Reliable Multiprecision Evaluation of Special Functions

Stefan Becuwe[1], Annie Cuyt
Departement Wiskunde en Informatica,
Universiteit Antwerpen,
Middelheimlaan 1, B-2020 Antwerpen, Belgium;
{*stefan.becuwe, annie.cuyt*}*@ua.ac.be*

Introduction

Special functions are pervasive in all fields of science and industry. The most well-known application areas are in physics, engineering, chemistry, computer science and statistics. Because of their importance, several books and a large collection of papers have been devoted to algorithms for the numerical computation of these functions.

Virtually all present-day computer systems, from personal computers to the largest supercomputers, implement the IEEE 64-bit floating-point arithmetic standard, which provides 53 binary or approximately 16 decimal digits accuracy. For most scientific applications, this is more than sufficient. However, for a rapidly expanding body of applications, 64-bit IEEE arithmetic is no longer sufficient. These range from some interesting new mathematical investigations to large-scale physical simulations performed on highly parallel supercomputers. Moreover in these applications, portions of the code typically involve numerically sensitive calculations, which produce results of questionable accuracy using conventional arithmetic. These inaccurate results may in turn induce other errors, such as taking the wrong path in a conditional branch. Such blocks of code benefit enormously from a combination of reliable numeric techniques and the use of high-precision arithmetic. Indeed, the aim of reliable numeric techniques is to deliver, together with the computed result, a guaranteed upper bound on the total error or, equivalently, to compute an enclosure for the exact result.

Instead of high accuracy, some applications only require a very modest but guaranteed number of significant digits. These applications can profit from a reliable implementation with scalable precision. For instance, in electromagnetic simulation models the required accuracy is usually in the order of only 2 to 3 significant digits.

Up to this date, even environments such as Maple, Mathematica, MATLAB and libraries such as IMSL, CERN and NAG offer no routines for the reliable evaluation of special functions. The following quotes concisely express the need for new developments in the evaluation of special functions:

- *"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 director of the DLMF project, and Frank Olver [2].
- *"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 [5].

Implementing a Function Library

The realization of a machine implementation of a function $f(x)$ is a three-step process.

1. For a given argument x, the evaluation $f(x)$ is often reduced to the evaluation of f for another argument $\tilde{x}$ lying within specified bounds and for which there exists an easy relationship between $f(x)$ and $f(\tilde{x})$. For instance, for the exponential function in a base β implementation,

$$\exp(x) = \beta^k \exp(\tilde{x}),$$
$$\tilde{x} = \mathrm{mod}(x, \ln \beta), \qquad |\tilde{x}| \leq \ln \frac{\beta}{2}.$$

Although the given argument x is known exactly, because it is a given floating-point number, usually the reduced argument $\tilde{x}$ cannot be computed exactly, but is subject to a rounding error. The issue of argument reduction is a topic in its own right and mostly applies to only the simplest transcendental functions such as the elementary functions.

2. After the reduced argument is determined, the mathematical model F for f is constructed and a truncation error

$$\frac{|f(\tilde{x}) - F(\tilde{x})|}{|f(\tilde{x})|}$$

comes into play, which needs to be bounded.

3. When implemented, in other words evaluated as $\mathtt{F}(\tilde{x})$, this mathematical model is also subject to a rounding error

$$\frac{|F(\tilde{x}) - \mathtt{F}(\tilde{x})|}{|f(\tilde{x})|},$$

which needs to be controlled.

[1]The author is supported by the Institute for the Promotion of Innovation through Science and Technology in Flanders.

Finally the effect of switching from the argument x to the reduced argument $\tilde{x}$ must be taken into account. This introduces a final additional error.

Toolkit for a Reliable Library

The technique to provide a floating-point model $F(x)$ of a function $f(x)$ differs substantially when going from a fixed finite precision context to a finite multiprecision context. In the former, the aim is to provide an optimal mathematical model, valid on a reduced argument range and requiring as few operations as possible. Here optimal means that, in relation to the model's complexity, the truncation error is as small as it can get. The total relative error should not exceed a prescribed threshold, round-off error and possible argument reduction effect included. In the latter, the goal is to provide a more generic technique, from which an approximant yielding the user-defined accuracy, can be 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. At the same time the generic technique should generate an approximant of as low complexity as possible.

We aim, on the one hand, at a generic technique suitable for use in a multiprecision context, which on the other hand, is efficient enough to compete with the traditional hardware algorithms. We also want our implementation to be reliable, in the sense that a sharp interval enclosure for the requested function evaluation is returned without any additional cost.

Besides series representations, 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, meaning that the partial numerators and denominators converge.

It is well-known that the tail or rest 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 need to converge to zero; it does not even need to converge at all. In order to develop a useful continued fraction technique, we first need to obtain sharp a priori truncation error estimates for a general class of continued fractions, taking into account that a suitable approximation of the disregarded continued fraction tail may speed up the convergence of the continued fraction approximants. Hence the truncation error estimate needs to be valid for use with nonzero continued fraction tail estimates. Such estimates are developed in the framework of this project [3]. The rounding error involved can subsequently be bounded by a classical result obtained in [4].

Special Function Coverage

The implementation will be made available in two forms: as a C/C++ library and as a Maple library. Both will make use of the fully IEEE 754-854 compliant multiprecision library `MpIeee`, in which the user can select the base β, precision t and exponent range $[L, U]$ of the computations. We aim, for the evaluation of all functions, at a relative error $|f(x) - \mathtt{F}(x)|/|f(x)|$ bounded above by 1 ULP (Unit-in-the-Last-Place) or $\beta^{-(t-1)}$.

Which special functions will be supported? Among the special functions that enjoy rapidly converging limit-periodic continued fraction representations are the ones listed in the table below. In the column marked AS, we indicate whether the standard work [1] contains at least one continued fraction representation for the function in question. The second column, marked CFHB, tells us whether our new handbook [3] contains a useful continued fraction representation for the purpose. In the third column we have added for which special functions an implementation is (✓) or will be (✓*) available soon.

	AS	CFHB	
elementary functions	✓	✓	✓
$\psi_1(z)$, $\psi_2(z)$		✓	✓*
$\gamma(a, z)$		✓	✓
$\Gamma(a, z)$	✓	✓	✓
$\mathrm{erf}(z)$	✓	✓	✓
$\mathrm{erfc}(z)$		✓	✓
$C(z)$, $S(z)$ (Fresnel)		✓	
$E_n(z)$	✓	✓	✓
${}_2F_1(a, b; c; z)$		✓	✓
${}_1F_1(a; b; z)$		✓	✓
$J_\nu(z)$ (Bessel)	✓	✓	✓*
$I_\nu(z)$ (Bessel)		✓	✓*
$I_x(a, b)$ (beta)	✓	✓	✓*

References

[1] M. Abramowitz and I. A. Stegun, editors. *Handbook of mathematical functions with formulas, graphs, and mathematical tables*, volume 55 of *NIST*. 1964.

[2] B. A. Cipra. A new testament for special functions? *SIAM News*, 31(2), 1998.

[3] A. Cuyt, V. Petersen, B. Verdonk, H. Waadeland, W. B. Jones, and C. Bonan-Hamada. *Handbook of continued fractions for special functions*. Kluwer Academic Publishers, 2006.

[4] W. B. Jones and W. J. Thron. Numerical stability in evaluating continued fractions. *Math. Comp.*, 28:795–810, 1974.

[5] C. Moler. The tetragamma function and numerical craftmanship. *MATLAB News & Notes*, 2002.