Applied and
Computational
Mathematics
Division

NISTIR 5916

Computing and Applied Mathematics Laboratory

A Proposed Software Test Service for Special Functions

Daniel W. Lozier

October 1996

U. S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards and Technology
Gaithersburg, MD 20899

PREPRINT

This paper will appear in final form in *The Quality of Numerical Software: Assessment and Enhancement*, Ronald F. Boisvert, editor, Chapman & Hall, London, 1997. It was presented orally at the IFIP/TC2/WG2.5 Working Conference 7, held at St. Catherine's College, Oxford, England, July 7–13, 1996.

ABSTRACT

This is a proposal to develop a software test service at the National Institute of Standards and Technology for use in testing the accuracy, or numerical precision, of mathematical software for special functions. The service would use the World Wide Web to receive test requests and return test results. The tests would be run on a network of workstations at the Institute. It is hoped that such a service will be of practical utility to anyone who uses special functions in physics or other applications, and that it will stimulate the interest of applied mathematicians who are interested in the computation of special functions as well as computer scientists who are interested in innovative uses of the Internet. The author solicits comments on any aspect of the proposed service.

1 Introduction

Mathematical software is deeply embedded in the computing environment. Since this environment is evolving rapidly, its impact on mathematical software needs to be revisited regularly.

Progress in parallel computing has stimulated much reworking of numerical algorithms, particularly in computational linear algebra. The earlier introduction of vector computers had a similar effect. Recent advances in communications and networking have led to the global interconnection of computers by high bandwidth communication links, one result of which is improved access to information about mathematical software via the Internet and World Wide Web. For example, electronic catalogs and repositories such as *xnetlib* [8] are now consulted routinely for help in locating and obtaining mathematical software packages for specific tasks.

Vector and parallel developments have had only a modest impact on the computation of mathematical functions. Some references can be cited, for example [4] and [10]. However, mathematical functions seem particularly appropriate for demonstrating a new and potentially valuable use of the Web: mathematical software testing. The problem of testing is intrinsically simpler for mathematical functions than for other kinds of numerical computation. The input and output Euclidean spaces have low dimension. In contrast, numerical linear algebra deals with Euclidean spaces of high dimension, and most other numerical computations deal with function spaces of infinite dimension. Test procedures for mathematical functions can be devised that apply, in theory, to all possible inputs.

A question that needs answering at this point is: What would be the advantages, compared to current testing practice, in using the Web to test mathematical software? Our answer is that tests can be tailored to suit a particular need, and they can be performed on demand.

To see why this is useful, we divide current practice into two categories. *Supplier testing* is performed by the software writer or project team. Referees, editors and software maintainers also play a role. These people have direct responsibility for the correctness of the software. For commercial software, high license fees are justified largely by the high costs associated with software maintenance and testing. *Independent testing* is performed by other individuals and groups. Users often perform this kind of testing for their own purposes because it leads to an increased confidence in the correctness of the software. Sometimes independent tests are conducted and published in journal articles and institutional reports as a guide for prospective users.

Published tests of either kind have a 'frozen-in-time' quality about them, having been performed at some time in the past in a computing environment that may be very different from the prevailing one. Even more unsettling, since tests are never complete, their results may not apply directly to the numerical computation of current interest.

Just the simple repetition of a mathematical function test is often impractical because the test program is unavailable. Notable exceptions are the test programs of Cody and Waite [7] and Cody [5] for elementary functions of real and complex arguments, and of Cody [6] for special functions of real argument. Here we are following the conventional terminology of calling the transcendental functions met in calculus courses *elementary functions* and the higher functions that appear in advanced applications *special functions*. The general unavailability of test programs is undoubtedly related to the considerable effort that is required to raise them to an acceptably high standard for publication or public distribution. Another problem is lack of generality. For example, most test programs apply only to a builtin set of test arguments, often with an element of randomness included.

In this paper a software test service for special functions is proposed and some implementational details are given. The emphasis is on special functions, because this is where the need is greatest, but the service will apply equally well to elementary functions. The service will provide a tool that can be used to tailor tests to specific requirements. Therefore it should be of interest in both supplier and independent testing.

2 Proposal

The purpose of the proposed software testing service for special functions is to assess the accuracy, or numerical precision, of computed function values through the use of a comparison method. Test values will be compared against reference values computed in higher precision by highly accurate algorithms. Test requests will be submitted to a Web server at the National Institute of Standards and Technology. The tests will be conducted at the Institute using software developed for the purpose. The test results will be returned to the requester in the form of an appropriate document on the Web server.

The key components of the service will be

Reference Software This will consist of highly accurate and reliable, but not necessarily efficient, numerical procedures for generating high-precision reference values of special functions over very extensive argument domains. The reference software will be an excellent repository for advanced algorithms because it will be embedded in a computing environment that mitigates the computer arithmetic liabilities (underflow, overflow, and limited precision) of conventional computing environments.

Comparison Software This will serve the purpose of orchestrating the generation of reference values and determining the precision of test values. The comparison software will utilize parallel methods via the simple device of domain partitioning. An appropriate measure of precision will be defined in terms of interval mathematics.

Communication Interface This will be an appropriately designed Web document with associated subdocuments for accepting test requests and returning test results via the Internet.

The comparison method was chosen because of its conceptual simplicity. An alternative approach, numerical verification of identities, is advocated and used by W. J. Cody and his coworkers. It avoids the need for higher precision but it requires careful attention in the choice of identity to guarantee against incorrect conclusions that could arise if the identity were not entirely independent of the algorithm used in the implementation of the function. Also, care must be taken to separate the error in the function evaluation from the error in the evaluation of the identity. These complications will be avoided in the test service by taking full advantage of the tremendous power of current capabilities for computation and communication.

3 Reference Software

The reference software is at the heart of the proposed software testing system for special functions. Not only must it be highly accurate, a definite bound on the error in each computed reference value is essential. Otherwise, no one can be certain of the results of a test. For this reason, the reference software should be written using interval techniques. An introduction to interval computations is given in the book by Alefeld and Herzberger [2]. However, aside from the elementary functions, very little has been published on interval algorithms for specific mathematical functions. An opportunity and a need exists here for numerical analysts to develop interval algorithms that generate the required error bounds.

The service must be able to test double-precision as well as single-precision software. Thus it is appropriate to write the reference software in multiple precision. The Fortran package of Bailey [3] is available and applicable for this purpose. Because of its vast exponent range in comparison to conventional computer arithmetic systems, Bailey's package relieves the need to be careful about underflow and overflow. The occurrence of these conditions can completely invalidate an otherwise pristine computation. The algorithms will take fully into consideration stability and roundoff questions because these too, if ignored, can destroy a computation.

Highly efficient software, at least for functions of one variable, is usually precision-limited because it employs polynomial or rational approximations that are constructed with respect to a fixed target precision. Flexibility is more important than efficiency for reference software. Ideally, reference algorithms will accept an arbitrary tolerance specification so that the same programs can be executed in increased precision without a major effort to generate approximation coefficients for the higher precision. This means that methods will be

constructed from Taylor expansions, asymptotic expansions, differential or difference equations, integral representations, and other analytical properties, just as is done in much existing software for functions of two or more variables.

4 Comparison Software

The comparison software has a mathematical component and a computer science component. The mathematical component is concerned with measuring the error in test values. This could be done simply with pointwise absolute or relative error but an interval formulation is more appropriate. The computer science component is needed to collect and process the test and reference values. This is a natural application for parallel processing with a loosely coupled network of computer workstations.

Only the mathematical component will be considered here. It is easy to describe, at least when all variables are real. Let us consider a function

$$y = f(x), \qquad x \in \mathcal{R}^m, \qquad y \in \mathcal{R}, \tag{1}$$

where $\mathcal{R}$ denotes the set of real numbers. Let $\mathcal{F}$ be the set of real numbers that are representable exactly in the format of a particular computer arithmetic system, excluding any nonnumerical symbolic representations such as $\pm\infty$, ± 0 and NaN (Not-a-Number). Thus an approximating function

$$\tilde{y} = \tilde{f}(x), \qquad x \in \mathcal{F}^m, \qquad \tilde{y} \in \mathcal{F} \tag{2}$$

is defined by the software to be tested. Our problem is to measure the error committed when $\tilde{y}$ is taken as an approximation to y.

The pointwise absolute error, defined for $x \in \mathcal{F}^m$, is just $|y - \tilde{y}|$. Because absolute error is naturally associated with fixed-point computation, and not floating-point, relative error is more appropriate except near zeros of the function f. Instead of relative error $\rho = |(\tilde{y}-y)/y|$, we prefer to use *relative precision*

$$\mathrm{rp}(y, \tilde{y}) = \begin{cases} |\ln(\tilde{y}/y)| & \text{if } y\tilde{y} > 0, \\ \text{undefined} & \text{otherwise;} \end{cases} \tag{3}$$

this definition was introduced in [13]. Since $\mathrm{rp}(y, \tilde{y}) = \rho + O(\rho^2)$, relative precision and relative error are nearly the same when $\tilde{y}$ is a good approximation to y. But relative precision has the advantage for detailed error analyses that it is a metric on $\mathcal{R}^+$ and $\mathcal{R}^-$, where these symbols denote the open real intervals $(0, \infty)$ and $(-\infty, 0)$, respectively.

Given $x \in \mathcal{F}^m$, the exact function value $y = f(x)$ determines the interval $Y = [y_\ell, y_u]$ where y_ℓ, y_u are two consecutive elements of $\mathcal{F}$. A criterion that is applied sometimes in the construction of computer software is to require the approximate function value $\tilde{y} = \tilde{f}(x)$ to satisfy either $\tilde{y} = y_\ell$ or $\tilde{y} = y_u$. This

will be called the *criterion of full precision.* It can be expressed in a different way. First we define the *machine epsilon*

$$\epsilon = \max_{t \in \mathcal{F}} \mathrm{rp}(t, t^+). \tag{4}$$

where t^+ denotes the successor of t in $\mathcal{F}$. Then the approximating function $\tilde{f}$ satisfies the criterion of full precision if, and only if, $\mathrm{rp}(y, \tilde{y}) \leq \epsilon$ for all $x \in \mathcal{F}^m$ such that $f(x)$ and $\tilde{f}(x)$ have the same sign. It is customary to employ full precision, or nearly full precision, in software for elementary functions.

The strongest possible criterion is correct rounding. Required by [9] for standard floating-point arithmetic operations, it is met by most up-to-date implementations of floating-point arithmetic as well as by some software for elementary functions, particularly when supplied with Fortran compilers.

The criterion of full precision is quite rigorous. If $x \in \mathcal{F}^m$, let x^- and x^+ denote its predecessor and successor (where the ordering is defined componentwise). If we regard x as a *representative* of the multivariate interval

$$X = [x_\ell, x_u] = \frac{1}{2}[x + x^-, x + x^+], \tag{5}$$

then the uncertainty in x is reflected in the range of f as its arguments vary throughout X. If a partial derivative of f is large, it can be argued that it is unnecessary to require full precision in $\tilde{f}(x)$. In fact, the computation of $\tilde{f}(x)$ to full precision is unwarranted if it requires an inordinate amount of execution time. This penalty is likely to be especially severe for special functions.

A more appropriate criterion of precision can be defined. Let $I(\mathcal{R})$ be the set of all closed interval subsets of $\mathcal{R}$. Alefeld and Herzberger [2] show that, if $A = [a_\ell, a_u]$ and $B = [b_\ell, b_u]$ are two intervals, then the function

$$q(A, B) = \max\{|a_\ell - b_\ell|, |a_u - b_u|\}, \qquad A, B \in I(\mathcal{R}) \tag{6}$$

is a metric. Also, since $q([a, a], [b, b]) = |a - b|$, the metric q generalizes the usual metric in $\mathcal{R}$. Arithmetic operations $A + B, A - B, AB$ and A/B are defined in $I(\mathcal{R})$ by operating on the endpoints of the intervals. They are continuous in the topology of $\{I(\mathcal{R}), q\}$. Similarly, it is possible to define continuous interval extensions of continuous real functions. For example, for the logarithmic function, the interval extension $\ln(A) = [\ln a_\ell, \ln a_u]$ is defined and continuous on $I(\mathcal{R}^+)$.

Next, we define *interval relative precision*

$$\mathrm{rp}(Y, \tilde{Y}) = \begin{cases} q(\ln Y, \ln \tilde{Y}) & \text{if } Y, \tilde{Y} \in I(\mathcal{R}^+), \\ \mathrm{rp}(-Y, -\tilde{Y}) & \text{if } Y, \tilde{Y} \in I(\mathcal{R}^-), \\ \text{undefined} & \text{otherwise.} \end{cases} \tag{7}$$

This is easy to compute, since it can be shown that

$$\mathrm{rp}(Y, \tilde{Y}) = \max\{\mathrm{rp}(y_\ell, \tilde{y}_\ell), \mathrm{rp}(y_u, \tilde{y}_u)\}. \tag{8}$$

Interval relative precision is a metric on $I(\mathcal{R}^+)$ and $I(\mathcal{R}^-)$, and it generalizes pointwise relative precision since $\mathrm{rp}([y,y],[\tilde{y},\tilde{y}]) = \mathrm{rp}(y,\tilde{y})$.

Now consider the test argument $x \in \mathcal{F}^m$, again as a representative of the multivariate interval $X = [x_\ell, x_u]$, and assume the function f is continuous on X. Let Y be the range of f on X:

$$Y = [y_\ell, y_u] = f(X) = \{f(x) \mid x_\ell \le x \le x_u\}. \tag{9}$$

Finally, let the test function value $\tilde{y} = \tilde{f}(x) \in \mathcal{F}$ represent the interval

$$\tilde{Y} = [\tilde{y}_\ell, \tilde{y}_u] = \frac{1}{2}[\tilde{y} + \tilde{y}^-, \tilde{y} + \tilde{y}^+]. \tag{10}$$

Then we will say that the approximating function $\tilde{f}$ satisfies the *interval criterion of precision* if

$$\mathrm{rp}(Y, \tilde{Y}) \le \max\{\epsilon, \mathrm{rp}(y_\ell, y_u)\} \tag{11}$$

for all x such that the relative precisions are defined. The right side of this inequality provides a standard of comparison. It takes into account the behavior of f as its arguments vary throughout the neighborhood represented by x. It establishes the allowable range of relative errors over this neighborhood. The left side measures the distance between the allowable range of f and the interval represented by the test function value. If the interval criterion (11) is satisfied, then the set intersection $Y \cap \tilde{Y}$ is nonempty. If $\tilde{Y} \subseteq Y$ or $Y \subseteq \tilde{Y}$, then (11) is satisfied. In all cases when (11) is satisfied, a simple interpretation in terms of pointwise relative error can be given. This will be discussed in a future paper.

A fundamental problem in interval mathematics is how to compute the range of real functions. Evaluation of explicit expressions using interval arithmetic does not necessarily produce the range; to the contrary, the range may be overestimated substantially. Also, it is necessary to construct strict bounds for all errors caused by truncating infinite processes. This problem will need to be faced in the design and construction of reference software for the software test service.

5 Communication Interface

For the software test system, an *argument set* is a subset of the domain of a function. For each test, the test requester provides an argument set together with corresponding function values to the communication interface. Then the reference software computes the function to higher precision at all points in the argument set, the comparison software compares the reference values against the test requester's function values, and finally the communication interface returns a table or plot of the interval relative precision to the test requester.

A careful development of the software test service requires attention to the processes of decimal-to-binary and binary-to-decimal conversion. Base conversion processes between arbitrary bases are considered in detail in Matula [11]

and [12]. A p-digit, base-β *significance space* is the set S_β^p of all p-digit normalized floating-point numbers in the base β, excluding zero and without regard to size. Let S_β^p and S_ν^q be two significance spaces. The *rounding conversion mapping* R_ν^q from S_β^p into S_ν^q is the mapping that is defined by converting $x \in S_\beta^p$ into its ν-ary expansion to sufficiently high precision, then rounding it to q base-ν digits. The *truncation conversion mapping* T_ν^q is defined similarly. The composition of base conversion mappings is possible. An interesting kind of composition is an *in-and-out conversion mapping* which maps S_β^p into S_ν^q, and then back to S_β^p. Matula proved two theorems:

Theorem 1 (Base Conversion Theorem) *If $\beta^i \neq \nu^j$ for any positive integers i, j, then the base conversion mappings $R_\nu^q : S_\beta^p \to S_\nu^q$ and $T_\nu^q : S_\beta^p \to S_\nu^q$ are:*

1. *one-to-one onto their ranges if and only if $\nu^{q-1} \geq \beta^p - 1$;*
2. *onto if and only if $\beta^{p-1} \geq \nu^q - 1$.*

Theorem 2 (In-and-Out Conversion Theorem) *If $\beta^i \neq \nu^j$ for any positive integers i, j, then*

1. *$R_\beta^p R_\nu^q : S_\beta^p \to S_\beta^p$ is the identity if and only if $\nu^{q-1} > \beta^p$, and*
2. *$R_\beta^p T_\nu^q : S_\beta^p \to S_\beta^p$ is the identity if and only if $\nu^{q-1} \geq 2\beta^p - 1$.*

The condition $\beta^i \neq \nu^j$ for any positive integers i, j excludes the trivial case when the bases β and ν are integral powers of a common base. Under the conditions of Theorem 2, R_ν^q and T_ν^q are one-to-one onto their ranges and their inverse mappings coincide with $R_\beta^p : S_\nu^q \to S_\beta^p$.

As an example, consider decimal output from single-precision computer arithmetic as defined in [9]. Then $\beta = 2$, $p = 24$, $\nu = 10$, and q is to be determined according to some criterion. The rounding conversion mapping $R_{10}^q : S_2^{24} \to S_{10}^q$ is

1. one-to-one onto its range if and only if $q \geq 9$;
2. onto if and only if $q \leq 6$;

similarly for T_{10}^q. Thus decimal output precision q need not exceed 9 digits if each internal number is to have a unique decimal representation, and it cannot exceed 6 digits if the complete set of q-digit decimal numbers is to be covered. Also, either of $R_2^{24} R_{10}^q$ or $R_2^{24} T_{10}^q$ is the identity mapping if and only if $q \geq 9$.

Now let S_β^p denote the significance space associated with the test requester's computer arithmetic $\mathcal{F}$. Let S_{10}^q be the decimal significance space with minimum q such that the necessary and sufficient condition in part 1 of Theorem 2 is satisfied. Finally, let $S_{\beta'}^{p'}$ denote the significance space associated with the

reference software. We assume β' and β are positive integral powers of two, and we assume p' is such that $S_\beta^p \subset S_{\beta'}^{p'}$.

Then the In-and-Out Conversion Theorem allows us to choose whether to represent an argument set in binary or decimal. If we choose binary, the base conversion mapping from S_β^p to $S_{\beta'}^{p'}$ is trivial. It must be noted, however, that this choice leads to programming complications that are not entirely trivial. Also, conversion to decimal is necessary for human interpretation. Therefore it seems that the decimal choice should be considered. Assuming that rounding conversion mappings are correctly implemented in the computing environment of the test requester and also in the computing environment used by the software test service, it is immaterial (except possibly for practical concerns involving execution speed) whether argument sets are represented in binary or decimal. The same remark is true concerning other test data, such as computed function values, that are passed through the communication interface from $\mathcal{F}^m$ to the reference software or vice versa.

To summarize, a test requester will want to consider argument sets as originating in one of two ways:

Decimal Origination The function $y = f(x)$ is to be tested to obtain a general impression of its accuracy over parts of its domain. Here knowledge of exact binary representations is not important, so it is natural to specify the argument set in decimal.

Binary Origination The function $y = f(x)$ is to be tested at a set of exactly machine-representable arguments. For example, if f is used in an application program, it might be useful to have the capability of testing f at the exact arguments that arise in a particular program execution. Here decimal representation is still permissible, provided the conditions of Theorem 2 are met.

For decimal origination, use of a *test generator* avoids the need to supply arguments explicitly. Let $s : [0,1]^m \to \mathcal{R}^m$ be a monotonic function, where monotonicity is defined componentwise. To generate J test arguments, the formula

$$x_j = s(j/(J+1)), \qquad j = 1, 2, \ldots, J \tag{12}$$

is used. Examples of univariate test generators are the *equidistant generator* $s(t) = x_0(1-t) + x_J t$ and the *logarithmic generator* $s(t) = x_0^{1-t} x_J^t$. These generators produce J test arguments in the x-interval $[x_0, x_J]$ with equidistant or logarithmic spacing. An element of randomness is introduced by using a pseudo-random number generator to produce a t-sequence $t_1 < t_2 < \ldots < t_J$ instead of the t-sequence defined by $t_j = j/(J+1)$. With sufficiently careful coding of the software test service, and assuming all rounding base conversion mappings are implemented correctly, test generators will produce identical argument sets when executed either by the test requester or the software test service.

We conclude this section by introducing a general approach that can be used to specify argument sets on a proper submanifold of the function domain. Let

$$D_f = D_1^{(f)} \times D_2^{(f)} \times \ldots \times D_m^{(f)} \subseteq \mathcal{R}^m. \tag{13}$$

denote the domain of a function f. If the dimension m exceeds 1, it may be desirable to hold one or more variables fixed for the duration of the test. The chief reason for such a procedure is that some of the variables may be fixed in the application that gave rise to the test. A straightforward approach would be to list the argument set with the components corresponding to fixed variables remaining constant throughout. However, another approach avoids this unnecessary specification of fixed variables. It has the advantage that it can be generalized to permit testing on a k-dimensional submanifold in the m-dimensional domain, where $k < m$, and it also can be used to change the coordinate system.

First we reorder the variables so that the first k of them in the new order are the ones that will vary; the remaining $m-k$ are held constant. We suppose that k is given such that $1 \leq k \leq m$. Let ρ be a permutation, or rearrangement, of $(1, 2, \ldots, m)$, and let $\overline{\rho} = \rho^{-1}$. Denoting the reordered variables $\xi_1, \xi_2, \ldots, \xi_m$, we have

$$\xi_r = x_{\rho_r}, \qquad x_r = \xi_{\overline{\rho}_r} \qquad (r = 1, 2, \ldots, m), \tag{14}$$

and the test is applied to the function

$$y = g(\xi) = f(x), \qquad \xi \in \mathcal{R}^k, \qquad x \in \mathcal{R}^m, \qquad y \in \mathcal{R}; \tag{15}$$

compare Eq. (1). The test requester provides the integer k, the permutation ρ, the fixed arguments $\xi_{k+1}, \xi_{k+2}, \ldots, \xi_m$, and the argument set in the domain $D_g = D_1^{(g)} \times D_2^{(g)} \times \ldots \times D_k^{(g)}$.

Let us consider as an example the *incomplete gamma function*

$$\gamma(a, z) = \int_0^z e^{-t} t^{a-1}\, dt \qquad (\Re a > 0). \tag{16}$$

Define $a = x_1 + ix_2$, $z = x_3 + ix_4$. Then

$$f(x_1, x_2, x_3, x_4) = \gamma(x_1 + ix_2, x_3 + ix_4). \tag{17}$$

Suppose a test is wanted in which x_1 is held constant. Then $k = 3$, the required permutations are $\rho = (2, 3, 4, 1)$, $\overline{\rho} = (4, 1, 2, 3)$, and the function g is defined by

$$g(\xi_1, \xi_2, \xi_3) = \gamma(\xi_4 + i\xi_1, \xi_2 + i\xi_3). \tag{18}$$

Alternatively, suppose a test is wanted in which the variables are restricted to real values. Then $\rho = \overline{\rho} = (1, 3, 2, 4)$ and $g(\xi_1, \xi_2) = \gamma(\xi_1 + i\xi_3, \xi_2 + i\xi_4)$.

6 Concluding Remarks

The proposed software test service is undergoing active development at the National Institute of Standards and Technology. The initial emphasis is on the construction of the communication interface and associated Web documents. This substantial programming task is being accomplished with the assistance of M. A. McClain of the Applied and Computational Mathematics Division.

The envisioned communication interface will be accessed as a Web 'home page' for the test service. It will present a menu of functions from which the test requester will choose by clicking the mouse. Initially at least, the menu will follow the classification that is used in [1]. Special functions are subject to alternative definitions arising from varying normalization criteria, modification by scaling functions, and other practical or theoretical considerations. This has posed an identification problem in existing software for evaluating and testing special functions because of the severely restricted character set used in computing. An important feature of Web documents is that they support the full range of mathematical notation. This feature is being used to avoid any ambiguity in the identification of functions in our software test service. It also facilitates the possibility of offering a wide range of alternative function definitions for testing.

The software test service will be able to supply numerical function values on demand as well as to evaluate software. Thus the comparison software will not be relevant to all usage of the service. The reference software is essential to all usage, and it is very demanding to provide. It will require a long-term research and development effort. However, symbolic computing environments exist that support numerical computing to arbitrary precision. Some have extensive support for special functions, including computing numerical values to high precision. Initially at least, these environments will be used to supply reference values. Although they do not meet our requirements for the provision of strict error bounds, they probably represent the best currently available source of reference software.

Finally, in view of the new approach to testing that is introduced in this paper, readers may have opinions, recommendations or criticisms that would be useful in improving the proposed software test service. The author would be most grateful for receiving all such comments.

References

[1] M. Abramowitz and I. A. Stegun, editors. *Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables*, volume 55 of *National Bureau of Standards Applied Mathematics Series*. US Government Printing Office, Washington, DC, 1964.

[2] G. Alefeld and J. Herzberger. *Introduction to Interval Computations*. Academic Press, 1983.

[3] D. H. Bailey. Algorithm 719: Multiprecision translation and execution of FORTRAN programs. *ACM Trans. Math. Software*, 19:288–319, 1993.

[4] R. F. Boisvert and B. V. Saunders. Portable vectorized software for Bessel function evaluation. *ACM Trans. Math. Software*, 18:456–469, 1992. For corrigendum see same journal v. 19 (1993), p. 131.

[5] W. J. Cody. Algorithm 714. CELEFUNT*:* A portable test package for complex elementary functions. *ACM Trans. Math. Software*, 19:1–21, 1993.

[6] W. J. Cody. Algorithm 715. SPECFUN*:* A portable Fortran package of special function routines and test drivers. *ACM Trans. Math. Software*, 19:22–32, 1993.

[7] W. J. Cody and W. Waite. *Software Manual for the Elementary Functions.* Prentice Hall, 1980.

[8] J. Dongarra, T. Rowan, and R. Wade. Software distribution using xnetlib. *ACM Trans. Math. Software*, 21:79–88, 1995.

[9] IEEE. IEEE standard for binary floating-point arithmetic. ANSI/IEEE Std 754-1985, The Institute of Electrical and Electronics Engineers, New York, 1985.

[10] D. W. Lozier and F. W. J. Olver. Airy and Bessel functions by parallel integration of ODEs. In R. F. Sincovec et al., editors, *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, volume 2, pages 531–538. Society for Industrial and Applied Mathematics, Philadelphia, 1993.

[11] D. W. Matula. Base conversion mappings. In *Proceedings of the 1967 Spring Joint Computer Conference*, volume 30, pages 311–318, 1967.

[12] D. W. Matula. In-and-out conversions. *Comm. ACM*, 11:47–50, 1968.

[13] F. W. J. Olver. A new approach to error arithmetic. *SIAM J. Numer. Anal.*, 15:368–393, 1978.