EFFICIENT COMPUTATION OF THE CYLINDRICAL BESSEL FUNCTIONS OF COMPLEX ARGUMENT

Ed KUSHNER * and Rick BROUSSARD

Floating Point Systems, Inc., P.O. Box 23489, Portland, OR 97223, USA

Received 16 June 1986

An algorithm that generates the cylindrical Bessel function very accurately for a wide range of complex arguments has been developed by Mason. The Mason algorithm consists of four different methods that apply to different portions of the complex plane. Experience with the Floating Point Systems FPS-364 minisupercomputer indicates several ways by which these methods can be made more efficient. Specific improvements relate to: 1) the method for determination of the point where backward recursion is initiated for the Bessel functions of the first kind; 2) the way that the Bessel functions of the first and second kind are normalized when $|y| < 5$ and $|x| \leqslant 20$; and 3) the extent that asymptotic expansions are used when $|x| > 20$ and $|y| < 5$. The first and third modifications will result in increased efficiency for all architectures. The second modification will be of value for many, but probably not all, architectures.

1. Introduction

The need to compute the Bessel functions of the first kind, $J_n(z)$, the Bessel functions of the second kind, $Y_n(z)$, and Hankel functions, $H_n^1(z)$ and $H_n^2(z)$, arise when investigating the propagation or scattering of electromagnetic and acoustic waves in cylindrical geometries. Absorption effects require that these functions be computed for complex arguments. Ref. [1] describes an algorithm comprised of four methods that cover different portions of the complex plane. Experience with the FPS-364 indicates a number of ways these methods can be made more efficient. The purpose of this paper is to describe these improvements and to indicate the extent each of them reduces computation time.

The nature of Bessel function software is determined to a large extent by the accuracy and dynamic range of the computer to be used. The FPS-364 is a 64-bit computer with 53-bit mantissa and 11-bit exponent that runs at a peak rate of 11 MFLOPS. The division between mantissa and exponent gives 15 significant figures of accuracy with a dynamic range from 2.8×10^{-309} to 9×10^{307}. Numerical experimentation indicated that 12 significant figures would be a reasonable design criterion for accuracy. As for dynamic range, it was decided that the only limitations on the size of the argument or on the maximum order that could be computed were limitations imposed by this parameter. Consequently, the software was designed to provide results within a square region of the complex plane whose size was determined by dynamic range. The size of the Bessel functions approach the limits of dynamic range for the FPS-364 when $y = \pm 600$. Thus, the software was designed to provide accurate results within $\pm 600 \pm 600$i. The extent that dynamic range limits the maximum order that can be computed (call this order NMAX) depends upon the argument z. Numerical experimentation indicated that the dependence can be approximated as an increasing function of $|z|$. Specific values of NMAX near the extremes are

$$\text{NMAX} = 90 \text{ for } |z| = 0.01, \tag{1}$$

$$\text{NMAX} = 750 \text{ for } |z| = 400. \tag{2}$$

* The author to whom inquiries should be addressed.

2. Modifications

2.1. Determination of the point when backward recursion is initiated

Most methods for determining $J_n(z)$ utilize backward recursion. The standard method for determining the point where recursion begins (call this point ISTART) was first described in ref. [2]. This method uses two inputs – the complex argument, z, and the number of order required, N. The method provides results that are as accurate as specified by the user (up to machine precision) but they come at a cost in computation time. A faster method is to compute ISTART as an explicit function of the inputs. The idea here is to generate a data base that characterizes ISTART over the entire region of interest in the complex plane, to fit this data base with simple functions, and then to incorporate these functions in the software that generates the Bessel functions.

Fig. 1 gives the point at which backward recursion must begin in order to obtain 12 significant figures of accuracy in $J_0(z)$ as functions of the real (x) and imaginary (y) part of z. These data were easily fit using polynomials of the form

$$f(z) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^i. \tag{3}$$

Four polynomials, each covering a different portion of the region bounded by $\pm 600 \pm 600\text{i}$, were sufficient to meet the accuracy requirement. Generation of the fit was made easier because

$$f(z) \geqslant \text{ISTART} \tag{4}$$

was actually needed. In other words, the recursion can begin at a point larger than ISTART with the only penalty being a slight increase in computation time.

In most instances ISTART will be greater than N. When z is small and/or N very large then the inequality may be reversed. If such is the case, then setting ISTART $= N$ ensures that all orders are computed and that $J_0(z)$ is accurate. In order to insure that $J_N(z)$ is also accurate, ISTART must, in general, be increased. Numerical experiments indicated a modest increase, typically of the order of 10% or less. Thus, it was relatively easy to generate another polynomial to give this amount.

Data that indicates that relative efficiencies of

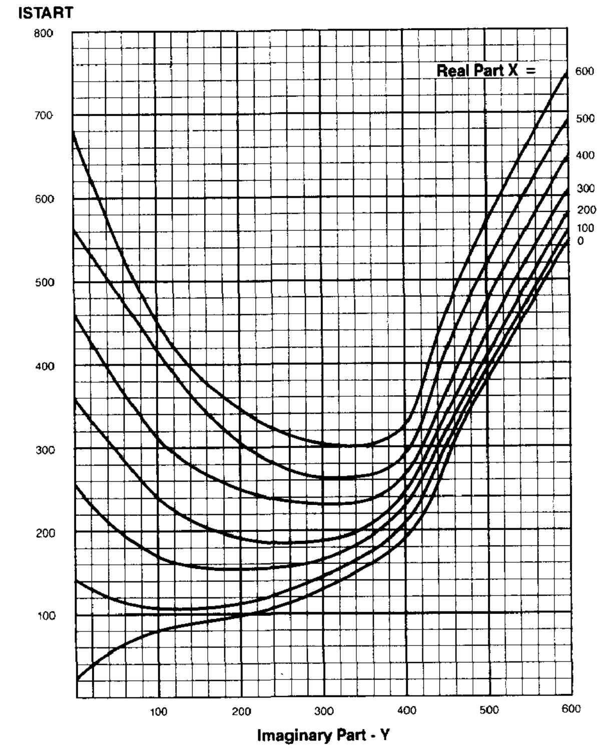

Fig. 1. Initiation point (ISTART) for backward recursion in order to obtain 12 significant figures for $J_0(z)$. Applicable to the region $|x| \leqslant 600$; $|y| \leqslant 600$.

the method described above (method 1) and the method of ref.[2] (method 2) can be found in table 1. Results are presented for a single typical value, taken arbitrarily as $z = 1 + \text{i}$, and for a composite consisting of the average of the computation times for 84 different values in the region

$$0 \leqslant |x| \leqslant 600; \quad 0 < |y| < 5. \tag{5}$$

Values with $|y| \geqslant 5$ were excluded because the time spent in backward recursion in these cases is insignificant when compared with the time spent

Table 1
Comparison of methods for determining the point where backward recursion begins for $J_n(z)$

Argument	Number of orders	Computation time (ms) for $J_n(z)$		
		method 1	method 2	
		Acc = 12 SF	Acc = 12 F	Acc = 7 SF
1 + i	20	0.51	0.74	0.67
	100	1.38	1.96	1.91
composite	20	1.54	1.81	1.59
	100	2.09	2.54	2.37

normalizing the sequence that is the result of the recursion [1]. Two sets of results are given for method #2. The first set was obtained with inputs set to give the same accuracy as method #1. The second set resulted after inputs were relaxed to give seven significant figures. The data indicates that method 1 outperforms method 2 by 15 to 30% when both methods are accurate to 12 significant figures. It also can be seen that method 1 is faster even when the accuracy of method 2 has been reduced to seven significant figures.

2.2. Method for normalizing the recursion sequences for $J_n(z)$ and $Y_n(z)$ in the vicinity of the origin

Ref. [1] recommends the use of power series to compute $J_0(z)$, $J_1(z)$, $Y_0(z)$ and $Y_1(z)$ when $|y| < 5$ and $|y| < 16$. These values are then used to normalize the sequences that are generated for $J_n(z)$ by backward recursion and $Y_n(z)$ by forward recursion. An alternate approach [3] is to use the relationship

$$1 = J_0(z) + 2 \sum_{k=1}^{\infty} J_{2k}(z) \qquad (6)$$

to normalize the first sequence and to use

$$Y_0(z) = \frac{2}{\pi}[\ln(z/2) + \gamma] J_0(z) - \frac{4}{\pi} \sum_{k=1}^{\infty} (-1)^k \frac{J_{2k}(z)}{k}, \qquad (7)$$

$$Y_1(z) = \left[J_1(z) Y_0(z) - \frac{2}{\pi z} \right] / J_0(z), \qquad (8)$$

to obtain initial values for the second sequence. Unnormalized values for the infinite sums can be computed during backward recursion. They come at no additional cost when the recursion is coded in the assembly language of the FPS-364.

It is easily verified that the alternate approach is accurate to machine precision along the real axis, but loses roughly half a significant figure for each unit increase in y. Since the machine used is accurate to 15 significant figures and the accuracy requirements was 12 significant figures, this approach is adequate for $|y| < 5$.

Data in table 2 compare the method of ref. [1] (normalization method 1) with the alternative (normalization method 2). The table gives the time in ms to compute $J_n(z)$ and $Y_n(z)$ for four combinations of the argument and the number of orders requested. The alternative method is a viable option only when the recursion operations

$$J_{n-1}(z) = (2n/z) J_n(z) - J_{n+1}(z) \qquad (9)$$

and

$$Y_{n+1}(z) = (2n/z) Y_n(z) - Y_{n-1}(z) \qquad (10)$$

are in assembly language. Consequently two sets of data are included for normalization method 1 to give a fair comparison. The first set is based on FORTRAN code while the second set was obtained with recursion done in assembly language.

Coding eqs. (9) and (10) in assembly language results in significant reductions in the time to compute the cylindrical Bessel functions. The data in the table indicate that two separate effects explain these reductions. The more important factor is the increased efficiency of code written in assembly language compared to FORTRAN code. Of lesser importance but still significant is the use of the alternative method for normalizing the sequences. A comparison between the second and third columns of the table indicate that this method reduces computation times by 13 to 20%.

Table 2
Comparison of methods for normalizing the recursion sequences. The values include the total time for $J_n(z)$ and $Y_n(z)$

Argument	Number of orders	Computation time (ms)		
		normalization method 1		normalization method 2
		FORTRAN	assembly ~~language~~	
1+i	20	0.75	0.65	0.54
	100	1.96	1.38	0.91
composite *	20	1.75	0.92	0.75
	100	2.60	1.22	1.06

* Composite consists of the 44 arguments of the initial 81 for which $|y| < 5$ and $|x| < 16$.

2.3. Use of asymptotic expansions when y is small and x is large

When $|y| < 5$ and $|x| > 16$, ref. [1] recommends the use of asymptotic expansions to compute $J_0(z)$, $J_1(z)$, $Y_0(z)$ and $Y_1(z)$ to normalize the recursion sequence for $J_n(z)$ and to initiate the sequence for $Y_n(z)$. The relevant equations can be found in ref. [3] as eqs. 9.2.5. through 9.2.10.

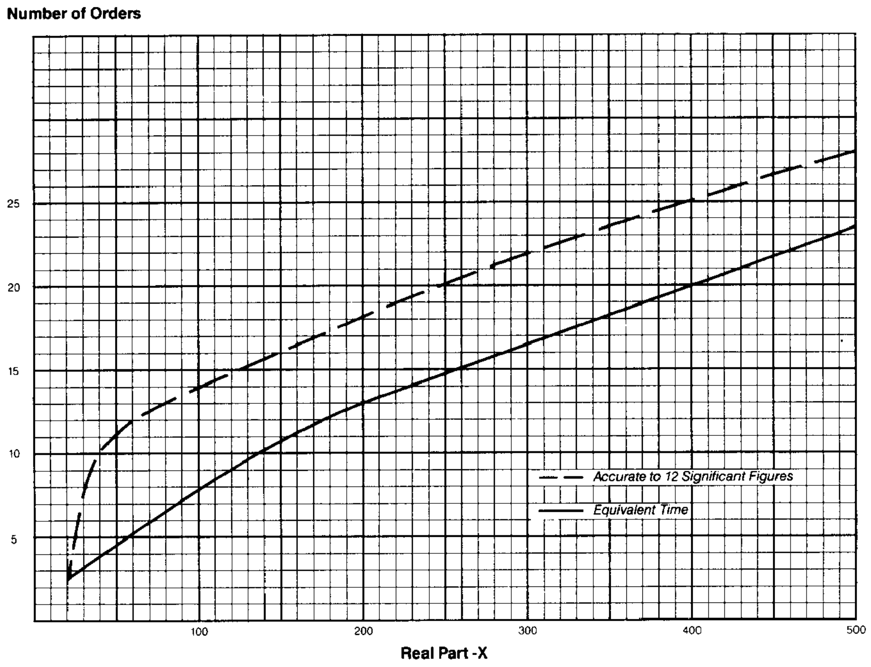

Fig. 2. Number of orders that can be computed using asymptotic expansions for $J_n(z)$ and $Y_n(z)$.

Numerical experimentation indicates that these equations can be used beyond the first two orders.

For fixed argument these equations become less accurate as the number of orders is increased. As the real part of the argument is increased, however, more orders can be computed for a fixed accuracy requirement. The latter effect is seen in the upper curve of fig. 2 which gives the number of orders that can be computed to 12 significant figures as a function of x. The lower curve in the figure gives the points where the computation time is equivalent to the alternative: using recursion to compute $J_2(z)$ through $J_N(z)$ and $Y_2(z)$ through $Y_N(z)$. The modification under discussion is of value when both conditions are met; that is, the region in the figure below the lower curve.

Some explanatory comments relating to fig. 2 are in order. First, the timing comparison is against the implementation in which the recursion is done in assembly language. Hence this curve is machine dependent. Second, the data were generated along the real axis. Similar curves were generated at the other extreme ($y = \pm 5$), and they differed only slightly from those of the figure. Finally, the infinite series represented by eqs. 9.2.9 and 9.2.10 of ref. [3] were truncated at seven terms per function. This fact together with the requirement that all values be accurate to 12 figures probably explains why the switch to asymptotic expansions occurs at $x = 20$ in our implementation, whereas it occurs at $x = 16$ in ref. [1].

The increased use of asymptotic formulae can result in sizable reductions in computation time. An upper limit on the value occurs when $x = 600$ and $N = 3$. For this set of inputs, the time to compute $J_n(z)$ and $Y_n(z)$ was reduced by 70%.

3. Concluding remarks

Ref. [1] provides an accurate method for the computation of the cylindrical Bessel functions for a wide range of complex arguments. This paper has described several modifications that make this method more efficient. The most valuable modification was the coding of all recursion equations in assembly language. This conclusion is machine-dependent and may not apply to other architectures, especially those that cannot achieve full efficiency in operations that involve recursion (e.g., vector computers) [4]. Among the other modifications, explicit calculation of the point where backward recursion is begun and the modified method for normalizing the recursion sequences can result in moderate reductions in computation time for a large portion of the complex plane. The increased use of asymptotic expansions can result in larger reductions but is applicable less frequently.

A final remark is that the method of ref. [1] as modified herein can be the basis for the computation of the modified Bessel functions. The relevant equations are 9.6.3 and 9.6.4 of ref. [3]. Outputs from these equations also appear to be accurate to 12 significant figures within $\pm 600 \pm 600i$.

References

[1] J.P. Mason, Comput. Phys. Commun. 30 (1983) 1.
[2] D.J. Sookne, J. Res. Nat. Bur. Std. 77B (1973) 111.
[3] M. Abramowitz and I. Stegun, eds., Handbook of Mathematical Functions, National Bureau of Standards Applied Mathematics Series 55 (Government Printing Office, Washington, DC, 1964) pp. 364, 375, 386.
[4] R.W. Hockney and C.R. Jesshope, Parallel Computers (Adam Hilger, Bristol, 1981) p. 235.
[5] A.N. Lowan, ed., Table of the Bessel Functions $J_0(z)$ and $J_1(z)$ for Complex Arguments, Mathematical Tables Project, National Bureau of Standards (Columbia Univ. Press, New York, 1947).
[6] A.N. Lowan, ed., Table of the Bessel Functions $Y_0(z)$ and $Y_1(z)$ for Complex Arguments, Computation Laboratory, National Bureau of Standards (Columbia Univ. Press, New York, 1947).
[7] F.W.J. Olver and D.J. Sookne, Math. Comput. 26 (1972) 941.
[8] S. Makinouchi, Tech. Rep. Osaka Univ. 15 (1965) 185.
[9] W.J. Cody, ACM Trans. Math. Soft. 9 (1983) 242.