 1, \ldots, n$$

with

$$\sum_{i=1}^{n} \mu_i = 1.$$

Then

$$\left| \frac{\prod_{i=1}^{m}(1+\epsilon_i)}{\prod_{i=m+1}^{n}(1+\epsilon_i)} \right| \leq 1+\kappa$$

for any $0 \leq m \leq n$.

PROOF. We first look at the numerator. Use the bound on ϵ_i to write

$$|\prod_{i=1}^{m}(1+\epsilon_i)| \leq \prod_{i=1}^{m}\left(1+\frac{\mu_i\kappa}{1+\kappa}\right).$$

Putting

$$\sum_{i=1}^{m} \mu_i = c \leq 1$$

we use Lemma 1 with μ_i/c instead of μ_i to find

$$\begin{aligned}\prod_{i=1}^{m}\left(1+\frac{\mu_i\kappa}{1+\kappa}\right) &\leq \left(1+\frac{\kappa c}{m(1+\kappa)}\right)^m \\ &\leq \left(1+\frac{\kappa c}{m(1+\kappa c)}\right)^m \\ &= f(m).\end{aligned}$$

Differentiating $f(m)$ and writing (for simplicity) $x = \kappa c/(m(1+\kappa c))$ we get

$$f'(m) = \left[\log(1+x) - \frac{x}{1+x}\right](1+x)^m \geq 0$$

where the last equality holds because of Lemma 2. Furthermore we have that

$$\lim_{m\to\infty} f(m) = \exp\left(\frac{\kappa c}{1+\kappa c}\right) \leq 1 + \kappa c$$

where the last equality again holds because of Lemma 2. It is also easy to see that $f(1) \leq 1 + \kappa c$. Combining the previous statements we see that

$$|\prod_{i=1}^{m}(1+\epsilon_i)| \leq 1 + \kappa c.$$

Now let us bound the denominator. First observe that

$$\left|\frac{1}{1+\epsilon_i}\right| \leq \frac{1}{1-|\epsilon_i|} \leq \frac{1+\kappa}{1+\kappa(1-\mu_i)}$$

so that we obtain

$$\left|\prod_{i=m+1}^{n}\frac{1}{1+\epsilon_i}\right| \leq \frac{(1+\kappa)^{n-m}}{\prod_{i=m+1}^{n}[1+\kappa(1-\mu_i)]} .$$

Apply Lemma 1 to the denominator of this expression with $\mu_i/(1-c)$ instead of μ_i to find that

$$\prod_{i=m+1}^{n}[1+\kappa(1-\mu_i)] \geq (1+\kappa c)(1+\kappa)^{n-m-1}.$$

Combining this with the previous results proves the theorem.

The following lemma is needed to compute the working precision in the continued fraction algorithm.

Lemma 3. *Define the function $f : \mathbb{R}^3 \to \mathbb{R}$ by*

$$f(a,b,c) = \frac{a+4}{8ac}\left(4b + ab + ac - \sqrt{(4b+ab+ac)^2 - 16abc}\right).$$

If $a \geq 0$, $b \geq 1$ and $0 < c \leq 1$ then

$$\frac{1}{2} < f(a,b,c) \leq 1.$$

Proof. First note that the argument of the square root is positive on the given domain (it is positive for $a = 0$ and increases with a), so the function f is indeed real.

Some computations show that

$$\frac{\partial}{\partial b} f(a,b,c) \geq 0,$$

from which it follows that

$$f(a,1,c) \leq f(a,b,c) \leq f(a,\infty,c) = 1,$$

proving the upper bound (equality holds when $a = 0$).

Some more computations yield that

$$\frac{\partial}{\partial c} f(a, 1, c) \leq 0,$$

from which it follows that $f(a, 1, c) \geq f(a, 1, 1)$.

Finally, it is easily shown that $f(a, 1, 1) > 1/2$, with equality in the limit when $a \to \infty$. Combining the previous inequalities gives the lower bound of the lemma, thus finishing the proof.

[1] M. Abramowitz, I. Stegun, Handbook of mathematical functions with formulas, graphs and mathematical tables, U.S. Government Printing Office, NBS, Washington, D. C., 1964.

[2] F. W. J. Olver, D. W. Lozier, R. F. Boisvert, C. W. Clark, NIST Digital Library of Mathematical Functions, http://dlmf.nist.gov/ (online companion to [23] with errata list and other updates).

[3] B. A. Cipra, A New Testament for Special Functions, SIAM News 31 (2), march (1998).

[4] C. B. Moler, Cleve's Corner: The Tetragamma Function and Numerical Craftsmanship: MATLAB's special mathematical functions rely on skills from another era, Technical note, The MathWorks, Inc. (2002).
URL `http://www.mathworks.com/company/newsletter/clevescorner/winter02_cleve.shtml`

[5] Floating-Point Working Group, IEEE standard for binary floating-point arithmetic, SIGPLAN 22 (1987) 9–25.

[6] J.-M. Muller, Elementary functions: Algorithms and implementation, 2nd Edition, Birkhäuser, 2006.

[7] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, P. Zimmermann, MPFR: a multiple-precision binary floating-point library with correct rounding, ACM Trans. Math. Software 33 (2) (2007) Art. 13, 15.

[8] R. Brent, A FORTRAN multiple-precision arithmetic package, ACM Trans. Math. Software 4 (1978) 57–70.

[9] D. Bailey, A FORTRAN 90-based multiprecision system, ACM Trans. Math. Software 21 (1995) 379–387.

[10] D. Smith, Algorithm 786: Multiple-precision complex arithmetic and functions, ACM Trans. Math. Software 24 (1998) 359–367.

[11] A. Cuyt, B. Verdonk, H. Waadeland, Efficient and reliable multiprecision implementation of elementary and special functions, SIAM J. Sci. Comput. 28 (2006) 1437–1462.

[12] L. Lorentzen, A priori truncation error bounds for continued fractions, Rocky Mountain J. Math. 33 (2) (2003) 409–474.

[13] M. Colman, A. Cuyt, J. Van Deun, Validated computation of certain hypergeometric functions, ACM Trans. Math. Software 38, article nr. 11.

[14] A. Cuyt, V. Brevik Petersen, B. Verdonk, H. Waadeland, W. Jones, Handbook of Continued Fractions for Special Functions, Springer Verlag, 2008.

[15] IEEE Std 754-2008, IEEE Standard for Floating-Point Arithmetic, IEEE, New York, NY, USA, 2008.

[16] F. Backeljauw, A Library for Radix-Independent Multiprecision IEEE-Compliant Floating-Point Arithmetic, Tech. Rep. 2009-01, Universiteit Antwerpen (2009).

[17] N. J. Higham, Accuracy and Stability of Numerical Algorithms, 2nd Edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002.

[18] J. F. Hart, Computer approximations, R.E. Krieger, 1978.

[19] D. Lozier, F. Olver, Numerical Evaluation of Special Functions, in: W. Gautschi (Ed.), AMS Proceedings of Symposia in Applied Mathematics, Vol. 48, 1994, pp. 79–125, updated version (december 2000) available at `http://math.nist.gov/nesf/`.

[20] L. Lorentzen, H. Waadeland, Continued fractions with applications, North-Holland Publishing Co., Amsterdam, 1992.

[21] J. Gill, Infinite Compositions of Möbius Transformations, Trans. Amer. Math. Soc. 176 (1973) 479–487.

[22] W. Jones, W. Thron, Numerical stability in evaluating continued fractions, Math. Comp. 28 (1974) 795–810.

[23] F. W. J. Olver, D. W. Lozier, R. F. Boisvert, C. W. Clark, NIST Handbook of Mathematical Functions, Cambridge University Press, New York, NY, 2010, print companion to [2].