Fast and accurate Bessel function computation

John Harrison, Intel Corporation

ARITH-19 Portland, OR

Tue 9th June 2009 (11:00 – 11:30)

Bessel functions and their computation

Bessel functions are certain canonical solutions to the differential equations

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

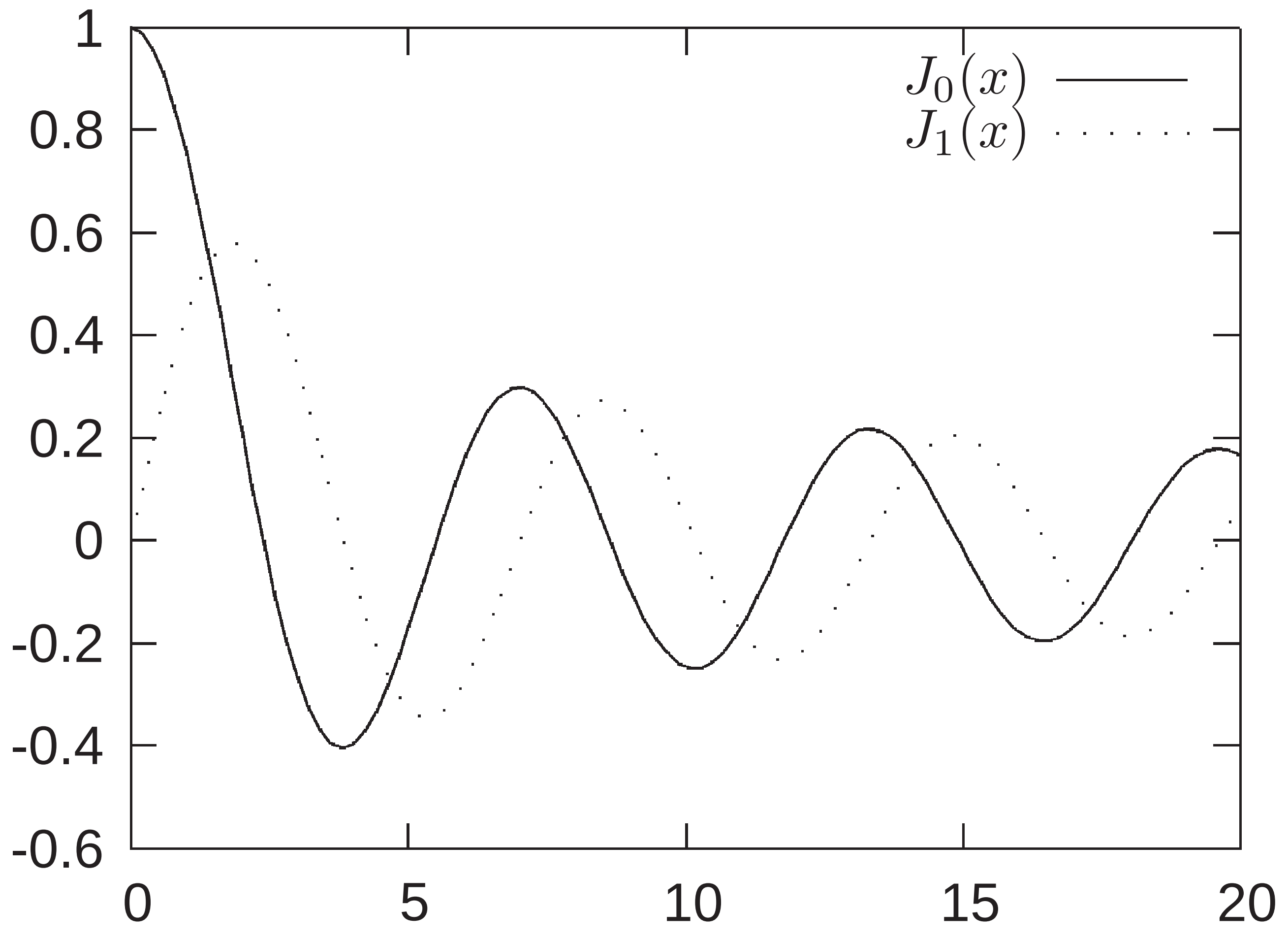
They often appear when analyzing physical systems with cylindrical symmetry.

Bessel functions of the first kind $J_n(x)$ are nonsingular at the origin; those of the second kind $Y_n(x)$ are singular there.

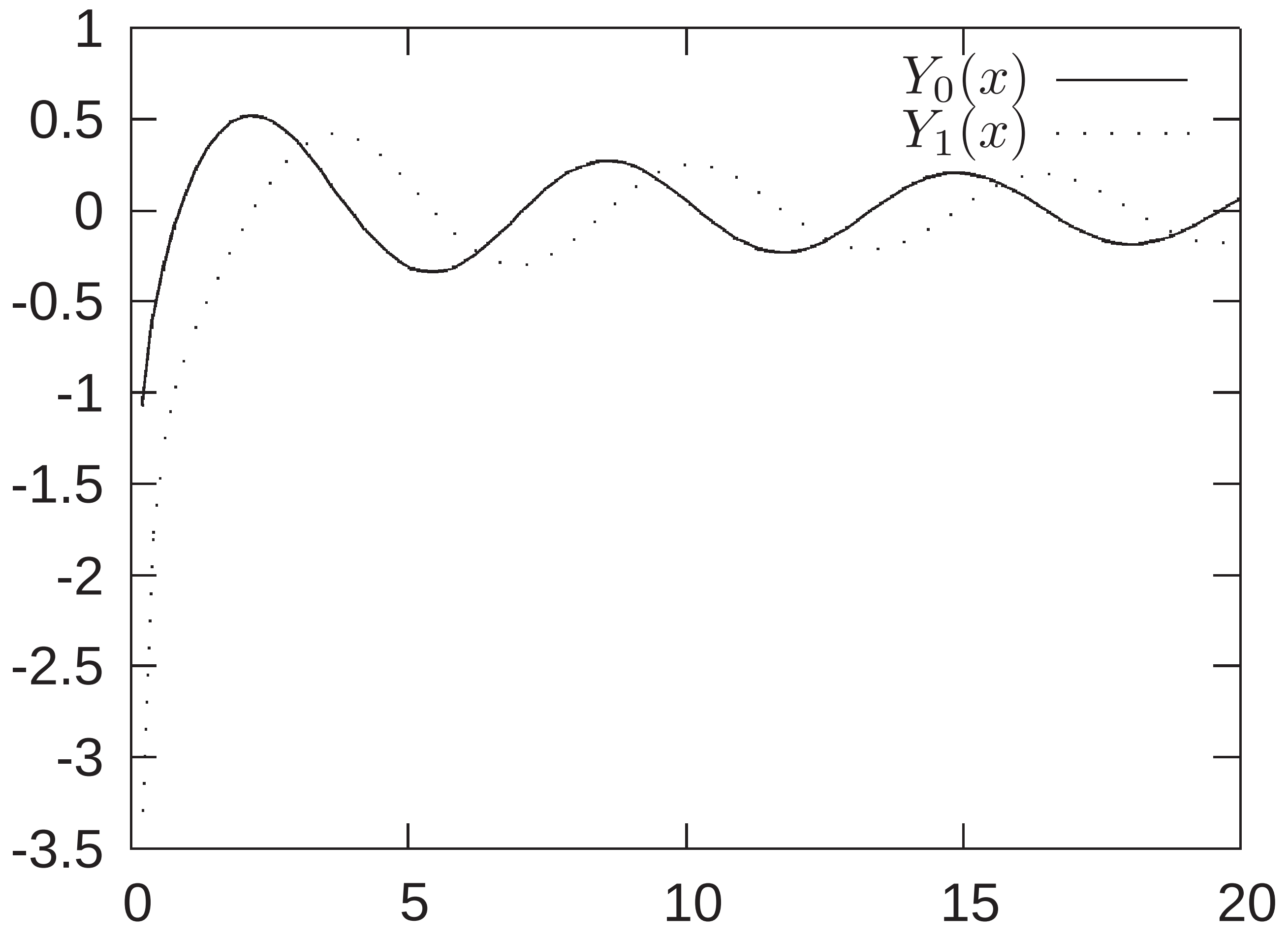
Can be defined via power series, e.g.

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

Bessel functions of the first kind: $J_0(x)$ and $J_1(x)$



Bessel functions of the second kind: $Y_0(x)$ and $Y_1(x)$



Why are they difficult?

There has been general pessimism about computing them even with the usual relative/ulp error guarantee (let alone perfect rounding).

> . . . 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. [Hart et al., “Computer Approximations”]

Why are they difficult?

There has been general pessimism about computing them even with the usual relative/ulp error guarantee (let alone perfect rounding).

> ... 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. [Hart et al., “Computer Approximations”]

Our goal is to try to achieve the usual relative error guarantees without sacrificing (much) performance.

- Just for fixed precision (double here, could generalize)
- For specific order n, not as binary functions

How to solve the problem?

The trigonometric functions suffer from analogous problems:

- They can in principle be computed via simple power series that converge everywhere.
- In practice, that would be inefficient, and inaccurate in relative terms near the zeros.

How to solve the problem?

The trigonometric functions suffer from analogous problems:

- They can in principle be computed via simple power series that converge everywhere.
- In practice, that would be inefficient, and inaccurate in relative terms near the zeros.

But the trig functions are much easier because the zeros are evenly spaced. We know very well by now how to do accurate range reduction by $\pi/2$ or whatever.

Can we do something analogous for Bessel functions?

In other words, move the inevitable cancellation back as much as possible?

Small zeros of Bessel functions

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

Computation near small zeros

For each ‘small’ zero ($|x| < 45$ for double precision), we store:

- The zero itself in two pieces $z_k = z_k^{hi} + z_k^{lo}$
- Coefficients for a power series expansion $\sum_{i=0}^{n_k} a_i(x - z_k)^i$.

Given an argument x in this range, we find roughly the closest zero, then compute a reduced argument $(x - z_k^{hi}) - z_k^{lo}$ and use the power series.

In fact we use extrema as well as zeros for the z_k, to make the reduced argument smaller and avoid monotonicity worries.

Hankel expansions (1)

For larger arguments, the traditional approach uses Hankel's asymptotic expansions:

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

and

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

where $P_n(x)$ and $Q_n(x)$ can be defined by integrals.

Hankel expansions (2)

For large arguments the functions $P_n(x)$ and $Q_n(x)$ 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:

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

Modified expansions

The Hankel expansions are still not a complete solution because the $\sin$ and $\cos$ terms can cancel. We want to move the cancellation forward into simpler expressions by writing

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

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

for some functions $\alpha_n(x)$ and $\beta_n(x)$.

Asymptotic expansions for $\alpha(x)$ and $\beta(x)$

We can find asymptotic expansions for $\alpha_n(x)$ and $\beta_n(x)$ from those for $P_n(x)$ and $Q_n(x)$ by formal power series manipulations.

The functions $\alpha_n(x)$ were already used extensively by Stokes to analyze zeros of the Bessel functions, though not as part of a computational method. See also the ‘modulus’ and ‘phase’ functions in Abramowitz and Stegun.

We have succeeded in moving cancellation back into a purely algebraic expression.

We now only need to apply the correction $\alpha_n(x)$ as an additional tweak to a standard trigonometric range reduction.

Computation patterns using modified expansions

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

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

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

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

How much cancellation?

Even though we've moved the cancellation back into a simple expression, we still need to understand how bad it can be, to decide how much extra precision we must use.

Compare the analysis of standard trigonometric range reduction: how small can $x - N\pi$ get for floating-point $x \neq 0$? Answer: about 2^{-60}.

We want to know how small $x - N\pi/4 - \alpha_n(x)$ can get. The answer is probably similar, but it would be better to *prove* that.

Finding the zeros

To find zeros of $J_0(x)$, recall

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

We seek values for which the cosine term here is zero, i.e. where for some integer N we have

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

Writing $p = (N - 1/4)\pi$, this is $x - \alpha_0(x) = p$, which we can revert to get

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

Analysis of zeros

We can analyze how close the zeros are to floating-point numbers:

- For small $|x| < 45$, we already have the small zeros accurately tabulated.
- For $|x| > 2^{70}$, the $\alpha_0(x)$ correction is negligible and we can take existing results for ordinary trig range reduction.
- In between we use the Levèvre-Muller technique applied to the asymptotic series for the zero in terms of p

Worst-case zeros for $|x| \leqslant 2^{90}$

There are indeed no unpleasant surprises:

Approximate decimal value	Distance from zero	Function
$1.08423572255 \times 10^{20}$	$2^{-58.438199858}$	Y_1
$8.67379045745 \times 10^{26}$	$2^{-58.4361612221}$	Y_1
$8.67379045745 \times 10^{26}$	$2^{-58.4361612218}$	J_0
$1.08423572255 \times 10^{20}$	$2^{-58.4356039989}$	J_0
283411312348.0	$2^{-57.4750503229}$	J_0
$6.04745635398 \times 10^{22}$	$2^{-57.2168697228}$	J_1
$6.04745635398 \times 10^{22}$	$2^{-57.216867725}$	Y_0
$2.26862308183 \times 10^{24}$	$2^{-57.1898493381}$	Y_1
$2.26862308183 \times 10^{24}$	$2^{-57.1898492858}$	J_0

Implementation

We have written a simple reference implementation for the Intel® Itanium® architecture. It exploits:

- Double-extended precision for internal calculations.
- Fused multiply-add (`fma`)

However, the same basic techniques could be used on other architectures at a certain performance cost.

Our implementation runs in 160 cycles for all finite inputs. With more aggressive scheduling we believe it should be possible to hit 100 cycles.

Outline of implementation

The implementation is based on a floating-point variant of traditional Payne-Hanek range reduction using precomputed values stored in 3 pieces:

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

for each possible input exponent a, with a runtime computation of $(P_a \cdot x)\ \texttt{mod}\ 1$.

The series for the correction term $\alpha_n(x)$ is accumulated in Horner fashion using a double-double representation.

Each double-double Horner step takes 5 `fma` latencies, but it can be pipelined to start a new step with a throughput of one step per `fma` latency.

Conclusions

- If it’s considered important, it is quite feasible to get the usual relative error for Bessel functions of fixed order with good performance.
- Our approach makes systematic use of ‘moving cancellation forwards’.
- The implementation shows that the ideas can work well, but we have not yet implemented it in a completely portable way.
- Several avenues for future work not yet explored:
  - Correctly rounded versions
  - General multiple-precision versions
  - Versions where the order is an additional argument.