Fast and Accurate Bessel Function Computation

John Harrison
Intel Corporation, JF1-13
2111 NE 25th Avenue
Hillsboro OR 97124, USA
Email: johnh@ichips.intel.com

Abstract

The Bessel functions are considered relatively difficult to compute. Although they have a simple power series expansion that is everywhere convergent, they exhibit approximately periodic behavior which makes the direct use of the power series impractically slow and numerically unstable. We describe an alternative method based on systematic expansion around the zeros, refining existing techniques based on Hankel expansions, which mostly avoids the use of multiprecision arithmetic while yielding accurate results.

1. Introduction and overview

Bessel functions are certain canonical solutions to the differential equations

$$x^2 \frac{d^2y}{dx^2} + x\frac{dy}{dx} + (x^2 - n^2)y = 0$$

We will consider only the case where n is an integer. The canonical solutions considered are the Bessel functions of the first kind, $J_n(x)$, nonsingular at $x = 0$, and those of the second kind, $Y_n(x)$, which are singular there. In each case, the integer n is referred to as the *order* of the Bessel function. Figure 1 shows a plot of $J_0(x)$ and $J_1(x)$ near the origin, while Figure 2 is a similar plot for $Y_0(x)$ and $Y_1(x)$.

The Bessel functions $J_n(x)$ have power series that are convergent everywhere, with better convergence than the familiar series for the exponential or trigonometric functions:

$$J_n(x) = \sum_{m=0}^{\infty} \frac{(-1)^m (x/2)^{n+2m}}{m!(n+m)!}$$

However, the direct use of the power series would require too many terms for large x, and even for moderate x is likely to be quite numerically unstable close to the zeros. The difficulty is that the final value $J_n(x)$ can be small for large x even when the intermediate terms of the power series are large. The trigonometric functions like $\sin(x)$ also have this property, but they are still quite easy because they are exactly periodic. The Bessel functions are not quite periodic, though they do start to look more and more like scaled trigonometric functions for large x, roughly speaking:[1]

$$J_n(x) \approx \sqrt{\frac{2}{\pi x}}\ \cos(x - [n/2 + 1/4]\pi)$$

$$Y_n(x) \approx \sqrt{\frac{2}{\pi x}}\ \sin(x - [n/2 + 1/4]\pi)$$

For extensive detail on the theory of the Bessel functions, as well as a little history and explanation of how they arise in physical applications, the reader is referred to Watson's monograph [9].

The not-quite periodicity has led to some pessimism about the prospects of computing the Bessel functions with the same kind of relative accuracy guarantees as for most elementary transcendental functions. For example Hart et al. [3] say:

> However, because of the large number of zeros of these functions, it is impractical to construct minimum relative error subroutines, and the relative error is likely to be unbounded in the neighborhood of the zeros.

However, it is important to remember that this was written at a time when giving good relative accuracy guarantees even on the basic trigonometric functions would have been considered impractical too. We shall see that, at least for specific order and specific floating-point precisions, producing results with good relative accuracy is not so very difficult. In this paper we focus exclusively on the functions J_0, J_1, Y_0 and Y_1 in double-precision (`binary64`) floating-point arithmetic. The results should generalize straightforwardly to other specific Bessel functions of integer order and other floating-point formats, but the various parameters, polynomial degrees, domain bounds, worst-case results etc. would need to be computed afresh for each such instance.

Our general approach

The main theme in our proposed approach is to *expand each function about its zeros*. More precisely, we want to

1. These are not, properly speaking, asymptotic results $\sim$, since the zeros of the two functions do not coincide exactly. The exact relationship will be made precise below.

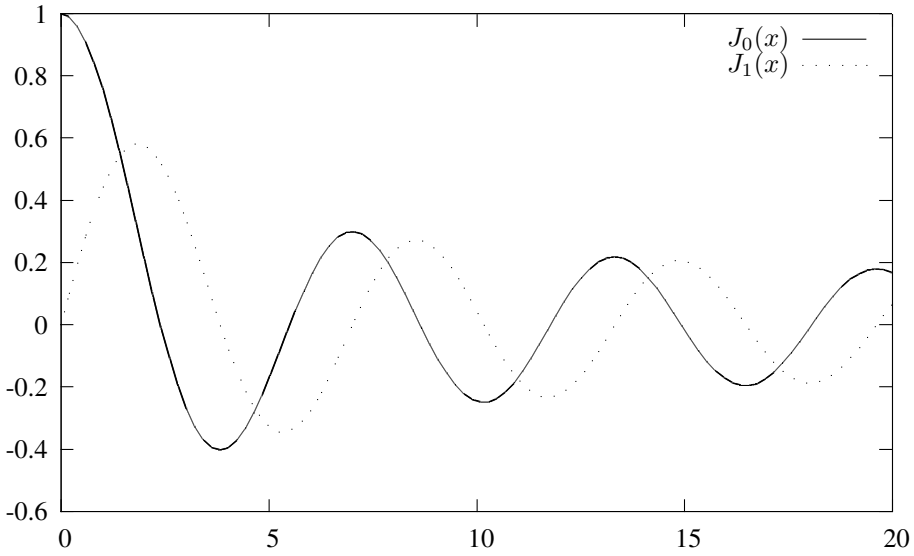

Figure 1. Bessel function of the first kind, J_0 and J_1

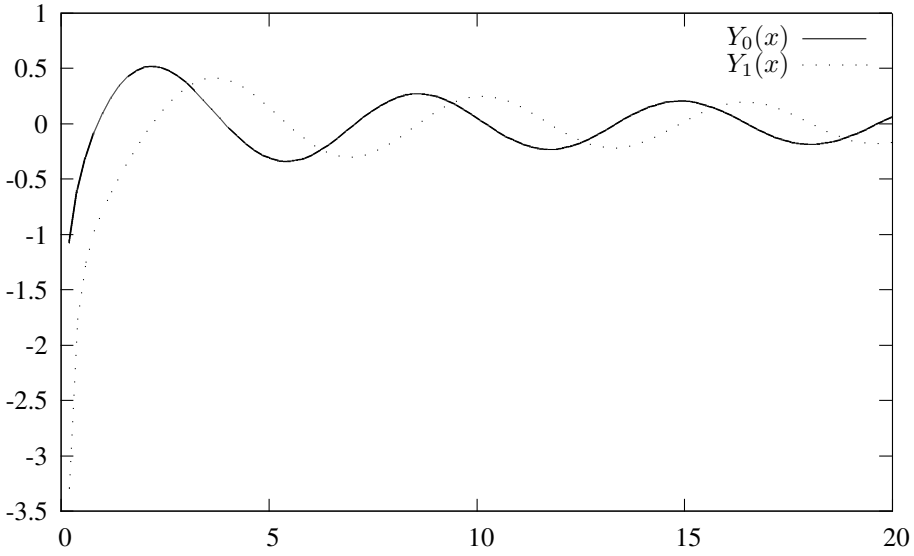

Figure 2. Bessel function of the second kind, Y_0 and Y_1

formulate the algorithms to move the inevitable cancellation forward in the computation to a point before there are rounding errors to be magnified. For example, if the input argument x is close to a zero z, we want to, in effect, compute $x - z$ accurately at once and use that value in subsequent stages.

If we can successfully realize this idea, we can expect accurate results even close to zeros without performing all the intermediate computations in very high precision. Still, in order to assess the accuracy required we need a rigorous examination of the zeros to see how close they may be to double-precision floating-point numbers, and this will also form part of the present document. We will consider in turn:

- Evaluation near 0 for the singular Y_n
- Evaluation for 'small' arguments, roughly $|x| < 45$, but away from the singularities of the Y_n at zero
- Evaluation for 'large' arguments, roughly $|x| \geq 45$.

2. Y_n near singularities

The Y_n have various singularities at $x = 0$, including one of the form $\log(x)$, so it doesn't seem practical to use polynomial or rational approximations in this neighborhood. However, if we incorporate the logarithm and perhaps reciprocal powers, tractable approximations are available. For example we have (see [9] §3.51):

$$Y_0(x) = \frac{2}{\pi}(\begin{array}{l} [\gamma + \log(x/2)]J_0(x) - \\ \sum_{m=1}^{\infty} \frac{(-1)^m (x/2)^{2m}}{(m!)^2} [1 + \frac{1}{2} + \cdots + \frac{1}{m}]) \end{array}$$

If we use general floating-point coefficients, we can write this as follows, using two even power series $W_0(x)$ and $Z_0(x)$:

$$Y_0(x) = W_0(x)\log(x) - Z_0(x)$$

A similar but slightly more complicated expansion also works for $Y_1(x)$, in which case the corresponding power series $W_1(x)$ and $Z_1(x)$ are odd:

$$Y_1(x) = W_1(x)\log(x) - Z_1(x) - \frac{2}{\pi x}$$

For the moderate ranges required, minimax economizations of the $W_i(x)$ and $Z_i(x)$ only require about degree 16 with only alternate terms present for double-precision results.

3. Small arguments

For fairly small arguments, say $|x| < 45$, we can take the idea of expanding around zeros at face value. We don't need to do any kind of computation of zeros at runtime, but can just pre-tabulate the zeros z_1, z_2, $\ldots z_N$, since there aren't so many in range. Then at runtime, given an input x we start by reducing the input argument x to $r = x - z_k$, where z_k is approximately the closest zero to x, and then use an expansion in terms of r. Since even for small arguments the zeros are spaced fairly regularly at intervals of about π — see Table 1 — we can quite efficiently find an appropriate k by a simple division (or more precisely, reciprocal multiplication) and scaling to an integer. If we store z_k in two pieces and subtract the high part first, we get an accurate reduced argument

$$r = (x - z_k^{hi}) - z_k^{lo}$$

We use double-extended precision for the intermediate result, which ensures that the first subtraction is exact. (Since our estimate of the closest zero at the low end can be a little wayward, it is not immediately clear that we could rely on Sterbenz's lemma here if we used double-precision throughout.)

In fact, we tabulate both zeros and extrema among the z_k. This roughly halves the size of each interval of approximation to about $-\pi/4 \le r \le \pi/4$ and so permits us to use polynomials of lower degree. It also removes potential concerns over monotonicity near the extrema where the successive intervals of approximation fit together.

We then use a separate polynomial $p_k(r)$ for each k, with each one precomputed. About most z_k, we can achieve accuracy adequate for double-precision results using polynomials $p_k(r)$ of degree 14. The exceptions are the small zeros of the Y_n where the nearness of the singularities causes problems. Even here, provided in the case of the smallest zero we cut the range off well away from the origin, the polynomials required have tractable degree. Table 2 summarizes the degree of polynomials required to achieve relative errors suitable for the various floating-point precisions for the four functions J_0, J_1, Y_0 and Y_1. We cut off the range for the first zero of Y_0 at $r = -0.1$.

4. Asymptotic expansions

If we turn to larger arguments with about $|x| \ge 45$, the approach traditionally advocated is based on Hankel's asymptotic expansions. The Bessel functions can be expressed as

$$J_n(x) = \sqrt{\frac{2}{\pi x}}(\begin{array}{l} \cos(x - [n/2 + 1/4]\pi) \cdot P_n(x) - \\ \sin(x - [n/2 + 1/4]\pi) \cdot Q_n(x)) \end{array}$$

and

$$Y_n(x) = \sqrt{\frac{2}{\pi x}}(\begin{array}{l} \sin(x - [n/2 + 1/4]\pi) \cdot P_n(x) + \\ \cos(x - [n/2 + 1/4]\pi) \cdot Q_n(x)) \end{array}$$

where the auxiliary functions $P_n(x)$ and $Q_n(x)$ may, for example, be expressed as integrals — see [9] §7.2. For reasonably large values of x these auxiliary functions can be well approximated by asymptotic expansions:

$$P_n(x) \sim \sum_{m=0}^{\infty} \frac{(-1)^m (n, 2m)}{(2x)^{2m}}$$

$$Q_n(x) \sim \sum_{m=0}^{\infty} \frac{(-1)^m (n, 2m+1)}{(2x)^{2m+1}}$$

where the notation (n, m) denotes:

$$(n, m) = \frac{(4n^2 - 1^2)(4n^2 - 3^2) \cdots (4n^2 - [2m-1]^2)}{2^{2m} m!}$$

For example, we have

$$J_0(x) = \sqrt{\frac{2}{\pi x}} [\cos(x - \pi/4) P_0(x) - \sin(x - \pi/4) Q_0(x)]$$

where

$$P_0(x) \sim 1 - \frac{9}{128x^2} + \frac{3675}{32768x^4} - \frac{2401245}{4194304x^6} + \frac{13043905875}{2147483648x^8} - \cdots$$

and

$$Q_0(x) \sim -\frac{1}{8x} + \frac{75}{1024x^3} - \frac{59535}{262144x^5} + \frac{57972915}{33554432x^7} - \cdots$$

J_0	J_1	Y_0	Y_1
2.404825	0.000000	0.893576	2.197141
5.520078	3.831705	3.957678	5.429681
8.653727	7.015586	7.086051	8.596005
11.791534	10.173468	10.222345	11.749154
14.930917	13.323691	13.361097	14.897442
18.071063	16.470630	16.500922	18.043402
21.211636	19.615858	19.641309	21.188068
24.352471	22.760084	22.782028	24.331942
27.493479	25.903672	25.922957	27.475294
30.634606	29.046828	29.064030	30.618286
33.775820	32.189679	32.205204	33.761017
36.917098	35.332307	35.346452	36.903555
40.058425	38.474766	38.487756	40.045944
43.199791	41.617094	41.629104	43.188218
46.341188	44.759318	44.770486	46.330399
49.482609	47.901460	47.911896	49.472505
52.624051	51.043535	51.053328	52.614550
55.765510	54.185553	54.194779	55.756544
58.906983	57.327525	57.336245	58.898496
62.048469	60.469457	60.477725	62.040411
65.189964	63.611356	63.619215	65.182295
68.331469	66.753226	66.760716	68.324152
71.472981	69.895071	69.902224	71.465986
74.614500	73.036895	73.043740	74.607799

Table 1. The approximate values of first zeros of J_0, J_1, Y_0 and Y_1

Precision	Relative error	Typical degree	Worst-case degree
Single	2^{-31}	8-9 (all but 3)	12
Double	2^{-60}	14 (all but 6)	23
Extended	2^{-71}	16 (all but 7)	28

Table 2. Degrees of polynomials needed around small zeros and extrema

Although the series $P_n(x)$ and $Q_n(x)$ actually diverge for all x, they are valid asymptotic expansions, meaning that at whatever point m we truncate the series to give $P_n^m(x)$, we can make the relative error as small as we wish by making x sufficiently large:

$$\lim_{x\to\infty} \frac{|P_n^m(x) - P_n(x)|}{|P_n(x)|} = 1$$

In fact, it turns out — see [9] §7.32 — that the error in truncating the series for $P_n(x)$ and $Q_n(x)$ at a given term is no more than the value of the first term neglected and has the same sign, provided $2p \geq n - 1/2$ where the term in x^{-p} is the last one included

The Hankel formulas are much more appropriate for large x because they allow us to exploit periodicity directly. However they are still numerically unstable near the zeros, because the two terms $\cos(\cdots)P_n(x)$ and $\sin(\cdots)Q_n(x)$ may cancel substantially. This means that we need to compute the trigonometric functions in significantly more than working precision in order to obtain accurate results. In accordance with our general theme, we will try to modify the asymptotic expansions to move the cancellation forward.

Modified expansions

The main novelty in our approach to computing the Bessel functions for large arguments is to expand either $J_n(x)$ or $Y_n(x)$ in terms of a *single* trigonometric function with a further perturbation of its input argument as follows:

$$J_n(x) = \sqrt{\frac{2}{\pi x}} \cdot \beta_n(x) \cos(x - [n/2 + 1/4]\pi - \alpha_n(x))$$

$$Y_n(x) = \sqrt{\frac{2}{\pi x}} \cdot \beta_n(x) \sin(x - [n/2 + 1/4]\pi - \alpha_n(x))$$

where $\alpha_n(x)$ and $\beta_n(x)$ are chosen appropriately. These equations serve to define $\alpha_n(x)$ and $\beta_n(x)$ in terms of the Bessel functions by the following, where the $\equiv$ sign indicates congruence modulo π, i.e. ignoring integer multiples of π. (In some of the plots we eliminate spurious π multiples by successively applying $\tan$ then atan to such expressions.)

$$\begin{aligned} \alpha_n(x) &\equiv x - [n/2 + 1/4]\pi - \mathrm{atan}(Y_n(x)/J_n(x)) \\ &\equiv x - [n/2 + 3/4]\pi + \mathrm{atan}(J_n(x)/Y_n(x)) \end{aligned}$$

and

$$\beta_n(x) = \sqrt{\frac{\pi x}{2}(J_n(x)^2 + Y_n(x)^2)}$$

We will next obtain asymptotic expansions for the $\alpha_n(x)$ and $\beta_n(x)$ starting from those for $P_n(x)$ and $Q_n(x)$. Our derivations are purely formal, but we believe it should be possible to prove rigorously that the error on truncating the expansions is bounded by the first term neglected and has the same sign — see for example the expansion of the closely related 'modulus' and 'phase' functions in §9.2.31 of [1]. Using the addition formula we obtain:

$$J_n(x) = \sqrt{\frac{2}{\pi x}}\cdot\beta_n(x)[\ \cos(x-[n/2+1/4]\pi)\cos(\alpha_n(x))+ \sin(x-[n/2+1/4]\pi)\sin(\alpha_n(x))]$$

Comparing coefficients, we see

$$\beta_n(x)\cos(\alpha_n(x)) = P_n(x)$$

and

$$\beta_n(x)\sin(\alpha_n(x)) = -Q_n(x)$$

These imply that we should choose $\beta_n(x)$ as follows:

$$\begin{aligned}\beta_n(x)^2 &= \beta_n(x)^2[\cos(\alpha_n)^2+\sin(\alpha_n)^2] \\ &= (\beta_n(x)\cos(\alpha_n))^2+(\beta_n(x)\sin(\alpha_n))^2 \\ &= P_n(x)^2+Q_n(x)^2\end{aligned}$$

If we assume $\beta_n(x)$ is nonzero even at the zeros (and it will be since $Q_n(x) \approx 1$ for the large x we are interested in), we can also immediately obtain $\tan(\alpha_n(x)) = -Q_n(x)/P_n(x)$. We can carry through the appropriate computations on formal power series — see §15.52 of [9] for more rigorous proofs of some related results. We have:

$$\begin{aligned}\tan(\alpha_0(x)) &= -Q_0(x)/P_0(x) \\ &= \frac{1}{8x}-\frac{33}{512x^3}+\frac{3417}{16384x^5}-\frac{3427317}{2097152x^7}+\cdots\end{aligned}$$

and composing this with the series for the arctangent function, we obtain

$$\alpha_0(x) = \frac{1}{8x}-\frac{25}{384x^3}+\frac{1073}{5120x^5}-\frac{375733}{229376x^7}+\frac{55384775}{2359296x^9}-\cdots$$

Similarly, using $\beta_0(x) = \sqrt{P_0(x)^2+Q_0(x)^2}$ we get

$$\beta_0(x) = 1-\frac{1}{16x^2}+\frac{53}{512x^4}-\frac{4447}{8192x^6}+\frac{3066403}{524288x^8}-\cdots\cdots$$

In exactly the same way we get:

$$\alpha_1(x) = -\frac{3}{8x}+\frac{21}{128x^3}-\frac{1899}{5120x^5}+\frac{543483}{229376x^7}-\frac{8027901}{262144x^9}+\cdots$$

and

$$\beta_1(x) = 1+\frac{3}{16x^2}-\frac{99}{512x^4}+\frac{6597}{8192x^6}-\frac{4057965}{524288x^8}+\cdots$$

For moderate x, the asymptotic series quickly become good approximations, as shown in Figures 3 and 4, though to get the high accuracy we demand, somewhat larger x and many more terms of the series are necessary, explaining our chosen cutoff point of $|x| \geq 45$. Nevertheless, we can and do utilize a minimax economization of the expansion, which considerably reduces the required degree.

Watson [9] presents (around pp. 213–4) a discussion of some formulas due to Stieltjes for estimating the remainders on truncating the original expansions, and presumably similar analysis could be applied to the $\alpha_n(x)$ and $\beta_n(x)$.

The modified expansions immediately give rise to relatively simple computation patterns by truncating the series appropriately, e.g.

$$J_0(x) \approx \sqrt{\frac{2}{\pi x}}(1-\frac{1}{16x^2}+\frac{53}{512x^4})\cos(x-\frac{\pi}{4}-\frac{1}{8x}+\frac{25}{384x^3})$$

$$Y_0(x) \approx \sqrt{\frac{2}{\pi x}}(1-\frac{1}{16x^2}+\frac{53}{512x^4})\sin(x-\frac{\pi}{4}-\frac{1}{8x}+\frac{25}{384x^3})$$

$$J_1(x) \approx \sqrt{\frac{2}{\pi x}}(1+\frac{3}{16x^2}-\frac{99}{512x^4})\cos(x-\frac{3\pi}{4}+\frac{3}{8x}-\frac{21}{128x^3})$$

$$Y_1(x) \approx \sqrt{\frac{2}{\pi x}}(1+\frac{3}{16x^2}-\frac{99}{512x^4})\sin(x-\frac{3\pi}{4}+\frac{3}{8x}-\frac{21}{128x^3})$$

This seems a much more promising approach because numerical cancellation only needs to be dealt with in the simple algebraic expression $x - \alpha_n(x)$, integrated with standard trigonometric range reduction. The other components of the final answer, a trigonometric function, inverse square root and $\beta_n(x)$ are all well-behaved. The overall computational burden is therefore not much worse than with the trigonometric functions.

We now need to proceed with a careful analysis to show the range limits of this technique and the points at which we can afford to truncate the series. Since the value of $\beta_n(x)$ is approximately 1, the absolute error in it contributes directly to a relative error in the overall result. In the case of $\alpha_n(x)$ however, a given absolute error may translate into a much larger relative error if the result is close to zero. In order to place a bound on how much the error can blow up, we need to know how close the large zeros may come to floating-point numbers.

5. Worst-case zeros

The zeros of the Bessel functions are spaced about π from each other. Assuming that their low-order bits are randomly distributed, we would expect the closest one of the zeros would come to a precision-p floating-point number would be about $2^{-(p+\log_2 p)}$, say 2^{-60} for double precision [6]. Since at least the work of Kahan (see the `nearpi` program) it has been known that this is indeed the case for pure multiples of π, and we might expect similar results here. While from a practical point of view it seems safe to make this kind of naive assumption, and perhaps leave a little margin of safety, a more satisfactory approach is to verify this fact rigorously. We will do so for double precision, our main focus.

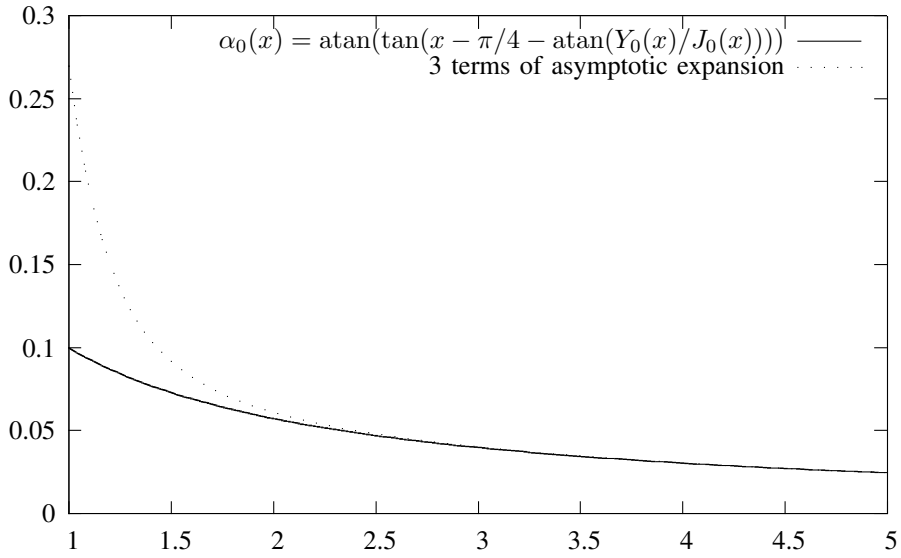

Figure 3. Plot of $\alpha_0(x)$ for moderate x versus approximation

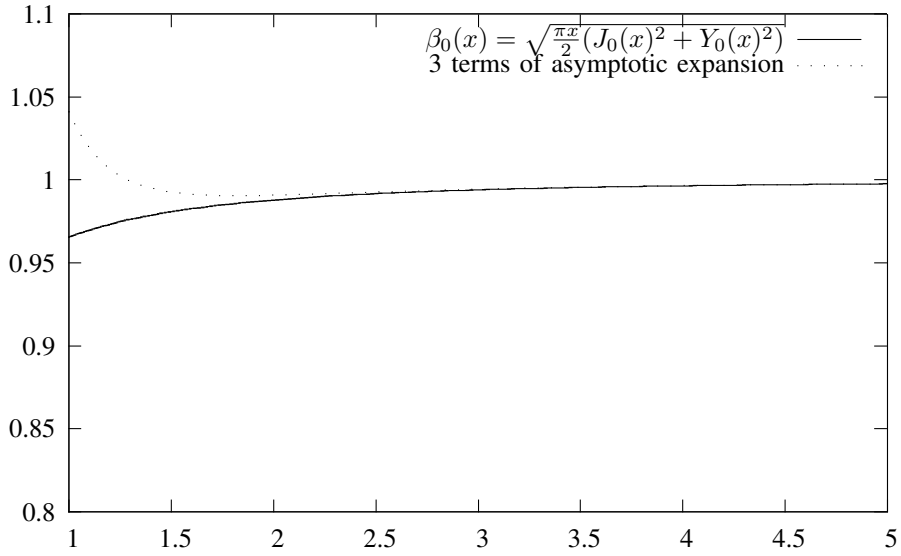

Figure 4. Plot of $\beta_0(x)$ for moderate x versus approximation

Locating the zeros

We have already accurately computed the small zeros of the Bessel functions of interest, since these were needed to produce the various expansions used at the low end. (All these were computed by straightforward use of the infinite series, which can be computationally lengthy but is quite straightforward.) We can therefore simply exhaustively check the small zeros to see how close they come to double-precision numbers.

For larger values of x we use a technique originally due to Stokes — see §15.52 of [9]. We can start with our functions $\alpha_n(x)$ (which were considered by Stokes in this context, though not as part of a computational method) and perform some further power series manipulations to obtain expansions for the various roots. For example, since

$$J_0(x) = \sqrt{\frac{2}{\pi x}} \cdot \beta_0(x) \cos(x - \pi/4 - \alpha_0(x))$$

we seek values for which the cosine term here is zero, i.e.

where for some integer m we have

$$x - \pi/4 - \alpha_0(x) = (m - 1/2)\pi$$

or if we write $p = (m - 1/4)\pi$:

$$x - \alpha_0(x) = p$$

Using the asymptotic expansion we can write:

$$x - \frac{1}{8x} + \frac{25}{384x^3} - \frac{1073}{5120x^5} + \frac{375733}{229376x^7} - \cdots = p$$

or

$$x\left(1 - \frac{1}{8x^2} + \frac{25}{384x^4} - \frac{1073}{5120x^6} + \frac{375733}{229376x^8} - \cdots\right) = p$$

Multiplying both sides by the inverse of the series $1 - \frac{1}{8x^2} + \frac{25}{384x^4} - \cdots$ we get:

$$x = p\left(1 + \frac{1}{8x^2} - \frac{19}{384x^4} + \frac{2999}{15360x^6} - \frac{16352423}{10321920x^8} + \cdots\right)$$

and therefore:

$$\frac{1}{p} = \frac{1}{x} + \frac{1}{8x^3} - \frac{19}{384x^5} + \frac{2999}{15360x^7} - \frac{16352423}{10321920x^9} + \cdots$$

By reversion of the power series we obtain:

$$\frac{1}{x} = \frac{1}{p} - \frac{1}{8p^3} + \frac{37}{384p^5} - \frac{1373}{5120p^7} + \frac{19575433}{10321920p^9} - \cdots$$

so

$$\frac{p}{x} = 1 - \frac{1}{8p^2} + \frac{37}{384p^4} - \frac{1373}{5120p^6} + \frac{19575433}{10321920p^8} - \cdots$$

Inverting the series again we get

$$\frac{x}{p} = 1 + \frac{1}{8p^2} - \frac{31}{384p^4} + \frac{3779}{15360p^6} - \frac{6277237}{3440640p^8} + \cdots$$

and so we finally obtain x in terms of $p = (m - 1/4)\pi$:

$$x = p + \frac{1}{8p} - \frac{31}{384p^3} + \frac{3779}{15360p^5} - \frac{6277237}{3440640p^7} + \cdots$$

These power series computations are quite tedious by hand but can be handled automatically in some computer algebra systems. We used our own implementation of power series in the OCaml language via higher-order functions [5], and with this we can obtain any reasonable truncation of these series automatically in an acceptable time. Needless to say, our results are consistent with the hand computations in [9], though often we need more terms than are given there. Zeros of the other Bessel functions are obtained in the same way. For $Y_0(x)$ the expansion is the same except that we use $p = (m + 1/4)\pi$, and we have the following expansion for the zeros of $J_1(x)$ and $Y_1(x)$:

$$x = p - \frac{3}{8p} + \frac{3}{128p^3} - \frac{1179}{5120p^5} + \frac{1951209}{1146880p^7} - \frac{223791831}{9175040p^9} + \cdots$$

where $p = (m + 1/4)\pi$ for $J_1(x)$ and $p = (m - 1/4)\pi$ for $Y_1(x)$.

For moderate m, say up to a few thousand, million, or even billion, we can examine the zeros exhaustively using a suitable truncation of this formula to approximate them. And for $m > 2^{70}$ or so, all terms but the first are sufficiently small in magnitude that we can reduce matters to the well-known linear case, which has already been analyzed for trigonometric range reduction. In the middle range, we have applied the Lefèvre-Muller technique [4] to the Stokes expansions. The idea of this technique is to cover the range with a large (but quite feasible) number of linear approximations that are accurate enough for us to be able to read off local results from. We then find 'worst cases' for the linear approximations in a straightforward manner by scaling the function so its gradient is approximated by a suitable rational approximation a/b, then solving congruences mod b.

Results

Table 3 shows all the double-precision floating-point numbers in the range $0 \leq x \leq 2^{90}$ that are within 2^{-55} of a zero of one of our four basic functions $J_0(x)$, $J_1(x)$, $Y_0(x)$ and $Y_1(x)$. As noted above, there are no surprises beyond 2^{90}, since the zeros are so close to $(n - 1/4)\pi$ that we know roughly what to expect from previous studies on the trigonometric functions — see e.g. [8] and [6]. (Note that if we have results for double-precision numbers close to $n\pi$, the results for multiples $n\pi/4$ and *a fortiori* $(n - 1/4)\pi$ can be at worst $1/4$ as large.) Indeed, the larger zeros of J_0 and Y_1, and those of J_1 and Y_0, are already becoming very close near the top of our range. Still, for completeness, Table 4 lists the closest zeros of the functions to double-precision numbers over the whole range. The overall lesson is that the results are in line with naive statistical expectations.

6. A reference implementation

To realize the ideas set out above, we have programmed a reference implementation of the double-precision Bessel functions J_0, J_1, Y_0 and Y_1 designed for the Intel® Itanium® architecture. This architecture allows us to use double-extended precision for internal calculations, so most of the computation can be done 'naively' with good final error bounds. The sole exception is the accurate computation of $\alpha_n(x)$ and its integration with trigonometric range reduction. Here we make good use of the `fma` (fused multiply-add) that this architecture offers.

The higher-order part of the $\alpha_n(x)$ series, consisting of terms that are still below about 2^{-60} in size, can also be calculated naively. But for the low-order part of the series we want to compute the summation in two double-extended pieces to ensure absolute accuracy of order 2^{-120}. One reasonable way of doing this is to make repeated use of

Exact value of double	Approximate decimal value	Distance from zero	Function
$2^{14} \times 6617649673795284$	$1.08423572255 \times 10^{20}$	$2^{-58.438199858}$	Y_1
$2^{37} \times 6311013172270677$	$8.67379045745 \times 10^{26}$	$2^{-58.4361612221}$	Y_1
$2^{37} \times 6311013172270677$	$8.67379045745 \times 10^{26}$	$2^{-58.4361612218}$	J_0
$2^{14} \times 6617649673795284$	$1.08423572255 \times 10^{20}$	$2^{-58.4356039989}$	J_0
$2^{-14} \times 4643410941512688$	283411312348.0	$2^{-57.4750503229}$	J_0
$2^{23} \times 7209129755475690$	$6.04745635398 \times 10^{22}$	$2^{-57.2168697228}$	J_1
$2^{23} \times 7209129755475690$	$6.04745635398 \times 10^{22}$	$2^{-57.216867725}$	Y_0
$2^{28} \times 8451279557623493$	$2.26862308183 \times 10^{24}$	$2^{-57.1898493381}$	Y_1
$2^{28} \times 8451279557623493$	$2.26862308183 \times 10^{24}$	$2^{-57.1898492858}$	J_0
$2^{-26} \times 8368094255856943$	124694321.392	$2^{-56.9469283257}$	Y_0
$2^{16} \times 4963237255346463$	$3.25270716766 \times 10^{20}$	$2^{-56.8515062657}$	J_1
$2^{16} \times 4963237255346463$	$3.25270716766 \times 10^{20}$	$2^{-56.8512179523}$	Y_0
$2^{32} \times 8754199225116346$	$3.75989993745 \times 10^{25}$	$2^{-56.8148254699}$	Y_1
$2^{32} \times 8754199225116346$	$3.75989993745 \times 10^{25}$	$2^{-56.8148254675}$	J_0
$2^{-18} \times 7757980709970194$	29594347801.1	$2^{-56.6251590484}$	Y_0
$2^{-42} \times 5944707359537560$	1351.66996177	$2^{-56.5865796184}$	J_1
$2^{-22} \times 5798262669118148$	1382413546.83	$2^{-56.3577856958}$	J_0
$2^{-30} \times 4670568619103095$	4349805.99126	$2^{-56.1575895598}$	J_1
$2^{16} \times 8272062092244105$	$5.42117861277 \times 10^{20}$	$2^{-56.1144022739}$	Y_1
$2^{16} \times 8272062092244105$	$5.42117861277 \times 10^{20}$	$2^{-56.1142984844}$	J_0
$2^{0} \times 6027843377079719$	$6.02784337708 \times 10^{15}$	$2^{-55.9891793419}$	J_0
$2^{-10} \times 5256649930600386$	$5.13344719785 \times 10^{12}$	$2^{-55.7177747539}$	J_0
$2^{-16} \times 6535297120514194$	99720720222.7	$2^{-55.7166597393}$	Y_0
$2^{-35} \times 6458928246558283$	187979.55262	$2^{-55.5787196547}$	Y_1
$2^{-26} \times 5245948062070016$	78170717.6875	$2^{-55.509211239}$	Y_0
$2^{-20} \times 7547179409128835$	7197551163.8	$2^{-55.4766161411}$	Y_1
$2^{-26} \times 8441237061651159$	125784234.131	$2^{-55.4755750981}$	Y_0
$2^{-45} \times 7046625970325583$	200.277155793	$2^{-55.4323204082}$	J_0
$2^{-40} \times 8936924570334870$	8128.08554686	$2^{-55.3841525656}$	J_1
$2^{-29} \times 8114890393276829$	15115161.2276	$2^{-55.1139949895}$	J_1
$2^{-53} \times 8048625784723434$	0.893576966279	$2^{-55.0615557705}$	Y_0

Table 3. Doubles in range $[0, 2^{90}]$ within 2^{-55} of Bessel zero

a special 2-part Horner step, which takes an existing 2-part value (h, l) and produces a new one (h', l') using 2-part coefficients (c_H, c_L) and dividing by a 2-part variable $x_H + x_L$. (In our application we have $x_H + x_L = x^2$ and the (c_H, c_L) are 2-part floating-point approximations to the α_n series coefficients.) That is, we get to good accuracy:

$$(h' + l') = (c_H + c_L) + \frac{1}{x_H + x_L}(h + l)$$

In a preamble, which only needs to be executed once for a given $x_H + x_L$, we compute an approximate inverse y and correction e:

$$\begin{aligned} y &= 1/x_H \\ e_1 &= 1 - x_H \cdot y \\ e_2 &= e_1 - x_L \cdot y \\ e &= y \cdot e_2 \end{aligned}$$

Each Horner step then consists of the following computation. One can show that provided the terms decrease in size (as they do in our application), this maintains a relative error of about twice the working precision.

$$\begin{aligned} h' &= c_H + y \cdot h \qquad t = c_L + e \cdot h \\ r_1 &= h' - c_H \\ r &= y \cdot h - r_1 \\ s &= t + r \\ l' &= s + y \cdot l \end{aligned}$$

The latency may seem like 5 `fma`s, but note that we get h' from h in just one latency and only use l to obtain l' on the last step. Thus we can in fact pipeline a number of successive applications separated only by one `fma` latency.

If we have a 2-part result (r, c) from argument reduction and (h, l) is a 2-part $\alpha_n^{(1)} + \alpha_n^{(3)}/x^2 + \alpha_n^{(5)}/x^4 + \cdots$ then we can make the final combination using the same step with only a 1-part x needed (this is just our input number x, whereas earlier we were dividing by x^2 at each stage):

$$(h' + l') = (r + c) + \frac{1}{x}(h + l)$$

and then perform a conventional floating-point addition $h' + l'$ to obtain the final result r as a double-extended number. Since this is bounded by approximately $|r| \leq \pi/4$ and the sine and cosine functions are well-conditioned in this area, we can just evaluate $\sin(r)$ or $\cos(r)$ straightforwardly.

Exact value of double	Approximate decimal value	Distance from zero	Function
$2^{796} \times 6381956970095103$	$2.65968632416 \times 10^{255}$	$2^{-61.8879179362}$	Y_0
$2^{796} \times 6381956970095103$	$2.65968632416 \times 10^{255}$	$2^{-61.8879179362}$	J_1
$2^{78} \times 5916243447979695$	$1.78807486485 \times 10^{39}$	$2^{-59.9300883695}$	Y_1
$2^{78} \times 5916243447979695$	$1.78807486485 \times 10^{39}$	$2^{-59.9300883695}$	J_0
$2^{938} \times 8444920710073313$	$1.96214685729 \times 10^{298}$	$2^{-59.7839697826}$	Y_0
$2^{938} \times 8444920710073313$	$1.96214685729 \times 10^{298}$	$2^{-59.7839697826}$	J_1
$2^{524} \times 5850965514341686$	$3.21325554979 \times 10^{173}$	$2^{-59.4812472194}$	Y_1
$2^{524} \times 5850965514341686$	$3.21325554979 \times 10^{173}$	$2^{-59.4812472194}$	J_0
$2^{807} \times 8160885118204141$	$6.96536316842 \times 10^{258}$	$2^{-59.1402789847}$	Y_1
$2^{807} \times 8160885118204141$	$6.96536316842 \times 10^{258}$	$2^{-59.1402789847}$	J_0
$2^{627} \times 8360820580228475$	$4.65646330711 \times 10^{204}$	$2^{-59.0913203893}$	Y_0
$2^{627} \times 8360820580228475$	$4.65646330711 \times 10^{204}$	$2^{-59.0913203893}$	J_1
$2^{144} \times 8583082635084172$	$1.91409138863 \times 10^{59}$	$2^{-59.0538090794}$	Y_1
$2^{144} \times 8583082635084172$	$1.91409138863 \times 10^{59}$	$2^{-59.0538090794}$	J_0
$2^{242} \times 7958046405119485$	$5.6242603729 \times 10^{88}$	$2^{-58.9935827902}$	Y_0
$2^{242} \times 7958046405119485$	$5.6242603729 \times 10^{88}$	$2^{-58.9935827902}$	J_1
$2^{561} \times 6808218460873451$	$5.13879213026 \times 10^{184}$	$2^{-58.9110422712}$	Y_0
$2^{561} \times 6808218460873451$	$5.13879213026 \times 10^{184}$	$2^{-58.9110422712}$	J_1
$2^{200} \times 7636753411044619$	$1.22717895908 \times 10^{76}$	$2^{-58.8981116632}$	Y_1
$2^{200} \times 7636753411044619$	$1.22717895908 \times 10^{76}$	$2^{-58.8981116632}$	J_0
$2^{193} \times 6366906923947931$	$7.99314450027 \times 10^{73}$	$2^{-58.5326981594}$	Y_1
$2^{193} \times 6366906923947931$	$7.99314450027 \times 10^{73}$	$2^{-58.5326981594}$	J_0
$2^{14} \times 6617649673795284$	$1.08423572255 \times 10^{20}$	$2^{-58.438199858}$	Y_1
$2^{37} \times 6311013172270677$	$8.67379045745 \times 10^{26}$	$2^{-58.4361612221}$	Y_1
$2^{37} \times 6311013172270677$	$8.67379045745 \times 10^{26}$	$2^{-58.4361612218}$	J_0
$2^{14} \times 6617649673795284$	$1.08423572255 \times 10^{20}$	$2^{-58.4356039989}$	J_0
$2^{886} \times 5648695676206402$	$2.91423343144 \times 10^{282}$	$2^{-58.152019003}$	Y_0
$2^{886} \times 5648695676206402$	$2.91423343144 \times 10^{282}$	$2^{-58.152019003}$	J_1
$2^{968} \times 6221301883130153$	$1.55209063452 \times 10^{307}$	$2^{-58.1018949515}$	Y_1
$2^{968} \times 6221301883130153$	$1.55209063452 \times 10^{307}$	$2^{-58.1018949515}$	J_0
$2^{502} \times 6951690029616219$	$9.10223874078 \times 10^{166}$	$2^{-58.0556510652}$	Y_0
$2^{502} \times 6951690029616219$	$9.10223874078 \times 10^{166}$	$2^{-58.0556510652}$	J_1
$2^{525} \times 8776448271512529$	$9.63976664937 \times 10^{173}$	$2^{-57.8962847187}$	Y_0
$2^{525} \times 8776448271512529$	$9.63976664937 \times 10^{173}$	$2^{-57.8962847187}$	J_1
$2^{90} \times 6101578227064009$	$7.55338799011 \times 10^{42}$	$2^{-57.8318447312}$	Y_0
$2^{90} \times 6101578227064009$	$7.55338799011 \times 10^{42}$	$2^{-57.8318447312}$	J_1
$2^{636} \times 8438258240500718$	$2.40619075865 \times 10^{207}$	$2^{-57.7878018343}$	Y_0
$2^{636} \times 8438258240500718$	$2.40619075865 \times 10^{207}$	$2^{-57.7878018343}$	J_1
$2^{129} \times 4595526034082901$	$3.12755295225 \times 10^{54}$	$2^{-57.7272223193}$	Y_1
$2^{129} \times 4595526034082901$	$3.12755295225 \times 10^{54}$	$2^{-57.7272223193}$	J_0
$2^{434} \times 7750232865711478$	$3.43821372626 \times 10^{146}$	$2^{-57.7099830586}$	Y_1
$2^{434} \times 7750232865711478$	$3.43821372626 \times 10^{146}$	$2^{-57.7099830586}$	J_0

Table 4. Doubles closest to Bessel zero

But in the interest of speed we use our own custom implementation of these trigonometric functions that bypasses the usual range reduction step, because this range reduction has already been performed, as described next.

One may be able to use an off-the-shelf argument reduction routine by reducing modulo $\pi/4$ and then modifying the result, provided that enough information about the quotient as well as the remainder is provided. Note that one can always adapt an argument reduction for $2^a\pi$ for $2^b\pi$ by multiplying the input and dividing the output(s) by 2^{a-b}. Instead, we programmed our own variant of the standard scheme [7], which optimizes for speed and simplicity at the cost of much greater storage. For an input $x = 2^a m$ with $m \in \mathbb{Z}$ we observe that

$$(x/\pi) \mod 1 = (2^a m/\pi) \mod 1 = (m \cdot 2^a/\pi) \mod 1$$

We may apply the modulus twice within the calculation without changing the result, since m is an integer:

$$(2^a m/\pi) \mod 1 = (m \cdot [(2^a/\pi) \mod 1]) \mod 1$$

In IEEE double precision there are only about 2048 possibilities for a, so we can simply precompute and tabulate the values:

$$P_a = ((2^a/\pi) \mod 1)/2^a$$

so we just need to calculate at runtime $(P_a \cdot x) \mod 1$. Each P_a is stored in three parts using floating-point numbers, so $P_a = P_a^{hi} + P_a^{med} + P_a^{lo}$. The final computation uses some straightforward `fma`-based techniques to ensure accuracy in the computations, also adding and subtracting a 'shifter' $S = 2^{62} + 2^{61}$ to fix the binary point and force integer truncation. The final result x_2 is an accurate reduced

argument $(x/\pi) \bmod 1$, though it may be a little greater than $1/2$ in magnitude, which can then be postmultiplied by π.

$$\begin{aligned} N_S &= S + x \cdot P_a^{hi} \\ N &= S - N_S \\ x_0 &= N + x \cdot P_a^{hi} \\ x_1 &= x_0 + x \cdot P_a^{med} \\ x_2 &= x_1 + x \cdot P_a^{lo} \end{aligned}$$

The complete code (written in C) runs in about 160 cycles for all arguments, and with more aggressive scheduling and optimization, we believe it could be brought below 100 cycles.

7. Further work

Our α_n series were designed to achieve better than 2^{-120} absolute accuracy over the whole range. However, this very high degree of accuracy is only required in the neighborhood of the zeros. It would be interesting to investigate computing a minimax economization using a more refined weight function to take this into account; it may be that a significant further reduction in the degree is possible. At least we could explore more sophisticated methods of arriving at minimax approximations, taking into account the fact that the coefficients are machine numbers [2].

In general the above techniques should generalize straightforwardly to J_n and Y_n for fixed, moderate n, yielding algorithms that are fast and accurate at the cost of relatively large tables. The problem of implementing generic functions when n is another parameter is potentially more difficult, and has not yet been considered. Another more ambitious goal still would be to strive for perfect rounding, but for this we would need more comprehensive 'worst case' data for the functions in general, not merely for their zeros.

Our current code is just a simple prototype, and makes use of the somewhat uncommon combination, found on the Intel® Itanium® architecture, of the double-extended floating-point type and the fused multiply-add. It should not be too difficult to produce a portable C implementation that would run on any reasonable platform supporting IEEE floating-point, though the performance figures would probably not be quite as good as those reported here. This could be of fairly wide interest, since we are not aware of any other double-precision Bessel function implementations offering the combination of accuracy and speed we attain here.

Acknowledgements

The author is grateful to Peter Tang, both for suggesting the Bessel functions as an interesting line of research and for his explanation of the established theory. Thanks also to the anonymous reviewers for ARITH, whose comments were of great help in improving the paper.

References

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions With Formulas, Graphs, and Mathematical Tables*, volume 55 of *Applied Mathematics Series*. US National Bureau of Standards, 1964.

[2] N. Brisebarre, J.-M. Muller, and A. Tisserand. Computing machine-efficient polynomial approximations. *ACM Transactions on Mathematical Software*, 32:236–256, 2006.

[3] J. F. Hart, E. W. Cheney, C. L. Lawson, H. J. Maehly, C. K. Mesztenyi, J. R. Rice, H. G. Thatcher, and C. Witzgall. *Computer Approximations*. Robert E. Krieger, 1978.

[4] V. Lefèvre and J.-M. Muller. Worst cases for correct rounding of the elementary functions in double precision. Research Report 4044, INRIA, 2000.

[5] M. D. McIlroy. Squinting at power series. *Software — Practice and Experience*, 20:661–683, 1990.

[6] J.-M. Muller. *Elementary functions: Algorithms and Implementation*. Birkhäuser, 2nd edition, 2006.

[7] M. Payne and R. Hanek. Radian reduction for trigonometric functions. *SIGNUM Newsletter*, 18(1):19–24, 1983.

[8] R. A. Smith. A continued fraction analysis of trigonometric argument reduction. *IEEE Transactions on Computers*, 44:1348–1351, 1995.

[9] G. N. Watson. *A treatise on the theory of Bessel functions*. Cambridge University Press, 1922.