A Java Math.BigDecimal Implementation of Core Mathematical Functions

Richard J. Mathar*
Leiden Observatory, Leiden University, P.O. Box 9513, 2300 RA Leiden, The Netherlands
(Dated: March 6, 2012)

The mathematical functions $\log(x)$, $\exp(x)$, $\sqrt[n]{x}$, $\sin(x)$, $\cos(x)$, $\tan(x)$, $\arcsin(x)$, $\arctan(x)$, x^y, $\sinh(x)$, $\cosh(x)$, $\tanh(x)$ and $\Gamma(x)$ have been implemented for arguments x in the real domain in a native Java library on top of the multi-precision `BigDecimal` representation of floating point numbers. This supports scientific applications where more than the double precision accuracy of the library of the Standard Edition is desired. The full source code is made available.

PACS numbers: 02.30.Gp, 02.30.Mv, 02.60.Gf
Keywords: Java, BigDecimal, mathematical functions, logarithm, exponential, trigonometric

I. OVERVIEW

A. Aim

Whereas many Java applications can use the Java Native Interface to bind to (C-based) multi-precision programs on a *host* platform when a higher precision than the 64-bit standard is needed [6], others may observe that there is only rudimentary support on the *native* platform if standard mathematical functions are needed.

The aim of this script is to provide a base implementation of the core trigonometric and algebraic functions [12] on top of the native `BigDecimal` class with infinite-precision capability. Demand originates from scientific and perhaps engineering computations, where accumulation of rounding errors (loss of digits) might pose a problem.

B. Design Choices

A characteristic feature of the implementation suggested here is that the floating point variables which are arguments to mathematical functions define by their number of digits which precision is achieved in the result. The estimate of the accuracy of the result is generally derived from the first order Taylor approximation of the function in question based on the accuracy of the input variable. The number of digits of the values returned will be larger than the number of digits of the variable where the function is flat—for example the $\arctan(x)$ where $x \gg 1$—, smaller where it is steep. This is a deliberate difference to most computer algebra systems. It provides some semi-automate detection of loss-of-precision through cancellation of digits, and it reduces some burden to the application programmer to decide on a `MathContext` interface prior to each individual call.

This is backed by some `xxxRound` functions where `xxx` are the fundamental `add`, `subtract`, `multiply`, `divide` operations, etc., which internally calculate estimators of the precision of the result based on the precisions of their arguments. The mathematical functions are typically some power series expansions, and they make heavy use of these to keep the error accumulation of the individual terms under control, that is, to chop off digits early to keep execution times short where intermediate results are known to be dominated by noise in the parameters.

As a side effect, the number of digits returned may be even smaller than the characteristic 6 digits of a single-precision calculation. In addition, the results depend on the number of trailing zeros of the inputs. (A function `scalePrec` is provided to boost the apparent accuracy of numbers by appending zeros.)

C. Known Limitations

As presented, the implementation is known to have jitters of 1 or 2 in the least significant digits in some values returned.

The algorithms have been chosen for reliability and simplicity, and may be slower than alternatives which have not been investigated.

Classes of important special functions (polynomials, Bessel functions, Elliptic Integrals,…) and complex arithmetic are absent.

II. IMPLEMENTATION STRATEGIES

A. Constants

The heavy-duty constants π, e, $\ln 2$ and γ are tabulated to high precision which presumably suffices for most purposes in engineering and sciences. The backup implementations for applications in some areas of mathematics are:

- π is evaluated by Broadhurst's equation (18) [3].
- $\log 2$ is evaluated by Broadhurst's equation (21) [3] finalized by pulling a square root.

* http://www.strw.leidenuniv.nl/~mathar; mathar@strw.leidenuniv.nl

- γ is generated by the series [5, (3.9)]

$$\gamma = 1 - \log\frac{3}{2} - \sum_{n\geq 1}\frac{\zeta(2n+1)-1}{4^n(2n+1)}. \tag{1}$$

- e is forwarded to the generic evaluation of $\exp(1)$, Section II C.

B. Roots

The roots $y = \sqrt[n]{x}$ for positive integer n, including the special case of square roots $n = 2$, are computed by iterative updates with the first order Newton method [1, (3.96)]

$$y \to y - \frac{1}{n}\left(y - \frac{x}{y^{n-1}}\right). \tag{2}$$

The initial estimates of y are set in double precision by a call to the `Math.pow`.

The `hypot` function computes

$$z = \sqrt{x^2+y^2} \tag{3}$$

from two arguments x and y. A derived case has been implemented taking an integer value x, because this implies that the precision of the result z is determined from the precision in y alone.

C. Exponential

If x is close to zero, the standard Taylor series [1, (4.2.1)]

$$\exp(x) = \sum_{k\geq 0}\frac{x^k}{k!} \tag{4}$$

is employed. For larger x, x is scaled by powers of 10

$$e^x = \left(e^{10^{-t}x}\right)^{10^t} \tag{5}$$

such that the value in parenthesis can be evaluated by recourse to (4). Scaling by powers of 10 is a cheap operation in the `BigDecimal` library because it only involves a diminuation of the scale. Powers with integer exponents are also more efficient than one might naïvely expect. The 10th power needs 4 multiplications, for example; see [10] and sequence A003313 in the OEIS [11].

The general powers are forwarded to a mixed call of the `log` and `exp` functions,

$$x^y = \exp(y\log x). \tag{6}$$

D. Logarithm

For arguments close to 1, the standard Taylor expansion [1, (4.1.24)]

$$\log(1+x) = \sum_{k\geq 1}(-1)^{j+1}\frac{x^k}{k} \tag{7}$$

is used. For larger x, adaptation to that range is achieved by scaling with some integer r

$$\log x = r\log\sqrt[r]{x} \tag{8}$$

with an auxiliary call to the `root` function of Section II B. The variable r is obtained by a call to the `Math.log` of the native library.

For some integer arguments, dedicated routines are implemented assuming that $\ln 2$ is instantly available,

$$12\ln 3 = 19\ln 2 + \sum_{k\geq 1}\frac{(-1)^{k+1}}{k}\left(\frac{7153}{524288}\right)^k, \tag{9}$$

$$6\ln 5 = 14\ln 2 - \sum_{k\geq 1}\frac{1}{k}\left(\frac{759}{16384}\right)^k, \tag{10}$$

$$\ln 7 = 3\ln 2 - \sum_{k\geq 1}\frac{1}{k8^k}. \tag{11}$$

These have practically no speed advantage compared to the alternative of adding zeros and handling them with the generic procedure described above.

E. Trigonometric

The arguments of sin, cos and tan are reduced to the fundamental domain modulo 2π or modulo π, then folded with standard shifting equations, Table 4.3.44 in the Handbook [1] into regions where the basic Taylor series converge well. These are in particular

$$\sin x = \sum_{k\geq 0}(-1)^k\frac{x^{2k+1}}{(2k+1)!}, \tag{12}$$

$$\cos x = \sum_{k\geq 0}(-1)^k\frac{x^{2k}}{(2k)!} \tag{13}$$

for $x < \pi/4$, and

$$\tan x = \sum_{k\geq 1}(-1)^{k+1}\frac{4^k(4^k-1)}{(2k)!}B_{2k}x^{2k-1} \tag{14}$$

for $x < 0.8$ [1, (4.3.67)]. The $\tan x$ is forwarded to a similar expansion of $\cot x$ [1, (4.3.70)] if $x > 0.8$.

F. Inverse Trigonometric

The arcsin is implemented as [1, (4.4.40)]

$$\arcsin x = \sum_{k\ge 0} \frac{(2k-1)!!}{(2k)!!(2k+1)} x^{2k+1} \tag{15}$$

where $x < 0.7$, and as the complementary [1, (4.4.41)] where $0.7 < x < 1$. The arctan is implemented by the standard Taylor expansions

$$\arctan x = \sum_{k\ge 0} (-1)^k \frac{x^{2k+1}}{2k+1}, \quad x < 0.7 \tag{16}$$

$$\arctan x = \frac{\pi}{2} - \sum_{k\ge 0} (-1)^k \frac{1}{(2k+1)x^{2k+1}}, \quad x > 3. \tag{17}$$

The intermediate cases are mapped to the region $x < 0.7$ by reverse application of [1, (4.4.34)]

$$2\arctan x = \arctan \frac{2x}{1-x^2} \tag{18}$$

at the cost of one additional square root.

G. Hyperbolic

The hyperbolic functions sinh and cosh are evaluated by their power series [1, (4.5.62),(4.5.63)] if the argument is close to zero, otherwise transformed by the multi-angle formulas. The tanh is implemented as

$$\tanh x = \frac{1-\exp(-2x)}{1+\exp(-2x)}. \tag{19}$$

The inverse hyperbolic functions are mapped to their logarithmic representations.

H. Gamma Function

The Γ function is reduced to the region near $x = 1$ by its functional equation

$$x\Gamma(x) = \Gamma(x+1), \tag{20}$$

and then expanded with [1, (6.1.33)]

$$\ln\Gamma(1+x) = -\ln(1+x) + x(1-\gamma) + \sum_{k\ge 2} (-1)^k \frac{\zeta(k)-1}{k} x^k. \tag{21}$$

This bypasses difficulties of regulating the errors in the Stirling formula [7, 13, 14], but needs a rather costly evaluation of the ζ function. For even indices we implement [1, (23.2.16)]

$$\zeta(2n) = \frac{(2\pi)^{2n}}{2(2n)!} |B_{2n}|, \tag{22}$$

for indices 3 or 5 the Broadhurst expansions [3], and for odd arguments ≥ 7 [4]

$$\zeta(n) = \frac{(2\pi)^n}{n-1} \sum_{k=0}^{(n+1)/2} (-1)^k (1-2k) \frac{B_{2k} B_{n+1-2k}}{(2k)!(n+1-2k)!} \\ -2\sum_{k\ge 1} \frac{1}{k^n (e^{2\pi k}-1)} \left(1 + \frac{2\pi k \epsilon_n}{1-e^{-2\pi k}}\right), \tag{23}$$

where

$$\epsilon_n = \begin{cases} 0, & n \equiv 3 \pmod 4, \\ 2/(n-1), & n \equiv 1 \pmod 4. \end{cases} \tag{24}$$

III. INTEGER CLASSES

The aim to delay rounding of rational numbers leads to the auxiliary implementation of a `Rational` data type which consists of a signed numerator and an unsigned denominator, both of the `BigInteger` type. The basic operations of multiplication, division, addition and raising to an integer power are all exact in that class, and also some integer roots if numerator and denominator are perfect powers.

The class `Bernoulli` creates a special instance of these rational numbers, the Bernoulli numbers which are helpful in (14) and (22). From a short initial table at small indices [1, (Tab 23.2)], values at larger indices are generated by a double sum [8, (1)]:

$$B_n = \sum_{k=0}^{n} \frac{1}{k} \sum_{j=0}^{k} (-1)^j j^n \binom{k}{j}. \tag{25}$$

This is augmented by a very rudimentary `Prime` class which grows dynamically, and a class `Ifactor` which represents a positive integer and its prime number factorization. The multiplicative sums-of-divisors function $\sigma_k()$ and $\varphi()$ of number theory are derived from such an intermediate prime number decomposition. The set of divisors of an integer is created as well from there by a multinomial scan of the exponents.

Numbers of the form $r\sqrt{d}$, where r is a signed rational number and d a non-negative rational number of the `Rational` class, are represented by the `BigSurd` class. Sums of these are represented by `BigSurdVec`, for which addition, subtraction and exponentiation with integer exponents can be represented exactly.

IV. APPLICATION: WIGNER $3n$-j SYMBOLS

As an independent test of of other programs that evaluate Wigner 3n-j symbols [9], footed on the exact representation of square roots presented above, the class `Wigner3j` allows computation of 3n-j symbols for unlimited n. The application interface has been modeled according to an earlier proposal [2]. One line is the integer $3n$. Two lines of integers are 1-based indices into a list of j-values, implicitly bundled in triads such that for each of the $2n$ factors of the underlying cubic graph the three contributing j-values are listed in the order of appearance in the upper row of their $3jm$ values. These indices are negated if the associated m-value in the $3jm$ symbol appears with a negative sign. All further lines of the input contain lists of j values by actually providing the positive, integer-valued $2j+1$.

The implementation does not use any reduction techniques. It performs the summation over all m-values of all angular momenta that have been defined, and is slow for that reason. (There is some look-ahead for small cycles in the associated cubic graph to take advantage of the selection rule of $3jm$-values.)

The class `Wigner3jGUI` is an online calculator for these. The connectivity schemes for all symbols from $6j$ up of $15j$ (in a serialized order reading the entries by rows in the braces of the standard notation) have been initialized and are selectable with a button.

V. SUMMARY

The most frequently used mathematical function with arguments and return values of the multi-precision `BigDecimal` type are presented in a Java library. Control over the variable requirements in precision is basically achieved by recourse to simple algorithms that allow semi-analytic estimations of the propagation of errors.

[1] Abramowitz, M., and I. A. Stegun (eds.), 1972, *Handbook of Mathematical Functions* (Dover Publications, New York), 9th edition, ISBN 0-486-61272-4.

[2] Bar-Shalom, A., and M. Klapisch, 1988, Comp. Phys. Commun. **50**(3), 375.

[3] Broadhurst, D. J., 1998, arXiv:math/9803067 [math.CA] .

[4] Cohen, H., L. Lewin, and D. Zagier, 1992, Exp. Math **1**(1), 25.

[5] Dilcher, K., 2005, Aequat. Math. **48**(1), 55.

[6] Fousse, L., G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann, 2007, ACM Trans. Math. Softw. **33**(2), # 13.

[7] Gordon, N. L., and A. H. Flasterstein, 1960, J. ACM **7**(4), 387.

[8] Gould, H. W., 1972, Amer. Math. Monthly **79**(1), 44.

[9] Mathar, R. J., 2011, arXiv:1102.5125 [math-ph] .

[10] McCarthy, D. P., 1986, Math. Comp. **46**(174), 603.

[11] Sloane, N. J. A., 2003, Notices Am. Math. Soc. **50**(8), 912, http://oeis.org/, URL `http://oeis.org/`.

[12] Smith, D. M., 1991, ACM Trans. Math. Softw. **17**(2), 273.

[13] Spira, R., 1971, Math. Comp. **25**(114), 317.

[14] Wrench Jr., J. W., 1968, Math. Comput. **22**(103), 617.