ALGORITHM 644
A Portable Package for Bessel Functions of a Complex Argument and Nonnegative Order

D. E. AMOS
Sandia National Laboratories

This algorithm is a package of subroutines for computing Bessel functions $H_\nu^{(1)}(z)$, $H_\nu^{(2)}(z)$, $I_\nu(z)$, $J_\nu(z)$, $K_\nu(z)$, $Y_\nu(z)$ and Airy functions $Ai(z)$, $Ai'(z)$, $Bi(z)$, $Bi'(z)$ for orders $\nu \geq 0$ and complex z in $-\pi < \arg z \leq \pi$. Eight callable subroutines and their double-precision counterparts are provided. Exponential scaling and sequence generation are auxiliary options.

Categories and Subject Descriptors: G.1.0 [**Numerical Analysis**]: General—*numerical algorithms*; G.1.m [**Numerical Analysis**]: Miscellaneous—*function evaluation*; G.m. [**Mathematics of Computing**]: Miscellaneous—*FORTRAN*

General Terms: Algorithms

Additional Key Words and Phrases: H, I, J, K, and Y Bessel functions, complex airy functions, complex Bessel functions, derivatives of airy functions, log gamma function

1. INTRODUCTION

This algorithm computes the Bessel functions $H_\nu^{(1)}(z)$, $H_\nu^{(2)}(z)$, $I_\nu(z)$, $J_\nu(z)$, $K_\nu(z)$, $Y_\nu(z)$ and Airy functions $Ai(z)$, $Ai'(z)$, $Bi(z)$, $Bi'(z)$ for orders $\nu \geq 0$ and complex z in $-\pi < \arg z \leq \pi$. References [2] and [3] document the mathematical details, while [4] documents the machine-oriented aspects of the package. The package consists of eight callable routines and their double-precision counterparts, along with quick check drivers that evaluate known relations for exercising the main loops in each subroutine (see Table I). The definitions and notation follow that of [1], chapter 9.

The auxiliary options include (1) exponential scaling to increase the argument range of z and (2) generation of N member sequences for orders ν, $\nu + 1$, ..., $\nu + N - 1$. Function values that would underflow are set to zero, and a flag NZ is returned to indicate the number of zeros in a sequence that were generated by

This work was performed at Sandia National Laboratories, and supported by the U.S. Department of Energy under contract number DE-AC 04-76 DP 00789.
Author's address: Numerical Mathematics Division, Sandia National Laboratories, Albuquerque, NM 87185.

Table I.

Function	Single-precision name	Double-precision name	Quick check drivers
$H_\nu^{(1)}(z)$, $H_\nu^{(2)}(z)$	CBESH	ZBESH	CQCBH, ZQCBH
$I_\nu(z)$	CBESI	ZBESI	CQCBI, ZQCBI
$J_\nu(z)$	CBESJ	ZBESJ	CQCBJ, ZQCBJ
$K_\nu(z)$	CBESK	ZBESK	CQCBK, ZQCBK
$Y_\nu(z)$	CBESY	ZBESY	CQCBY, ZQCBY
$Ai(z)$, $Ai'(z)$	CAIRY	ZAIRY	CQCAI, ZQCAI
$Bi(z)$, $Bi'(z)$	CBIRY	ZBIRY	
$\ln \Gamma(x)$, $x > 0$	GAMLN	DGAMLN	

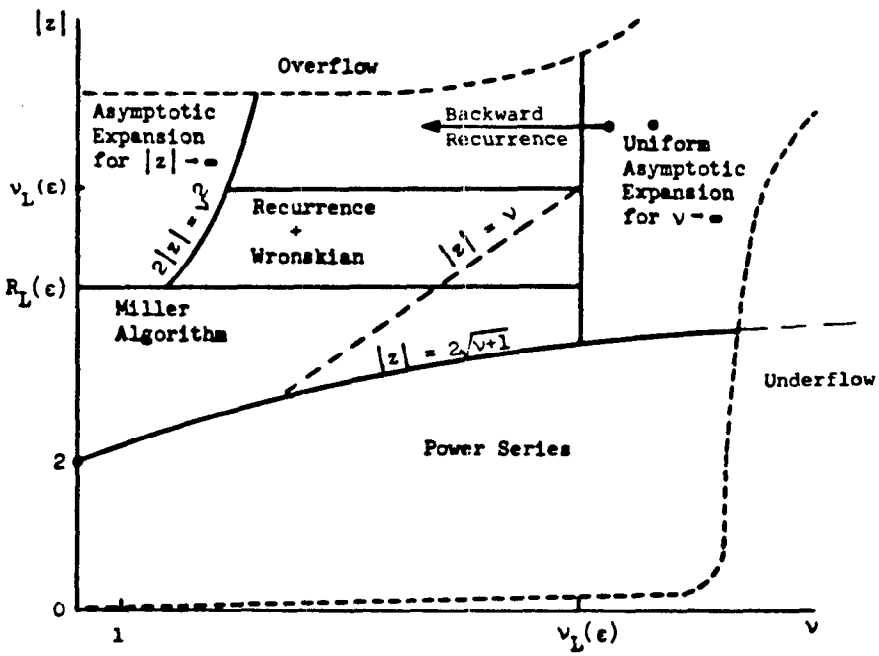

Fig. 1. Computation diagram for $I_\nu(z)$.

underflow. The prologue of the particular subroutine indicates whether these zero values are first or last in the sequence. A nonzero error flag IERR signals improper input and overflow conditions. The tops of Figures 1 and 2 show regions of exponential overflow or underflow for $I_\nu(z)$ and $K_\nu(z)$ in the right-half z plane. Since all functions, including their analytic continuations, are based on these two functions, the scaling option carries over to the remainder of the package.

The collection of software for complex functions in general is not very extensive. The *Collected Algorithms from ACM*, as well as NAG, IMSL, and SLATEC libraries [6], [8], are only minimally cognizant of complex variable computation.

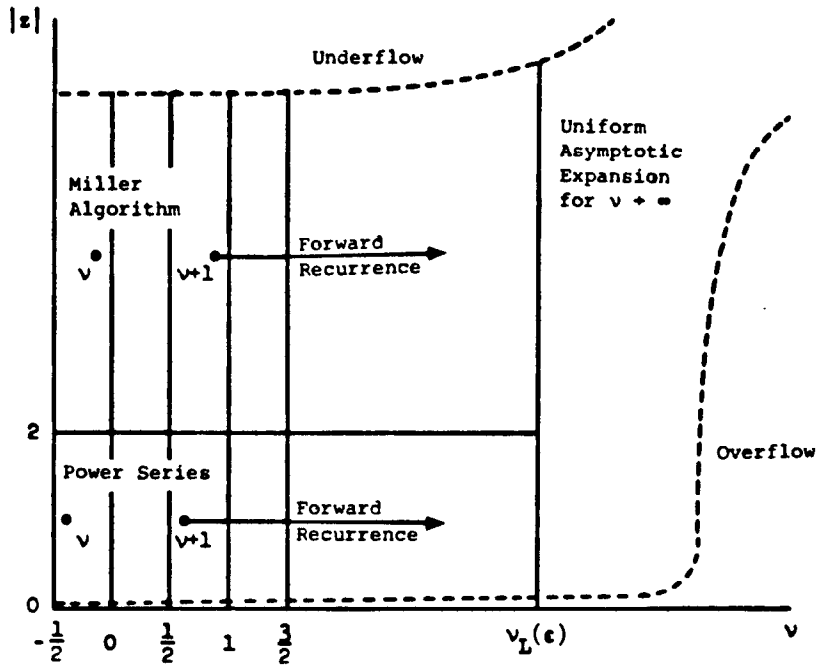

Fig. 2. Computation diagram for $K_\nu(z)$.

Quality software for complex Bessel functions is practically nonexistent, though [11] and [12] contribute substantially to the computation of $I_n(z)$, $n = 0, 1, \ldots$. Reference [5]cites a few other less accessible integer order routines.

This package fills a major part of the void in complex Bessel function software. First, all major Bessel functions are represented, including the spherical Bessel functions and Kelvin functions. Second, the package is written in a portable fashion in which all machine constants are relegated to three basic subroutines. Third, the argument ranges in $\nu \geq 0$ and $|z|$ are only restricted by the ability of the machine to represent ν, z, and the answer (i.e., the exponent range of the machine). Fourth, computational problems near underflow and overflow limits were addressed with automatic scaling to retain accurate answers [4]. Fifth, the double-precision version makes the package convenient for numerical work on short-word-length machines. Sixth, the subdivision of the $(\nu, |z|)$ plane shown in Figures 1 and 2 assures that the best computational ranges of each formula are used for speed and accuracy. And finally, extensive testing has led to the correction of coding errors as well as increased confidence in the package. The tests, most of which were motivated by the work in [2], [3], and [4], included computations to verify

(1) correct scaling near underflow and overflow limits,
(2) correct exponential scaling,
(3) correct analytic continuation,

(4) correct jump conditions across the negative real axis (i.e., subtraction of the analytic continuation formulae)

$$I_\nu(ze^{\pm i\pi}) = e^{\pm i\pi\nu}I_\nu(z)$$

$$K_\nu(ze^{\pm i\pi}) = e^{\mp i\pi\nu}K_\nu(z) \mp i\pi I_\nu(z)$$

$$J_\nu(ze^{\pm i\pi}) = e^{\pm i\pi\nu}J_\nu(z)$$

$$Y_\nu(ze^{\pm i\pi}) = e^{\mp i\pi\nu}Y_\nu(z) \pm 2i \cos \pi\nu J_\nu(z)$$

$$H_\nu^{(1)}(ze^{\pm i\pi}) = J_\nu(ze^{\pm i\pi}) + iY_\nu(ze^{\pm i\pi})$$

$$H_\nu^{(2)}(ze^{\pm i\pi}) = J_\nu(ze^{\pm i\pi}) - iY_\nu(ze^{\pm i\pi})$$

for $z = x$, $x > 0$,

(5) continuity of $\mathrm{Ai}(z)$ and $\mathrm{Ai}'(z)$ along $z = -x$ via the relations

$$\mathrm{Ai}(-x) = [(i\sqrt{x})/(\pi\sqrt{3})]K_{1/3}(\xi e^{3\pi i/2}) = [(-i\sqrt{x})/(\pi\sqrt{3})]K_{1/3}(\xi e^{-3\pi i/2})$$

$$\mathrm{Ai}'(-x) = [(x/(\pi\sqrt{3})]K_{2/3}(\xi e^{3\pi i/2}) = [(x/(\pi\sqrt{3})]K_{2/3}(\xi e^{-3\pi i/2})$$

$$\xi = \tfrac{2}{3}x^{3/2}, \qquad x > 0$$

(6) evaluation of $\mathrm{Ai}(z)$, $\mathrm{Ai}'(z)$, $\mathrm{Bi}(z)$, $\mathrm{Bi}'(z)$ for $z = x$ using local real routines on $-\infty < x < \infty$,

(7) evaluation of the Kelvin functions $\mathrm{ber}_\nu(x)$, $\mathrm{bei}_\nu(x)$, $\mathrm{ker}_\nu(x)$, $\mathrm{kei}_\nu(x)$ in

$$e^{-\nu\pi i/2}K_\nu(xe^{i\pi/4}) = \mathrm{ker}_\nu(x) + \mathrm{ikei}_\nu(x)$$

$$J_\nu(xe^{3\pi i/4}) = \mathrm{ber}_\nu(x) + \mathrm{ibei}_\nu(x)$$

using local real routines for $x > 0$,

(8) evaluation of $I_\nu(z)$, $J_\nu(z)$, $K_\nu(z)$, $Y_\nu(z)$ for $z = x$ using local real routines for $x > 0$ and selected values of $\nu \geq 0$,

(9) evaluation of auxiliary relations like the Wronskian, $H_\nu^{(1)}(x) = J_\nu(x) + iY_\nu(x)$, $x > 0$, and so forth,

(10) tables of zeros for $H_0^{(1)}(z)$, $H_1^{(1)}(z)$, $Y_n(z)$, and $n = 0$, 1, 2, 3, 4, 5, and 15 found in [7]. (Errata for this table can be found in [4].),

(11) internal consistency checks in which recurrences are started with one formula, carried across a boundary, and terminated with a check from a second formula (see Figures 1 and 2 for some formula boundaries).

Variants of items (9) and (11) are used in all of the quick check routines.

The basic idea is to compute the $I_\nu(z)$ and $K_\nu(z)$ functions in the right-half plane $\mathrm{Re}(z) \geq 0$, since good computational techniques are known for these functions. The extensions to the left-half plane and their relation to the other functions are provided by well-known formulae. Complete details are given in [2], [3], and [4] with brief descriptions and essential formulae given in each subroutine prologue. Figures 1 and 2 show the computation diagram for the right-half plane.

Since there is no standard type Double-Precision Complex, the double-precision routines (first letter of the subroutine name is Z for double-precision

complex and D for double-precision real) carry complex numbers as ordered pairs. Section 4, in addition to listing lower level routines, lists the common mathematical functions in the package that are needed to manipulate double-precision ordered pairs as complex numbers.

The package was written according to FORTRAN 66 standards and passed the PFORT analyzer. The package should compile on most FORTRAN 77 compilers.

2. PACKAGE ACCURACY

This Bessel function package has a precision P = max(unit round-off, 1.0E-18) because constants are stored to only 18 digits (UNIVAC double precision). Thus, the very best one can do in computation is to achieve precision P. However, owing to the wide argument ranges in ν and $|z|$, this precision is generally reduced. This comes about mainly through argument reduction in the elementary functions. Consequently, the best we can expect in terms of relative error is P*(10.0**S), where 10.0**S represents the increase in error owing to argument reduction. Here S = max(1, |ALOG10 (|z|) |, |ALOG10(ν)|) = max(1, |exponent of |z||, |exponent of ν|), approximately. For example, if $\nu = 10$ and $|z| = 1000$, one can expect a loss of about 3 significant digits. For the Airy functions S = max(1, $\frac{3}{2}$ |ALOG10(|z|)|), approximately.

It is in the nature of complex arithmetic that magnitudes tend to be computed with the prevailing relative precision, while phase angles can have either relative or absolute error. Absolute accuracy is most likely to occur when one component (in absolute value) is larger than the other by several orders of magnitude. If one component is (mathematically) larger than the other by a factor of 10.0**K, then one can only expect max(|ALOG10(P)| −K, 0) significant digits in the smaller component; or, stated another way, when K exceeds the exponent of P, no significant digits remain in the smaller component. However, the phase angle retains absolute accuracy because, in complex arithmetic with precision P, the smaller component will not, as a rule, decrease below P times the magnitude of the larger component. In these extreme cases, the principal phase angle is on the order of $\pm$ P or $\pm \pi/2 \mp$ P.

In testing, where results were compared, either with another computation from the package or externally from another (possibly double-precision) source, the magnitude of the relative error was always close to P*(10.0**S), except near zeros of the function being tested. In this latter case, absolute accuracy is all that can be expected.

3. MACHINE RELATED PARAMETERS AND ERROR FLAGS

Included in the package are function subroutines that define the machine environment in which the package is to operate. This helps make the package portable by relegating all machine dependent parameters to these function subroutines. These routines are called I1MACH( ), R1MACH( ), and D1MACH( ) which are enhancements of those presented in [9].

I1MACH( ) defines integer constants associated with the environment. For example, standard input, output, punch, and error message units are defined by I1MACH(I), I = 1, 4. The number of bits per storage unit (word) and number of

alphanumeric characters in a storage unit are defined in I1MACH(5) and I1MACH(6). The constants associated with integer, single precision, and double precision arithmetic are defined in I1MACH(7) through I1MACH(16). These include the base of the arithmetic B, maximum and minimum exponents, and the number of base B digits.

R1MACH(I), I = 1, 5 returns the smallest and largest (positive) floating (single-precision) numbers, the smallest relative spacing, the largest relative spacing (unit round-off), and log base 10 of the single-precision arithmetic base. Similar statements hold for D1MACH(I), I = 1, 5 for double-precision arithmetic.

To make the usage easier, these quantities are defined on comment lines for a wide variety of systems. To define one of these systems, one has only to replace C's in column 1 of these comment lines by spaces to make a particular set of FORTRAN statements active.

In the main routines, which are called by the user, package parameters TOL, ELIM, and ALIM are computed from machine constants and passed to lower level routines. These package parameters are slightly different from those that could be computed directly from I1MACH( ), R1MACH( ), and D1MACH( ) because we wish to impose further limitations. Thus,

```
TOL = AMAX1(R1MACH(4), 1.0E-18)
```

defines the package unit round-off that is limited to 1.0E-18, because constants are stored to only 18 digits (UNIVAC double precision). The statements

```
  K = MINO(IABS(I1MACH(12)), IABS(I1MACH(13)))
ELIM = 2.303EO * (FLOAT(K) * R1MACH(5) - 3.0EO)
 AA = 2.303EO * R1MACH(5) * FLOAT(I1MACH(11) - 1)
ALIM = ELIM + AMAX1(-AA, -41.45EO)
```

define the package exponential overflow or underflow limit ELIM cushioned by a factor of 10^3 to allow for some impreciseness in tests using first-term approximations. Also, one is not always sure that the ALOG (or DLOG) function would perform correctly on the largest or smallest machine numbers. Thus, ELIM = AMIN1(−ALOG(R1MACH(1)), ALOG(R1MACH(2))) may be desirable but not prudent. ALIM is a near-underflow or near-overflow quantity that triggers a nonunit scaling option described in Section 3 of [4]. More precisely, these computations express the relations

$$e^{\mathrm{ELIM}} = \mathrm{B}^{\mathrm{K}} \cdot 10^{-3}, \qquad e^{\mathrm{ALIM}} = e^{\mathrm{ELIM}} * \mathrm{TOL}$$
$$e^{-\mathrm{ELIM}} = \mathrm{B}^{-\mathrm{K}} \cdot 10^{3}, \qquad e^{-\mathrm{ALIM}} = e^{-\mathrm{ELIM}}/\mathrm{TOL}$$

where I1MACH(12) and I1MACH(13) are the minimum and maximum exponents possible for a floating number, that is, R1MACH(1) = $\mathrm{B}^{\mathrm{I1MACH(12)}-1}$, where B = I1MACH(10) = the floating point base. Notice that the computation of ELIM is accomplished without complicated function evaluations. For double-precision arithmetic, indices 11, 12, and 13 are replaced by 14, 15, and 16 respectively, and R1MACH( ) is replaced by D1MACH( ).

ELIM and ALIM are carried this way because all tests for overflow or underflow are made on the magnitude of the logarithm of a Bessel function. ± ALIM are exponent boundaries above or below which scaling is applied as described in Section 3 of [4].

Portability is also enhanced by the use of work arrays in call lists. Since a complete restoration of all variables to their former values is not always assured in successive calls, work arrays will pass those to be saved to the calling routine and restore them on the next call. This method is used in subroutine CUNIK where the coefficients of the uniform expansions for $\nu \to \infty$ are computed for either I or K and saved (the terms differ only in sign) for a subsequent call to compute the analytic continuation that needs both I and K functions in the right-half plane.

Diagnostics are returned by means of an error flag IERR. IERR should be tested after each call to ensure proper usage of a subroutine. IERR = 0 is a normal return, meaning that the computation was completed, and an answer is returned. IERR = 1 means an input error was detected, and IERR = 2 means an overflow condition was detected. When the magnitude of z or $\nu + N - 1$ is large, losses of significance by argument reduction in elementary functions occur. Consequently, if either $|z|$ or $\nu + N - 1$ exceeds U1 = SQRT(0.5/TOL), then losses exceeding half-precision are likely. In this case the computation is completed, but IERR = 3 is returned as a warning. If either $|z|$ or $\nu + N - 1$ is larger than U2 = 0.5/TOL, then all significance would be lost and U2 is an upper bound on $|z|$ and $\nu + N - 1$. In order to use the INT function, arguments must be further restricted not to exceed the largest machine integer U3 = I1MACH(9). Thus, the magnitude of z and $\nu + N - 1$ must not exceed min(U2, U3). If this restriction is violated, then IERR = 4 is returned, and no computation is attempted. On 32-bit machines, U1, U2, and U3 are approximately 2.0E + 3, 4.2E + 6, and 2.1E + 9 in single-precision arithmetic, and 1.3E + 8, 1.8E + 16, and 2.1E + 9 in double-precision arithmetic, respectively. This makes U2 and U3 limiting in their respective arithmetics. This means that one can expect to retain, in the worst cases on 32-bit machines, no digits in single and only 7 digits in double-precision arithmetic. Similar considerations hold for other machines. IERR = 5 indicates an algorithmic failure or machine error that should never occur. Function values are not defined inside a subroutine on IERR = 1, 2, 4, or 5. The use of an error flag in place of print or write statements enhances portability.

The return parameter NZ is the number of components of the answer vector that were set to zero due to underflow. Zero function values are considered logical extensions of underflow, and NZ > 0 is not considered an error. Thus IERR = 0 occurs even though NZ > 0. It is nevertheless important to flag underflows (via NZ > 0) so that the user can take suitable action should a zero value not be appropriate in an application.

Underflow and overflow can be a problem on machines with low exponent ranges. These problems occur quite frequently when $|z|$ or $|\mathrm{Re}(z)|$ is large compared with ν. In these cases, exponential scaling can be used to increase the argument range of the package. For example, the product, written in the form

$$\mathrm{I}_\nu(z_1)\mathrm{K}_\nu(z_2) = e^{|x_1|-z_2}[e^{-|x_1|}\mathrm{I}_\nu(z_1)][e^{z_2}\mathrm{K}_\nu(z_2)],$$
$$z_1 = x_1 + iy_1, \qquad z_2 = x_2 + iy_2,$$

shifts the exponential behavior in $\mathrm{I}_\nu(z_1)$ and $\mathrm{K}_\nu(z_2)$ to the factor $\exp(|x_1| - z_2)$. The quantities in brackets are well scaled and vary slowly with z_1 and z_2. When $x_2 > 0$, $|x_1| - x_2$ is smaller in magnitude than either factor, and $\exp(|x_1| - x_2)$

Table II.

Subroutine	Computation	Callable
GAMLN	$\ln \Gamma(x)$, $x > 0$	Yes
CBINU	$I_\nu(z)$, $\mathrm{Re}(z) \geq 0$	Yes
CBKNU	$K_\nu(z)$, $\mathrm{Re}(z) \geq 0$	Yes
CRATI	$I_{\nu+1}(z)/I_\nu(z)$, $\mathrm{Re}(z) \geq 0$ for CMLRI, CWRSK	Yes
CSHCH	$\sinh z$, $\cosh z$ for CBKNU	Yes
CSERI	$I_\nu(z)$ by power series for $(\lvert z\rvert/2)^\nu \leq \nu + 1$, $\mathrm{Re}(z) \geq 0$	No
CMLRI	$I_\nu(z)$ by Miller algorithm normalized by a Neumann series, $\mathrm{Re}(z) \geq 0$	No
CASYI	$I_\nu(z)$ by asymptotic expansion for $z \to \infty$, $\mathrm{Re}(z) \geq 0$	No
CWRSK	$I_\nu(z)$ by Miller algorithm normalized by a Wronskian, $\mathrm{Re}(z) \geq 0$	No
CBUNI	Driver for CUNI1 and CUNI2	No
CUNI1	$I_\nu(z)$ by (2.1) of reference [4]	No
CUNI2	$I_\nu(z)$ by (2.2) of reference [4]	No
CBUNK	Driver for CUNK1 and CUNK2	No
CUNK1	$K_\nu(z)$ by (2.1), (4.1), (4.2) of reference [4]	No
CUNK2	$K_\nu(z)$ by (2.2), (4.1), (4.2) of reference [4]	No
CACON	Analytic continuation of $K_\nu(z)$ to left-half plane	No
CACAI	Analytic continuation of Ai, Ai′ to left-half plane	No
CUNIK	Parameters for (2.1) of reference [4] for use in CUNI1, CUNK1, CUOIK	No
CUNHJ	Parameters for (2.2) of reference [4] for use in CUNI2, CUNK2, CUOIK	No
CUOIK	Set underflow or overflow indicators; set underflow values for I sequences	No
CUCHK	Checks for component underflow when magnitude $\leq \exp(-\mathrm{ALIM})$ = 1.0E + 3 * R1MACH(1)/TOL	No
CKSCL	Scaled recurrence on underflow for CBKNU	No
CS1S2	Addition of I and K functions for (4.2) of reference [4] on KODE = 2	No
	Additional Double-Precision Routines	
ZMLT	$z_1 * z_2 = z_3$	Yes
ZDIV	$z_1/z_2 = z_3$	Yes
ZSQRT	$\sqrt{z}$, $-\pi < \arg z \leq \pi$	Yes
ZEXP	e^z	Yes
ZLOG	$\ln z$, $-\pi < \arg z \leq \pi$	Yes
ZSHCH	$\sinh z$, $\cosh z$	Yes
ZABS	$\lvert z\rvert$	Yes

is more likely than either $I_\nu(z_1)$ or $K_\nu(z_2)$ to be on scale. Direct computation of the functions on the left is obtained with call list parameter KODE = 1, while the scaled functions in brackets on the right are obtained with KODE = 2. Notice that these manipulations shift the underflow-overflow problem from the package to the user's code where the problem can be managed by combining the exponential with other multiplicative (exponential) factors, if appropriate.

4. SUBROUTINE NAMES AND PURPOSES

The main callable routines of the package are listed in Section 1. This section lists lower level routines, their purposes, and the possibility of a call if one wishes to bypass argument checking and overflow or underflow checking. Unless one has complete information on the range of variables, calls to lower level routines are not recommended.

The names of double-precision complex versions are preceeded by a Z in place of a C. Since a type Double Precision Complex is not available as a standard FORTRAN type, complex numbers are carried as ordered pairs. This necessitates FORTRAN subroutines or functions for a variety of the common mathematical functions. These are listed at the bottom of Table II.

REFERENCES

1. ABRAMOWITZ, M., AND STEGUN, I. A. *Handbook of Mathematical Functions, NBS Applied Math Series 55*. U. S. Dept. of Commerce, Washington, D.C. 1955.
2. AMOS, D. E. Computation of Bessel functions of complex argument. SAND83-0086, Sandia National Laboratories, Albuquerque, N.M. (May 1983).
3. AMOS, D. E. Computation of Bessel functions of complex argument and large order. SAND83-0643, Sandia National Laboratories, Albuquerque, N.M. (May 1983).
4. AMOS, D. E. A subroutine package for Bessel functions of a complex argument and nonnegative order. SAND85-1018, Sandia National Laboratories, Albuquerque, N.M. (May 1985).
5. CODY, W. J. Preliminary report on software for the modified Bessel functions of the first kind. Argonne National Laboratory Rep. TM 357, Argonne, Ill. (Aug. 1980).
6. COWELL, W. R. *Sources and Development of Mathematical Software*. Prentice-Hall, Englewood Cliffs, N. J., 1984.
7. DORING, B. Complex zeros of cylinder functions. *Math. Comput. 20*, (1966), 215–222.
8. FONG, K. W., JEFFERSON, T. H., AND SUYEHIRO, T. Formal conventions of the SLATEC library. *SIGNUM Newsl. 19*, 1 (Jan. 1984), 17–22.
9. FOX, P. A., HALL, A. D., AND SCHRYER, N. L. Framework for a portable library. *ACM Trans. Math. Softw. 4*, 2 (June 1978), 177–188.
10. JONES, R. E., AND KAHANER, D. K. XERROR, The SLATEC error-handling package. SAND82-0800, Sandia National Laboratories, Albuquerque, N. M. (May 1982).
11. SOOKNE, D. J. Bessel functions I and J of complex argument and integer order. *NBS J. Res-B 77B* (1973), 111–113.
12. SOOKNE, D. J. Certification of an algorithm for Bessel functions of complex argument. *NBS J. Res-B 77B* (1973), 133–136.

Received December 1985; revised September 1986; accepted October 1986