Performance Evaluation of Programs for Certain Bessel Functions

W. J. CODY and L. STOLTZ
Argonne National Laboratory

This paper presents methods for performance evaluation of the K Bessel functions. Accuracy estimates are based on comparisons involving the multiplication theorem. Some ideas for checking robustness are also given. The techniques used here are easily extended to the Y Bessel functions and, with a little more effort, to the I and J functions. Details on a specific implementation for testing the K Bessel functions are included.

Categories and Subject Descriptors: G.1.0 [**Numerical Analysis**]: General—*numerical algorithms*; G.4 [**Mathematics of Computing**]: Mathematical Software—*certification and testing*

General Terms: Algorithms, Performance, Reliability

Additional Key Words and Phrases: Accuracy, Bessel functions, performance evaluation, robustness

1. INTRODUCTION

We believe that testing of a function program should be akin to a physical examination. That is, the test procedures should be thorough, including assessments of efficiency, accuracy, and robustness, and should seek to discover both strengths and weaknesses of programs under test. Test programs should be general enough to be used on any program for the given function, should be written with the same care and attention to detail that goes into other numerical software, and should be highly transportable.

Not all of these goals are yet attainable. We still know of no way to determine efficiency in a portable way; efficiency implies timing the execution of the program, hence an intimate system-dependent interaction with the operating system. There are portable ways to obtain information about accuracy and robustness, however.

In the next section we discuss an accuracy-testing methodology that relies on carefully selected identities, while Section 3 presents some ideas on assessing robustness. Although the discussions concentrate on the particular case of test programs for the real function $K_\nu(x)$, the methodology is widely applicable.

This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under contract W-31-109-Eng-38.
Authors' current address: Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439-4801.

Details of a specific implementation are given in Section 4. Section 5 demonstrates the degree to which we have achieved our goals of transportability and generality in the tests, by presenting test results obtained on several different computing systems for several different function programs. Generalizations of the test procedures for use on other Bessel functions are outlined in Section 6.

2. ACCURACY TESTING

Almost any testing procedure, even comparison against published tables, will quickly pick out function programs that are accurate to only a few digits. More sophisticated methods are necessary to demonstrate accuracy approaching machine precision. The best-known technique for sensitive accuracy testing is a controlled comparison against higher-precision computations on the host machine. This technique is not transportable, however, and cannot be conveniently used when the function being tested already uses the maximum precision on the machine.

A third possibility lies somewhere between these two extremes in sensitivity—the careful evaluation of identities. This method was exploited successfully in the ELEFUNT suite of test programs for elementary functions [5]. While accuracy statistics generated in this approach are not as discriminating as those generated in tests against higher-precision computations, they have proven useful, and they are relatively consistent across machines.

Of course, testing must be based on an identity that is not likely to be used in a program computing the function. The best identities involve no function other than the one being tested, although it is not always possible to find such an identity. We base our tests for the K Bessel functions on the multiplication theorem (Equation 9.6.51 in [1])

$$K_\nu(y) = \lambda^\nu \sum_{k=0}^{\infty} A_k(x)K_{\nu+k}(x) = \lambda^\nu\left\{K_\nu(x) + \sum_{k=1}^{\infty} A_k(x)K_{\nu+k}(x)\right\},$$

where

$$A_k(x) = \frac{(1-\lambda^2)^k(x/2)^k}{k!},$$

$y = \lambda x$, $0 \le \nu \le 1$, and $\lambda < 1$. This identity compares a single function value against a sum constructed from a sequence of function values at a slightly different argument. We note that $K_\nu(y) > K_\nu(x)$ (because $\lambda < 1$) and that $0 < K_{\nu+k}(x) < K_{\nu+k+1}(x)$ for all k. Thus all terms in the summation are positive, $K_\nu(y)$ is essentially $K_\nu(x)$ plus a small correction term, and we expect the summation to be numerically stable.

Let $S_n(x)$ denote the partial sum of the series through the $K_{\nu+n}(x)$ term, and assume that n is large enough that $S_n(x)$ theoretically represents the sum to within machine precision. Then we estimate the relative error in $K_\nu(y)$ with the expression

$$E = \frac{K_\nu(y) - \lambda^\nu S_n(x)}{K_\nu(\lambda x)}.$$

Let δ be the relative error in the computation of $K_\nu(y)$, and Δ_k that in $K_{\nu+k}(x)$. Then,

$$E = \frac{K_\nu(y)(1+\delta) - \lambda^\nu \sum_{k=0}^{n} A_k(x)K_{\nu+k}(x)(1+\Delta_k)}{K_\nu(y)(1+\delta)}.$$

Simple algebra, ignoring error terms higher than the first order, gives finally,

$$E = \delta - \frac{\lambda^\nu}{K_\nu(y)} \sum_{k=0}^{n} A_k(x)K_{\nu+k}(x)\Delta_k.$$

Because the series is convergent, the Δ_k for larger k have little effect, and the error associated with the first few terms of the series dominates.

Round-off error in the evaluation of the series is easily controlled. Let $a_k(x) = (1-\lambda^2)(x/2)/k$, and evaluate $S_n(x)$ with Horner's nested multiplication scheme:

$$S_n(x) = K_\nu(x) + a_1(x)[K_{\nu+1}(x) + a_2(x)[K_{\nu+2}(x) + \cdots + a_n(x)K_{\nu+n}(x)] \cdots].$$

This process is numerically stable for a wide range of x provided some care is exercised in selecting the control variable λ. If $1-\lambda^2 < \frac{1}{8}$, then $a_k(x) < 1/k$ for $x < 16$. Further, given a random machine argument x, it can be perturbed slightly so that x, y, and $1-\lambda^2$ are all exact machine numbers (see Section 4). Because the series converges so rapidly, only the error in $a_1(x)$ is potentially visible in the final sum, and that error has been rendered small.

This analysis suggests that a reasonably sensitive error test can be built with the multiplication theorem. In Section 5 we show how to verify this expectation experimentally.

There remains the task of generating the necessary sequence of Bessel functions. Because the recurrence

$$K_{\nu+j+1}(x) = K_{\nu+j-1}(x) + \frac{2(\nu+j)}{x} K_{\nu+j}(x)$$

is known to be numerically stable in the forward direction, the required sequence can be constructed accurately from the first two elements provided overflow does not interfere. (Prevention of overflow is also discussed in Section 4.) Testing the accuracy of a routine for either $K_0(x)$ or $K_1(x)$ thus requires that both routines be used. When testing the accuracy of a routine that generates a sequence of functions, that routine could be used to generate the entire sequence. However we believe that only the first two elements of the sequence should be obtained from that routine, and that the rest should be generated explicitly in the test program.

3. ASSESSING ROBUSTNESS

Robustness refers to the ability of a program to recover from misuse. Robust programs have built-in recovery from illegal arguments and are written to avoid intermediate underflow or overflow. While it is probably impossible to test all misuses of a Bessel function program, our test programs check the most obvious ones in a series of calls with illegal arguments, and arguments at or beyond thresholds where Bessel programs are expected to malfunction.

Tests are ordered in estimated severity of the expected error. The first set of tests uses invalid arguments, such as negative values for x and ν. The expectation is that the Bessel program will diagnose the problem and warn the user without aborting execution. The second set of tests uses extreme parameters, such as the smallest positive floating-point number and, in the case where the function program returns $e^x K_\nu(x)$, the largest floating-point number, *XMAX*. Finally, $K_\nu(x)$ is tested near its underflow threshold, $x_{\max}$, which must be determined by the test program. Asymptotically, $K_\nu(x)$ has the form

$$K_\nu(x) \sim e^{-x} \sqrt{\frac{\pi}{2x}} \left(1 + \frac{4\nu^2 - 1}{8x} + \frac{(4\nu^2 - 1)(4\nu^2 - 9)}{2!(8x)^2} + \ldots\right).$$

The underflow threshold is the solution to the equation $K_\nu(x) = XMIN$, where *XMIN* is the smallest positive floating-point number. The test program determines $x_{\max}$ by using Newton iteration on the equation

$$e^{-x} \sqrt{\frac{\pi}{2x}} \left(1 - \frac{1}{8x} + \frac{9}{128x^2}\right) = \beta^{\text{minexp}},$$

where β is the radix for the floating-point representation and *minexp* is the smallest representable power of β. This equation is obtained from the first three terms of the asymptotic form with $\nu = 0$, which, because of the monotonic behavior of $K_x(x)$ for fixed x, is the worst case.

4. IMPLEMENTATION DETAILS

The ideal test program reports only error generated by the function software under test operating on exact arguments. That error is easily contaminated by rounding error generated in the testing process, and by errors attributed to inexact arguments. Careful implementation of the test algorithms minimizes the first contamination and eliminates the source of the second completely.

The key is to generate values of λ, $1 - \lambda^2$, x, and y that are all exact machine numbers. This approach permits accurate generation of the a_k needed for the Horner scheme and guarantees that the Bessel function programs use exact arguments. Everything hinges on the choice of λ.

Recall that we want $\lambda < 1$ and $1 - \lambda^2 < \frac{1}{8}$. If $\lambda = (2^n - 1)/2^n$, then $1 - \lambda^2 = (2^{n+1} - 1)/2^{2n}$, and only n bits are needed to represent λ and $n + 1$ bits to represent $1 - \lambda^2$ exactly on a binary machine. All of the conditions on λ are satisfied for $n > 3$.

Given y, the corresponding x must be generated exactly. This is done with a process called *argument purification*, in which trial values of the desired arguments are generated and then perturbed slightly so the desired mathematical relations hold. In this instance, starting with a random value of y, the corresponding trial value of x is perturbed sufficiently to introduce n low-order zero bits (just enough to guarantee that subsequent multiplication by λ will be exact), and then y is reconstructed. Clearly, a small value of n in the definition of λ will minimize the perturbation on x. The following sequence of FORTRAN statements purifies the arguments when $n = 4$:

$$\begin{aligned} X &= Y/XLAM \\ W &= SIXTEN * X \\ T1 &= W + X \\ X &= T1 - W \\ Y &= X * XLAM \end{aligned}$$

where $XLAM = 15.0/16.0$ and $SIXTEN = 16.0$. The addition of $16x$ to x, and its subsequent subtraction off again, introduces the desired four low-order zero bits, provided the intermediate results are not retained to more than working precision. The sequence of statements given above is intended to fool optimizing compilers on machines with over-length registers into storing (and using) these intermediate results at working precision. Even so, it may be necessary to control optimization through compiler directives on some machines.

With these values for λ and x, it is now possible to generate the a_k, $k = 1, \ldots, n$, and the partial sum $S_n(x)$ accurately. Because x is exact with trailing zero bits, $x/2$ will be exact on most binary machines. Therefore, $a_k = (1 - \lambda^2)(x/2)/k$ will be correct to within one or two rounding errors. The Horner scheme evaluates the summation starting with the smallest terms. Because $a_n < x/(16n)$, rounding errors in the evaluation of a_n, $K_{\nu+n}(x)$, and their product are lost in the subsequent addition to $K_{\nu+n-1}(x)$. For x small, rounding errors in each step of the Horner scheme are lost in this way. Even for $x = 20$, only rounding errors from the final few steps will appear in the sum.

The necessary sequence of Bessel functions is generated from the recurrence relation, starting with the first two elements. The problem is to generate sufficient function values to ensure convergence of the sum and yet to avoid overflow in the recurrence. There are sufficient terms for convergence when $A_k(x)K_{\nu+k}(x) < S_1(x)\epsilon/100$, where ϵ is the smallest number such that $1 + \epsilon > 1$. Avoidance of overflow is a little trickier. Overflow is most likely to occur on machines in which the total exponent range is less than five times the length of the significand, and then when $y < 1/2$. In this situation the recurrence is scaled by ϵ at the start, and $S_n(x)$ is unscaled at the end. As an added precaution, comparison of $(2/x)(\nu + j)K_{\nu+j}(x)$ against the largest machine-representable number, $XMAX$, is used to check for overflow in the next recurrence step. To avoid overflow in the comparison itself, the test becomes $(2/x)(\nu + j) > XMAX/K_{\nu+j}(x)$. If overflow is predicted, the recurrence is abandoned and computation skips to the next argument.

The test program for $K_\nu(x)$ uses $\lambda = 15/16$, ν random in [0, 1], and x chosen randomly from three intervals: [0.0, 1.0], [1.0, 10.0], and [10.0, 20.0]. The first interval isolates the troublesome region near the singularity at the origin, and is the region where most of the cases will be rejected because of unavoidable overflow in the recurrence. The second interval is a region where the test program and the Bessel function programs are expected to perform well. The upper limit of the third interval is dictated by the earlier error analysis. Each interval is divided into 2,000 subintervals, and a test argument y is randomly selected from each, thus ensuring a uniform distribution of test arguments.

The statistics reported for each interval are the maximum relative error (MRE) and root mean square error (RMS) in $K_\nu(y)$. These statistics are expressed in terms of the estimated loss of base-β digits in the floating-point significand, a measure that is independent of the machine wordlength. For example, the reported value of MRE is

$$MRE = p - \ln(\max|E|)/\ln(\beta),$$

where p is the number of significant base-β digits in the significand.

Machine-dependent parameters, such as $XMAX$, β, and $minexp$, are determined dynamically with the subroutine MACHAR [4].

5. TEST RESULTS

Our test program is designed to report error generated in the function program being tested, but some error is inherent in the testing process. To measure this, we must *test* the testing procedure itself. This *calibration* process, or benchmark, is accomplished by running the test program on an *ideal* function program, a single-precision program that does all computations in double precision and returns the best possible single-precision result. The calibration results then approximate the error introduced by the testing program.

Table I lists results for accuracy tests run on a VAX 11/780 under 4.2 BSD UNIX[1], a SUN 3/60 running SUN UNIX 4.2, an Alliant FX/8 running Concentrix 4.0.0, and an IBM XT using the MicroSoft Fortran 4.01 compiler. The VAX tests were all run with locally developed replacements for some of the double-precision elementary functions.

Not many programs are available for $K_\nu(x)$. In addition to the calibration runs, tests were performed on programs from the SPECFUN package [3], on single-precision programs from FNLIB (obtained from NETLIB [6]), and on programs from the IMSL SFUN library [7]. The SPECFUN program, RKBESL, is a heavily revised portable version of Campbell's IBM program RBESK [2]. The revisions include the addition on nonscaled functions, more accurate approximations, parameterization of machine dependencies, and the elimination of underflow/overflow problems. The IMSL library was only available to us on the VAX. However we believe that many of the FNLIB routines are early versions of SFUN programs. Note that the VAX results for the FNLIB program for $K_\nu(x)$ are identical to those for the IMSL SFUN program. Also note the uniformity of the calibration results across machine lines, and the general agreement of the results for a particular program for different machines.

6. GENERALIZATIONS FOR OTHER BESSEL FUNCTIONS

The procedures we have described in the context of testing programs for the $K_\nu(x)$ Bessel function can be extended for use on other Bessel functions. When stand-alone programs exist for $K_0(x)$ and $K_1(x)$, the modifications are minimal. These two programs must be considered together (to obtain the necessary starting values for the recurrence), and the details of the test program must be modified

[1] UNIX is a trademark of AT&T Bell Laboratories.

Table I. Test Results for $K_\nu(x)$, $0 \le \nu \le 1$

	Test intervals					
	(0.0, 1.0)		(1.0, 10.0)		(10.0, 20.0)	
Code	MRE	RMS	MRE	RMS	MRE	RMS
VAX 11/780						
Calibration	1.86	0.00	1.56	0.00	2.05	0.23
SPECFUN (S.P.)	3.54	0.90	2.56	0.79	2.67	0.86
SPECFUN (D.P., D-Format)	4.67	1.13	2.65	0.94	3.05	1.07
FNLIB (S.P.)	6.20	2.05	5.70	2.00	3.18	1.27
SFUN (S.P.)	6.20	2.05	5.70	2.00	3.18	1.27
SFUN (D.P., D-Format)	5.13	1.88	5.30	2.41	3.85	2.08
SUN 3/60						
Calibration	1.61	0.00	1.81	0.00	1.99	0.15
SPECFUN (S.P.)	3.27	0.68	2.06	0.39	2.16	0.46
SPECFUN (D.P.)	3.93	0.84	2.04	0.32	2.11	0.45
FNLIB (S.P.)	5.36	1.70	4.89	1.43	2.63	0.90
Alliant FX/8						
Calibration	1.75	0.00	1.88	0.00	2.00	0.28
SPECFUN (S.P.)	3.28	0.87	2.67	0.86	2.58	0.90
SPECFUN (D.P.)	3.60	0.96	2.76	0.80	2.64	0.88
FNLIB (S.P.)	5.79	1.99	4.75	1.84	3.14	1.28
IBM XT (MS 4.01)						
Calibration	1.04	0.00	1.31	0.00	1.95	0.08
SPECFUN (S.P.)	3.27	0.66	1.97	0.30	2.10	0.42
SPECFUN (D.P.)	4.03	0.87	1.97	0.26	2.06	0.42
FNLIB (S.P.)	5.36	1.70	4.89	1.42	2.64	0.88

slightly when testing $K_1(x)$ to start the summation with the second term in the sequence obtained from the recurrence. Otherwise the test programs are similar to that for $K_\nu(x)$.

Extension of the procedures for use with the Y family of Bessel functions is more difficult. The main complication is that $Y_\nu(x)$ is ultimately oscillatory, and the summation often involves functions that are both positive and negative. Furthermore when λx is close to a zero of the function, the summation loses accuracy through subtraction error. All of this complicates the error analysis. Even so, with extra care in the selection of intervals and in the tests for convergence of the summation, it is possible to adapt our test procedures for the Y functions. Test programs have already been produced for the special cases $Y_0(x)$ and $Y_1(x)$, where the two programs are considered together. (This work was done with the assistance of G. Zazi, a Faculty Research Participant from Chicago State University.) We intend to write a test for general $Y_\nu(x)$.

The multiplication theorem can also be used to test programs for the I and J families of functions. The main additional complication here is that backward recurrence must be used to generate the function values for the summation, but the techniques for doing this are well known. We expect to produce a set of test programs for these functions analogous to those discussed for the K and Y functions.

REFERENCES

1. Abramowitz, M., and Stegun, I. A. *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. Campbell, J. B. A FORTRAN IV subroutine for the modified bessel functions of the third kind of real order and real argument. Report NRC/ERB-925, National Research Council, Canada, 1980.
3. Cody, W. J. SPECFUN—a portable special function package. In *New Computing Environments: Microcomputers in Large-Scale Scientific Computing.* A. Wouk, ed. SIAM, Philadelphia, 1987, pp. 1–12.
4. Cody, W. J. Algorithm 665: MACHAR: A subroutine to dynamically determine machine parameters. *ACM Trans. Math. Softw. 14,* 4 (Dec. 1988), 303–311.
5. Cody, W. J., and Waite, W. *Software Manual for the Elementary Functions.* Prentice-Hall, Englewood Cliffs, N.J., 1980.
6. Dongarra, J. J., and du Croz, J. Distribution of mathematical software via electronic mail. Tech. Rep. MCS-TM-48, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1985.
7. *SFUN/LIBRARY User's Manual.* IMSL, Inc., Houston, Tex., 1987

Received April 1988; revised July 1988; accepted September 1988