Applied and
Computational
Mathematics
Division

NISTIR 5490

Computing and Applied Mathematics Laboratory

Software Needs in Special Functions

Daniel W. Lozier

August 1994

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

THIS PAGE INTENTIONALLY LEFT BLANK

PREFACE

This paper was presented at the International Congress on Computational and Applied Mathematics, July 25–30, 1994, which was held at the Katholieke Universiteit, Leuven, Belgium. It will appear with minor revisions in 1996 in the Journal of Computational and Applied Mathematics, published by North-Holland.

THIS PAGE INTENTIONALLY LEFT BLANK

SOFTWARE NEEDS IN SPECIAL FUNCTIONS

DANIEL W. LOZIER

ABSTRACT. Currently available software for special functions exhibits gaps and defects in comparison to the needs of modern high-performance scientific computing and also, surprisingly, in comparison to what could be constructed from current algorithms. In this paper we expose some of these deficiencies and identify the related need for user-oriented testing software.

1. INTRODUCTION

A recent article by Lozier and Olver [21] provides a survey of algorithms and software for the numerical evaluation of special functions. Its emphasis is on the generation of function values although selected resources for zeros and integrals are included also. Journals, books, conference proceedings, and software documents were examined and a bibliography of nearly 500 references was constructed. Based on this investigation, the functions were classified and cross-referenced to bibliographic entries and to specific software libraries and systems[1].

The bibliography was prepared using the authors' professional experience supplemented by assistance from interested individuals. Twelve journals were searched systematically, and the review journals Mathematical Reviews and Zentralblatt für Mathematik were searched under Mathematics Subject Classification 65D20 (computation of special functions and construction of tables). The period covered by the bibliography is 1968–1993.

The survey disclaims any recommendation of algorithms or software. Its purpose is to *identify*, not to evaluate. The important topic of evaluating, or testing, numerical software for special functions is addressed in a few of the references in the bibliography.

The first purpose of this paper is to scrutinize [21] and identify those functions for which software is lacking, particularly in cases where algorithms have been described in journal articles. The need to fill these gaps will be supported, in part, by reviewing requests for software that have appeared on various electronic bulletin boards or were received directly by the author of this paper. We mention here, as an indication of current interest in numerical evaluation, that over 200 requests for a preprint of [21] were received within two weeks of an announcement of its

Key words and phrases. Special functions–computing, special functions–software, special functions–testing.

[1] Certain commercial software products are identified in this paper. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products are among the best available for the purposes they serve.

availability in the electronic NA Digest. Additional requests were generated by a later announcement in the newsletter of the SIAM Activity Group on Orthogonal Polynomials and Special Functions.

This paper's second purpose is to venture beyond the mere identification of resources into the more difficult terrain of software evaluation. A few examples of defects in function software will be presented, and some general observations will be made on the kind of developments we think are needed in the area of testing.

2. Libraries and Interactive Systems

Software is such a broad term, with so many meanings, that it is necessary to define the kind of software that is of concern in this paper. We begin by offering the following *elementary classification of scientific computing*:

- E1. Numerical Computing
  - E1.1. Fixed-Precision Floating-Point Computing
  - E1.2. Variable-Precision Floating-Point Computing
  - E1.3. Non-Floating-Point Computing
- E2. Non-Numerical Computing
  - E2.1. Symbolic Computing
  - E2.2. Graphical Computing

These categories are not mutually exclusive. Obviously, graphical computing requires numerical computing. Scaling, shading and rotating, for example, involve arithmetic operations and elementary functions, and coordinate transformations may involve special functions. Similarly, variable-precision floating-point computing requires a considerable amount of non-numerical computing. Nevertheless, this classification serves as a useful guide.

Special functions pervade all categories of this classification. Fixed-precision floating-point computing encompasses the historical development of computers and compilers for the purpose of numerically simulating the solutions of problems in engineering and science using mathematical and statistical models. Often, in the pre-computer era, these problems were considered solved when expressions in terms of special functions were obtained, for then one could, in principle, generate numerical results from tabulated values of the functions. Further, the mathematical properties of the special functions, particularly their asymptotic properties, contributed to a qualitative understanding of the solutions. Because of their prominence in applied mathematics, it is not surprising that subroutines for special functions were among the earliest examples of numerical software.

Many of the special functions arose from certain integrals and differential equations that appeared in the more tractable mathematical models. With the advent of computers, brute-force methods replaced continuous models with huge discrete analogs and permitted the numerical solution of more general problems. Nevertheless, special functions retain their importance in mathematical modeling. Their value as an aid to qualitative understanding is well recognized and accepted. They lead in some cases to more economical forms of solution, as measured by operation counts; this is particularly valuable in so-called supercomputing applications. An example is the use of the spectral method to solve partial differential equations in terms of spherical harmonics in weather and climate models. Special functions are used also in validating brute-force methods through the consideration of specialized test cases.

In statistics, cumulative distribution functions (integrals of density functions) and their inverses are special functions that form the basis of successful statistical analysis.

We conclude from the foregoing that the place of special functions in fixed-precision floating-point computing is firmly established in general scientific computation. Variable-precision floating-point and non-floating-point computing are not so clearly associated with scientific computing except in their connection to symbolic computing, that is, using the computer to do mathematics by manipulating symbols. This field of application of computers traces back almost as far as numerical computing but, historically, the approach to using the computer is different. Numerical computing, characterized by long but routine sequences of operations, proceeds very well without monitoring. Symbolic computing is more exploratory in nature, with responses from one calculation requiring human thought before the next calculation is initiated. We say symbolic computing is *interactive* whereas numerical computing is (relatively) non-interactive. A modern trend is for all computing to be more interactive, except possibly for very long numerical simulations done by supercomputers. Even in supercomputing, interactivity plays a role in the interpretation of results by graphical computing, so-called *visualization*, and in the derivation of complicated formulas by symbolic computing for use in mathematical models.

Decimal (or binary) approximations are avoided as much as possible in symbolic computing because the intention is to produce exact results. Integers and rational numbers, introduced as exact quantities and combined by exact arithmetic operations, avoid approximation and so they are admitted. Thus exact rational arithmetic, with its attendant need for variable storage control, supports a form of non-floating-point computing found in all symbolic computing systems. But decimal approximations cannot always be avoided. Sometimes a formula needs to be evaluated numerically, for example to produce a graph. Therefore, most symbolic computing systems provide, as an option that can be exercised by explicit commands, a means to evaluate formulas in variable-precision floating-point arithmetic. The ability to compute in higher precision is important because formulas generated by symbolic computing can be, and often are, numerically ill-conditioned. However, the numerical evaluation of special functions is sometimes limited to the fixed precision of the hardware arithmetic because of the difficulty of devising suitable variable-precision algorithms.

In this paper we are concerned exclusively with fixed-precision and variable-precision floating-point evaluation of special functions, and we utilize the following *software classification* [21]:

S1. Software Packages
S2. Intermediate, or Specialized, Libraries
S3. Comprehensive Libraries
S4. Interactive systems

These categories are to be regarded as progressively increasing in scope and organization.

A software package is an algorithm, or collection of algorithms, that has been implemented in a specific programming language and published in a research or technical article [2]. Its purpose is to make new algorithms available to programmers in an immediately usable form. Three important series of software packages are the *ACM Algorithms* in ACM Transactions on Mathematical Software [17], the *AS Algorithms* in Applied Statistics [28], and the *CPC Programs* in Computer Physics Communications [14, 15]. All contributions to these series are refereed before acceptance.

Intermediate and comprehensive libraries consist of software packages that have been collected, developed, organized and unified to meet the practical needs of programmers. Intermediate libraries are limited to a subset of numerical mathematics. Three examples that specialize in mathematical functions are the libraries of Baker [5], Moshier [25] and United Laboratories, Inc. [31]. Comprehensive libraries strive for complete coverage of numerical mathematics, with attention paid to uniformity of documentation, style of usage, and handling of errors. Among the many examples are CERN [8], IMSL [2], NAG [16], NSWC [24], Numerical Recipes [27], NUMPAC [3], Scientific Desk [4], and SLATEC [7].

Software packages and libraries are usually written in a standard programming language such as Fortran or C. They are used in the traditional compile-link-execute cycle of fixed-precision floating-point programming. Interactive systems break this cycle by providing a comprehensive set of commands, or in some cases items on menus, that produce an immediate response when entered at the keyboard or selected by the mouse. These systems are extensible in that user-written commands can be added, much like user-written subroutines are added to a library. However, straightforward extension of an interactive system requires programming in the specialized language of the system. Some systems provide a way of incorporating software written in a standard programming language but the process tends to be cumbersome. Three examples of interactive systems for symbolic computing with integrated support for graphics and floating-point computing are Macsyma [30], Maple [9] and Mathematica [32]. Three examples of interactive systems for fixed-precision floating-point computing with integrated support for graphics are HiQ [6], Mathcad [22] and Matlab [23]. HiQ and Mathcad, in particular, make extensive use of menus.

Software packages, intermediate and comprehensive libraries, and interactive systems serve different purposes. For special functions in a supercomputing application, the manufacturer's optimized library would be the preferred choice except that these libraries typically include little beyond algorithms for linear algebra and Fourier transforms. Accordingly, comprehensive libraries, augmented by software packages and intermediate libraries where necessary and when available, are the norm. Interactive systems are not much used in the number-crunching stage of supercomputing, because of the emphasis on very high execution rates, but symbolic and graphical systems are important during the algorithm development and

[2] It must be noted here that our usage of the term *software package* differs from common usage. More usually, it means a comprehensive, integrated, and usually commercially supplied, software product. For the latter we prefer to distinguish between *libraries* and *interactive systems*.

[3] Information can be obtained from Ichizo Ninomiya, Chubu University, Kasugai, Aichi, 487 Japan, or Yasuyo Hatano, Chukyo University, Yagoto, Nagoya, 466 Japan.

[4] Information can be obtained from C. Abaci, Inc., P. O. Box 2626, Raleigh, NC 27602.

visualization stages.

Similar remarks apply to numerical computing in general when a standard programming language is being used. However, interactive systems are becoming increasingly popular for small to medium-scale computations because they provide an integrated computing environment that can substantially ease the programming burden.

3. Current Software

To assess the current state of affairs with respect to the *availability* of fixed-precision and variable-precision floating-point software for special functions, the following *method of scoring* was applied. For a given function and library or system, the score is *zero* if the function is not present, *one* if the function is present but only for real variables, *two* if the function is present for complex variables but without separate provision for real variables, or *three* if there is separate provision for real and complex variables. The distinction between scores of 2 and 3 is made because real computation is less demanding, in general, than complex computation, and the resulting efficiency can be of importance in supercomputing applications. On the other hand, a score of 2 is quite satisfactory if high efficiency is not critical, a situation that is typical when an interactive system is being used in numerical applications and even in many standard programming applications with floating-point libraries.

This method of scoring takes into account only the presence or absence of a function. It is also important to know if technical support is available to answer questions and analyze failures. In general, this kind of support is very good for comprehensive libraries and interactive systems, particularly when they are commercially supplied. For other kinds of software, the original authors often are willing to assist.

Scores for 44 functions and 15 libraries and systems were determined with the aid of the survey paper [21], augmented by reference to software manuals and direct experimentation with the software where necessary. Figure 1 is a graphical depiction of the resulting matrix of scores where white corresponds to a score of 0, light gray to 1, dark gray to 2, and black to 3. A software package, or at least an algorithm, is listed in the survey paper for all the functions. The functions and software that correspond to the matrix rows and columns are identified in Tables 1 and 2, respectively. The rows and columns are arranged by their decreasing cumulative scores.

The figure and tables summarize and extend the survey paper. They can be used for several purposes. One immediately obvious use is in the software consultant's role in providing information to programmers upon request. Another use is in identifying gaps in the coverage of special functions. Among the comprehensive libraries, so important in supercomputing and general scientific computing, only 7 functions are present in every one: Bessel functions of real order, the error function and Dawson's integral, the exponential integrals, the complete and incomplete gamma functions, and the incomplete elliptic integrals. Eight functions are present in none: generalized hypergeometric and zeta functions, incomplete Bessel functions, integrals of Anger-Weber functions, Landau density and distribution functions, polylogarithms, spheroidal wave functions, and Weber parabolic

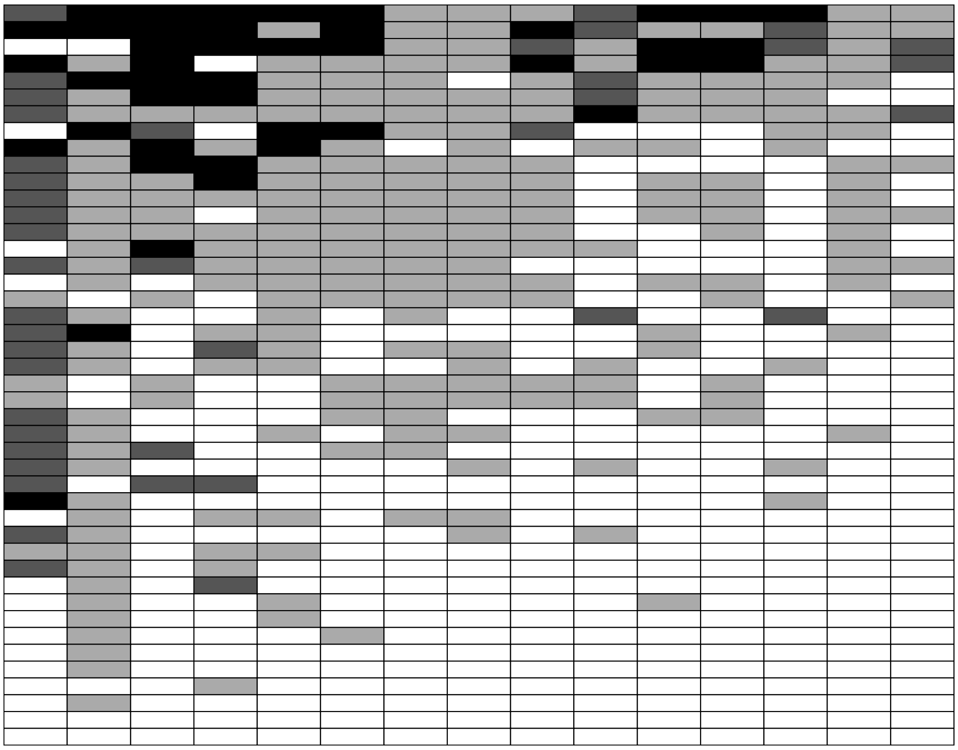

FIGURE 1. Matrix of scores for 44 functions and 15 libraries and interactive systems. See the text for the definition of *score*. A score of 3 is indicated by black, 2 by dark gray, 1 by light gray, and 0 by white. The functions are identified in Table 1. The libraries and systems are identified Table 2.

	Function	Cum. score
1	Gamma function	33
2	Error function	29
3	Bessel functions of real order	28
4	Airy functions	25
5	Exponential integrals	21
6	Psi and polygamma functions	19
7	Incomplete gamma function, generalized exponential integrals	19
8	Bessel functions of integer or half-integer order	17
9	Dilogarithm	16
10	Jacobian elliptic functions	16
11	Incomplete elliptic integrals	15
12	Dawson's integral	13
13	Incomplete beta function	13
14	Sine, cosine, hyperbolic sine and hyperbolic cosine integrals	12
15	Fresnel integrals	12
16	Complete elliptic integrals	12
17	Bessel functions of orders 0 and 1	10
18	Inverse error function	9
19	Classical orthogonal polynomials	9
20	Legendre and associated Legendre functions	9
21	Confluent hypergeometric functions	9
22	Zeta function	8
23	Inverse incomplete gamma function	8
24	Inverse incomplete beta function	8
25	Logarithmic integral	7
26	Hypergeometric functions	7
27	Weierstrass' elliptic functions	7
28	Generalized zeta function	6
29	Bessel functions of complex order	6
30	Polylogarithms	5
31	Struve functions or integrals of Struve functions	5
32	Generalized hypergeometric functions	5
33	Zeros of Bessel functions	4
34	Fermi-Dirac, Bose-Einstein and Debye integrals	4
35	Coulomb wave functions	3
36	Integrals of Bessel functions	3
37	Integrals of the error function	2
38	Mathieu functions	2
39	Weber parabolic cylinder functions	1
40	Integrals of Anger-Weber functions	1
41	Generalized polylogarithms	1
42	Spheroidal wave functions	1
43	Landau density and distribution functions	0
44	Incomplete Bessel functions	0

TABLE 1. Functions and their cumulative scores over 15 libraries and systems. See the text for the definition of *score*. See Figure 1 for individual scores. See also Table 2.

	Library or interactive system	Cum. score
1	Mathematica [S]	58
2	C Mathematical Function Handbook [IL]	46
3	Naval Surface Warfare Center Library [Fortran CL]	43
4	CERN Library [Fortran CL]	40
5	Nagoya University Mathematical Package [Fortran CL]	35
6	IMSL Library [Fortran CL]	31
7	Mathematical Function Library for Microsoft Fortran or C [IL]	25
8	Methods and C Programs for Mathematical Functions [IL]	25
9	NAG Library [Fortran CL]	24
10	Maple [S]	22
11	SLATEC Library [Fortran CL]	22
12	Scientific Desk [Fortran CL]	22
13	Macsyma [S]	18
14	Numerical Recipes [Basic, C, Fortran or Pascal CL]	17
15	Matlab [S]	12

TABLE 2. Intermediate libraries [IL], comprehensive libraries [CL], and interactive systems [S] and their cumulative scores over 44 functions. See the text for the definition of *score*. See Figure 1 for individual scores. See also Table 1.

cylinder functions. Among the 29 functions that are present in at least one but not all comprehensive libraries, we find Airy functions, Bessel functions of complex order, the dilogarithm, elliptic functions, Legendre and associated Legendre functions, inverse incomplete gamma and beta functions, the psi and polygamma functions, and the zeta function.

A third use is in identifying possibilities for software testing, a topic that will be treated in more detail in the next section of this paper. The comprehensive libraries support fixed-precision floating-point computing. Variable-precision floating-point computing is supported by Macsyma, Maple and Mathematica, and also by Matlab (which markets an add-on symbolic computing capability using Maple). Therefore, a library function could be tested, in principle, by comparison against the same function in one of the interactive systems, provided its system score is sufficiently high. From Figure 1, such a procedure is potentially feasible for 30 of the 36 functions that are included in one or more comprehensive libraries. The excluded functions are Struve functions or integrals of them, Coulomb wave functions, integrals of Bessel functions, integrals of the error function, Mathieu functions, and generalized polylogarithms.

The need to fill some of the gaps in the coverage of special functions is evidenced by recent inquiries. The NA Digest has been serving numerical analysts since 1987 with a moderated (edited) newsletter that is distributed by electronic mail. It is maintained by the Oak Ridge National Laboratory and it has a readership of approximately 4000. Sixteen inquiries have appeared: 3 for Legendre and associated Legendre functions; 2 each for Gauss hypergeometric, Mathieu and spheroidal wave functions; one each for Bessel functions of pure imaginary order, spherical Hankel functions, Kummer's or Whittaker's confluent hypergeometric functions, complex elliptic integrals, partial derivatives of the incomplete beta function, the inverse of the complementary error function, and standard probability functions accurate to full double precision. The newsgroup `sci.math.num-analysis` is an unmoderated electronic bulletin board with thousands of postings each year. Only a small number of these from the spring of 1994 have been reviewed. Inquiries about the following functions have been observed: the incomplete gamma function of complex argument; the integral of the incomplete gamma function; the digamma, Fermi-Dirac, Hankel, Jacobi and Weierstrass elliptic, parabolic cylinder, and zeta functions. Recent inquiries have been received directly by the author of this paper for the Gauss hypergeometric function, Legendre and associated Legendre functions, and spheroidal wave functions. In view of the number of gaps and the effort that would be required to fill them, those who wish to provide new software should be guided by the expressed needs of the scientific community.

4. Testing, Validation and Characterization

The previous section identified gaps in the coverage of special functions in current libraries and systems. It did not raise the question of the quality of the software. This topic has many ramifications but we are interested here only in *numerical accuracy.* As an example of the need for testing, we cite [26] in which an intermediate library with extensive coverage of special functions was reviewed. It was found that a subroutine for evaluating the Bessel function $J_\nu(x)$, x and ν real, returned highly inaccurate or even totally incorrect values for certain arguments and orders. For example, the sign and all the digits of the computed value of $J_\nu(x)$ were wrong when $x \approx 9\pi$ and $\nu = 2\frac{1}{2}(2)10\frac{1}{2}$. The reason appeared to be, at least in part, that Miller's backward recurrence algorithm was applied with normalization of the trial values by a computed value of $J_{1/2}(9\pi)$. Since $J_{1/2}(9\pi) = 0$, this cannot succeed. An alternative normalization based on

$$(\frac{x}{2})^\nu = \sum_{k=0}^{\infty} \frac{(\nu+2k)\Gamma(\nu+k)}{k!} J_{\nu+2k}(x) \quad , \quad \nu \neq 0, -1, -2, \ldots .$$

[1, eq. (9.1.87)] would have avoided the failure.

There are two principal approaches to testing. The first is *comparison against a standard.* More than 20 years ago, this was described and applied to elementary functions in [18, 19, 29]. The second approach is *verification of functional identities.* This has been described and applied to elementary and special functions in a long series of papers by W. J. Cody and co-workers, of which [11, 12, 13] are recent examples. Comparison testing is conceptually simple but requires computing in higher precision. Verification testing is performed entirely in one precision but requires great care in choosing an appropriate identity and in programming its verification. This complication is due to the necessity for separating the error that arises in the evaluation of the identity from the error in the numerical evaluation of the function itself.

Both approaches require a method of selecting test arguments and a method of measuring the error. Typically, test arguments are generated on a uniform grid or randomly except near special features of the function, algorithm, or computer arithmetic. These features include zeros, poles, and special values of the function, cross-over boundaries between approximations that are used in the algorithm, underflow and overflow thresholds, and special bit patterns. Often these non-random test arguments are the most instructive, as in the example of the Bessel function described above, but they also require careful analysis of the algorithm to identify its weaknesses. Unfortunately, black-box testing alone can never be used to prove correctness. Verification testing imposes an additional difficulty: the function must usually be evaluated at more than one point. This leads to the process known as "purification" in which the multiple arguments are carefully adjusted after initial generation so as to minimize the error due to evaluation of the identity; see, for example, Cody and Stoltz [13].

Relative error is the usual error measure but it suffers from two deficiencies: it is not a genuine distance function, and it is totally inappropriate in the vicinity of a zero. An alternative measure that overcomes the first deficiency, and that closely approximates relative error, is *relative precision* [10]

$$\mathrm{rp}(x,\overline{x}) = |\ln x - \ln \overline{x}| \quad , \quad x,\overline{x} > 0.$$

Next, in the vicinity of a zero, relative error is often replaced by absolute error. However, this is not entirely satisfactory because the location of the transition is arbitrary. A uniform error measure that avoids this difficulty is the distance function

$$d(x,\overline{x}) = |\psi(x) - \psi(\overline{x})| \quad , \quad x,\overline{x} \geq 0$$

where

$$\psi(x) = \begin{cases} x & \text{if } 0 \leq x \leq 1, \\ 1 + \ln x & \text{if } x \geq 1. \end{cases}$$

At their present stage of development, neither approach to accuracy testing is entirely satisfactory. Indeed, it is an accepted belief that good software must be developed in conjunction with test programs, and that the test programs should be distributed with the software. Test programs can be found that use either or both approaches. Typically, the tests are run when the software is installed. This process, often known as *validation*, is necessarily cursory in that only a tiny fraction of the whole set of possible inputs is tested. Validation cannot guarantee that the software will be accurate enough for a specific application. Therefore, a need exists for improved software that can be used to characterize the numerical accuracy of special functions in any subset of the input domain, to any degree of detail. We will use the term *characterization* for the process of determining the detailed behavior of the error in software for numerically evaluating a special function.

Testing that goes beyond validation to characterization should be oriented toward users and software analysts who are independent of the developers. This makes the conceptual simplicity of the comparison method particularly attractive. Fortunately, the computational obstacles that were burdensome twenty years ago can be largely overcome today. Previously, the limited capabilities of programming languages and the high cost of computing on main-frame equipment were serious issues. Now modern programming languages such as Fortran 90 and C^{++} are adequate to support higher-precision operations in a convenient, user-friendly manner; see, for example, [4]. Alternatively, following the suggestion made in the previous section, an interactive system could be used to provide higher precision for testing functions in fixed-precision libraries. And networks of workstations provide an abundance of numerical computing power that could be harnessed using modern programming and communications technology. It is possible today to consider establishing a software testing service center that would offer customized characterization of special functions on request. Ideally, such a service would be offered on the Internet so that individuals could formulate and carry out the desired tests. The remainder of this section is devoted to an example of how such a request could be answered with the aid of a graphical presentation of the results.

Let us take as our example two software packages for the Airy function $\mathrm{Ai}(z)$ for complex z. Package A is that of D. E. Amos [3]. This has been incorporated into numerous libraries and systems because of its comprehensive, efficient and accurate coverage of Bessel and Hankel functions. $\mathrm{Ai}(z)$ is provided through its representation in terms of the modified Bessel function $K_{1/3}(\zeta)$ where $\zeta = (2/3)z^{3/2}$. Package B, unpublished as yet, uses the algorithm described in [20]. This algorithm computes $\mathrm{Ai}(z)$ directly from its asymptotic expansion and its defining differential equation, which is integrated numerically. Both packages provide an option to evaluate the scaled function $\overline{\mathrm{Ai}}(z) = e^{\zeta}\,\mathrm{Ai}(z)$. We wish to characterize the error in $\overline{\mathrm{Ai}}(z)$ in the zero-free sector $|z| \leq 25$, $|\arg z| \leq 2\pi/3$.

Figures 2 and 3 show the results of this characterization of Packages A and B, respectively. The figures are surface plots of the error measure

$$e(z) = \mathrm{rp}(\overline{\mathrm{Ai}}_1(z), \overline{\mathrm{Ai}}_2(z))$$

where the subscript 1 indicates the single-precision approximation computed by Package A or B and 2 indicates the double-precision approximation computed by Package A. For small errors, $e(z)$ is almost the same as the conventional relative error.

The maximum of $e(z)$ is 1.7×10^{-5} in Figure 2 and 0.28×10^{-5} in Figure 3. For comparison, the maximum relative error due to a difference of one bit at the end of the 24-bit floating-point mantissa is 0.012×10^{-5}. Accordingly, the maximum error in this characterization of Package A affects the last 8 bits of the mantissa, compared to the last 5 bits for Package B.

Often the maximum and root-mean-square errors are the only statistics given in tests; see, for example, [11]. But Figures 2 and 3 show how much information can be lost by attempting to characterize a function with only one or two numbers. A user might feel more confidence in Package B because the error appears to be more regular and predictable. A software analyst might be interested in explaining a failure in one of the packages, or in ruling out a package as the cause of a failure in a larger application, or in attempting to improve the performance of the package. In all of these circumstances, detailed graphical output is illuminating and the ability to craft detailed tests is valuable in developing an adequate characterization of a function.

5. Summary and Conclusions

In section 2, we presented a simple classification of scientific computing in which we distinguished numerical computing from symbolic and graphical computing, and we described some of the ways special functions interact with the categories of the classification. Then, following the classification of software given in the survey

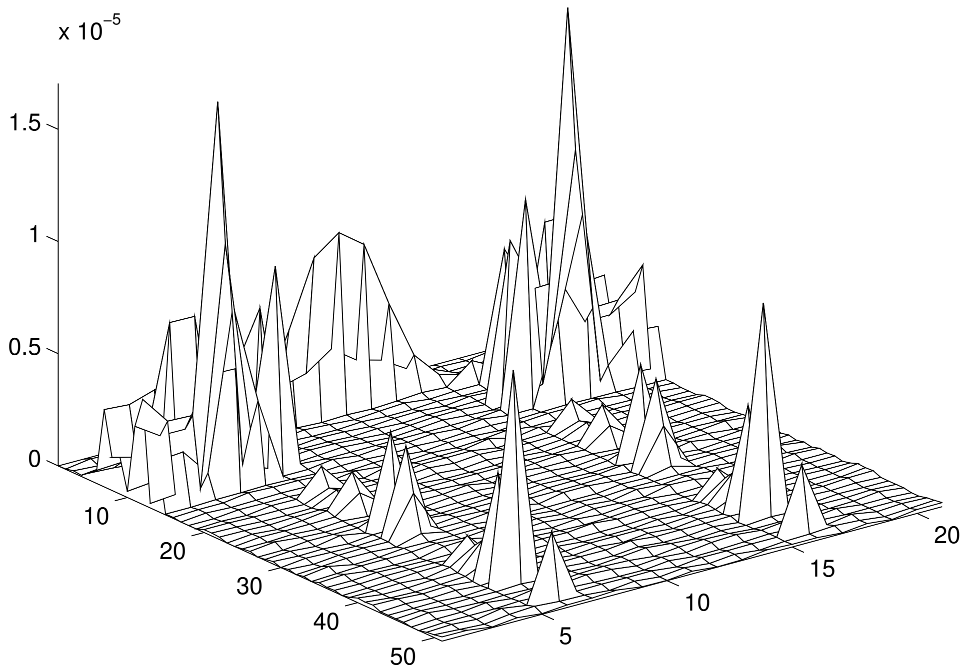

FIGURE 2. Error in D. E. Amos's software for the Airy function $e^{\zeta}\,\mathrm{Ai}(z)$, z complex, $\zeta = (2/3)z^{3/2}$, on the 51×21 grid with $|z_{ij}| = (i-1)/2$, $i = 1, 2, \ldots, 51$ and $\arg z_{ij} = -2\pi/3 + 2\pi(j-1)/30$, $j = 1, 2, \ldots, 21$. The maximum error on the grid is 1.7×10^{-5}.

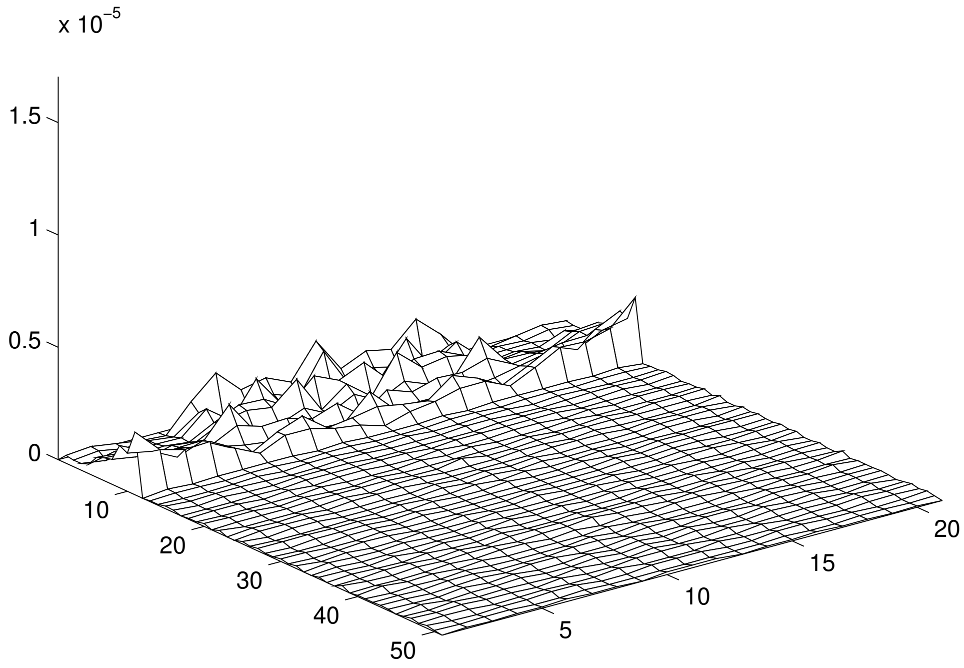

FIGURE 3. Error in a new algorithm for the Airy function $e^{\zeta}\,\mathrm{Ai}(z)$, z complex, $\zeta = (2/3)z^{3/2}$, on the 51×21 grid with $|z_{ij}| = (i-1)/2$, $i = 1, 2, \ldots, 51$ and $\arg z_{ij} = -2\pi/3 + 2\pi(j-1)/30$, $j = 1, 2, \ldots, 21$. The maximum error on the grid is 0.28×10^{-5}.

paper [21], we defined and gave examples of software packages, intermediate and comprehensive libraries, and interactive systems.

In section 3, we devised a method of scoring the coverage of a particular function in a library or interactive system, and we applied it to 44 functions and 15 libraries and systems. The resulting matrix of scores was depicted graphically in Figure 1 with accompanying detail presented in Tables 1 and 2. Then we discussed ways of discerning gaps in the coverage and how to evaluate the potential for testing library software by comparison against system functions. This approach is suggested by the fact that many systems support variable-precision floating-point computing while most libraries are limited to fixed precision. At the end of section 3, we reviewed inquiries about special functions that have surfaced in the electronic media and elsewhere, thereby providing evidence of need to fill the gaps.

We addressed the question of evaluating the quality (numerical accuracy) of function software in section 4. We distinguished between comparison testing and verification of functional identities as the two chief testing strategies, and we introduced the term *characterization* for the process of determining the detailed behavior of the error. Then we proposed the establishment of a software testing service center that would offer powerful testing software for use by anyone on the Internet. Finally, as an example of characterization, we presented surface plots in Figures 2 and 3 of the relative error in two different packages for the complex Airy function.

Our conclusions are as follows. First, given that over 50% of the scores in Figure 1 are 0, *the computation of special functions is not a mature field* when it comes to the provision of software in libraries and interactive systems. Second, since nearly 75% of the non-zero scores are 1, *the computation of complex functions is an area of particular need.* Third, since an algorithm or software package has been published for all the functions in Table 1, *the foundations exist for improving the coverage of special functions.* Fourth, in view of the characterization of the Airy functions in Figures 2 and 3, *describing the accuracy of functions by giving one or two simple statistics is not adequate.* And fifth, in view of the current advanced state of network communications and abundance of computational power, *the development of user-oriented software for characterizing functions, over the Internet if possible, should be considered.*

References

1. M. Abramowitz and I. A. Stegun (eds.), *Handbook of mathematical functions with formulas, graphs and mathematical tables*, National Bureau of Standards Applied Mathematics Series, vol. 55, U. S. Government Printing Office, Washington, D. C., 1964.
2. T. J. Aird, *The IMSL library*, Sources and Development of Mathematical Software (W. R. Cowell, ed.), Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984, pp. 264–301.
3. D. E. Amos, *Algorithm 644. A portable package for Bessel functions of a complex argument and nonnegative order*, ACM Trans. Math. Software **12** (1986), 265–273, for remark see same journal v. 16 (1990), p. 404.
4. D. H. Bailey, *Algorithm 719. Multiprecision translation and execution of Fortran programs*, ACM Trans. Math. Software **19** (1993), 288–319.
5. L. Baker, *C mathematical function handbook*, McGraw-Hill, Inc., New York, 1992, includes diskette.
6. *HiQ reference manual, version 2.0*, Bimillennium Corporation, 16795 Lark Avenue, Suite 200, Los Gatos, California 95030, 1993.

7. B. L. Buzbee, *The SLATEC common mathematical library*, Sources and Development of Mathematical Software (W. R. Cowell, ed.), Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984, pp. 302–320.
8. *CERNLIB short writeups*, CERN Program Library Office, CERN–CN Division, CH–1211 Geneva 23, Switzerland, 1993, electronic mail address is cernlib@cernvm.cern.ch.
9. B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monaghan, and S. M. Watt, *Maple V library reference manual*, Springer-Verlag, New York, 1991.
10. C. W. Clenshaw and F. W. J. Olver, *Beyond floating point*, J. Assoc. Comput. Mach. **31** (1984), 319–328.
11. W. J. Cody, *Algorithm 714. CELEFUNT: A portable test package for complex elementary functions*, ACM Trans. Math. Software **19** (1993), 1–21.
12. ———, *Algorithm 715. SPECFUN: A portable Fortran package of special function routines and test drivers*, ACM Trans. Math. Software **19** (1993), 22–32.
13. W. J. Cody and L. Stoltz, *The use of Taylor series to test accuracy of function programs*, ACM Trans. Math. Software **17** (1991), 55–63.
14. *Program master index volumes 1–40, July 1969—June 1986*, Comput. Phys. Comm. (1987), 1–75.
15. *Master index volumes 41–50, July 1986—July 1988*, Comput. Phys. Comm. (1990), 17–30.
16. B. Ford and J. C. T. Pool, *The evolving NAG library service*, Sources and Development of Mathematical Software (W. R. Cowell, ed.), Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984, pp. 375–397.
17. F. T. Krogh, *ACM algorithms policy*, ACM Trans. Math. Software **17** (1991), 427–430.
18. D. W. Lozier, L. C. Maximon, and W. L. Sadowski, *A bit comparison program for algorithm testing*, Comput. J. **16** (1973), 111–117.
19. ———, *Performance testing of a Fortran library of mathematical function routines—A case study in the application of testing techniques*, J. Res. Nat. Bur. Standards **77B** (1973), 101–110.
20. D. W. Lozier and F. W. J. Olver, *Airy and Bessel functions by parallel integration of ODEs*, Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, vol. 2 (R. F. Sincovec, D. E. Keyes, M. R. Leuze, L. R. Petzold, and D. A. Reed, eds.), Society for Industrial and Applied Mathematics, Philadelphia, 1993, pp. 531–538.
21. ———, *Numerical evaluation of special functions*, Mathematics of Computation 1943–1993: A Half-Century of Computational Mathematics (W. Gautschi, ed.), Proceedings of Symposia in Applied Mathematics, vol. 48, American Mathematical Society, Providence, Rhode Island 02940, 1994, pp. 79–125.
22. *Mathcad 4.0 user's guide*, MathSoft Inc., 201 Broadway, Cambridge, Massachusetts 02139, March 1993.
23. *MATLAB high performance numeric computation and visualization software reference guide*, The MathWorks, Inc., Cochituate Place, 24 Prime Park Way, Natick, Massachusetts 01760, August 1992, electronic mail address is info@mathworks.com.
24. A. H. Morris, Jr., *NSWC library of mathematics subroutines*, NSWCDD/TR–92/425, Naval Surface Warfare Center, Dahlgren Division, Dahlgren, Virginia, 22448, January 1993.
25. S. L. B. Moshier, *Methods and programs for mathematical functions*, Ellis Horwood Limited, Chichester, 1989, separate diskette.
26. F. W. J. Olver, *Review of United Laboratories, Inc., Mathematical Function Library for Microsoft-Fortran, Wiley, New York, 1989*, Math. Comp. **56** (1991), 879–885.
27. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes. The art of scientific computing*, second ed., Cambridge University Press, 1992, diskettes and example books available. Editions exist in Basic (1991), C (1992), Fortran (1992), Macintosh Fortran (1988) and Pascal (1989).
28. J. P. Royston, J. B. Webb, P. Griffiths, and I. D. Hill, *The construction and description of algorithms*, Appl. Statist. **36** (1987), 94–103.
29. W. L. Sadowski and D. W. Lozier, *A unified standards approach to algorithm testing*, Program Test Methods (W. C. Hetzel, ed.), Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973, pp. 277–290.
30. *MACSYMA reference manual, version 13*, Symbolics, Inc., 20 Academy St., Arlington, Massachusetts 02174-6436, November 1992.

31. *Mathematical function library for Microsoft–C*, United Laboratories, Inc., John Wiley & Sons, 1990, includes diskettes. Edition also exists in Fortran (1989).
32. S. Wolfram, *Mathematica, a system for doing mathematics by computer*, second ed., Addison-Wesley, Redwood City, California, 1991.

Applied and Computational Mathematics Division, National Institute of Standards and Technology, Gaithersburg, MD 20899
E-mail address: lozier@cam.nist.gov