Algorithm: Fortran 90 Software for Floating-Point Multiple Precision Arithmetic, Gamma and Related Functions

David M. Smith

Loyola Marymount University

A collection of Fortran-90 routines for evaluating the Gamma function and related functions using the FM multiple-precision arithmetic package.

Categories and Subject Descriptors: G.1.0 [**Numerical Analysis**]: General – *computer arithmetic;* G.1.2 [**Numerical Analysis**]: Approximation – *special function approximation;* G.4 [**Mathematics of Computing**]: Mathematical Software – *Algorithm analysis, efficiency, portability*

General Terms: Algorithms, Gamma function, multiple precision

Additional Key Words and Phrases: Accuracy, function evaluation, floating point, Fortran, mathematical library, portable software

---

1. INTRODUCTION

The FMLIB package of Fortran subroutines for floating-point multiple-precision computation now includes routines to evaluate the Gamma function and related functions. These routines use the basic FM operations and derived types [Smith 1991; 1998] for multiple-precision arithmetic, constants, and elementary functions. The new functions available are essentially those in the chapter on the Gamma function in a reference such as Abramowitz and Stegun [1965].

The FM routines almost always return correctly rounded results. Extensive testing has found no cases where the error before rounding the final value was more than 0.001 unit in the last place of the returned result. This means that in rare cases the returned result differed from the correctly rounded value by a maximum of one unit in the last place for a given precision. Most of these routines gain speed by storing constants such as Bernoulli numbers and Euler's constant so they do not have to be computed again on subsequent calls.

In addition to subroutines for Gamma and the other functions, there are also functions that provide an interface to the derived-type floating-point multiple-precision numbers defined in the FM package [Smith 1998]. The basic FM package has undergone several changes. A few new routines have been added, and the code has been translated to free-format Fortran-90. There is a ReadMe

file provided that lists all available operations and contains a user's manual and troubleshooting guide for the package.

2. EFFICIENCY

Many of the functions in this package require Bernoulli numbers and other constants such as π and e. Since the routines compute constants only when they are needed at higher precision than previously used, the first call to a function may be slower than later calls. As precision increases, the number of Bernoulli numbers required is roughly proportional to the precision. For example, using 30 digits in base 10^7 (about 200 significant digits) to compute $\Gamma(x)$ it takes about three or four times as long for the first call as for a later call.

For t digits, the theoretical complexity of these functions is greater than the $O(t^{2.33})$ complexity for the elementary functions [Smith 1989]. However, for precision up to 200 decimal digits the ratio of running times for $\Gamma(x)$ to times for $\tan^{-1}(x)$ varies only from about four to five as t increases. This indicates that unless precision is quite large these special functions are roughly a constant factor slower than the elementary functions.

The Beta and Binomial Coefficient functions are about three times slower than Gamma. The Incomplete Gamma and Incomplete Beta functions are often slower than Gamma, but their timing depends more on the input arguments.

3. SOME ALGORITHMS USED IN THE PACKAGE

3.1 Bernoulli Numbers

Since the Bernoulli numbers usually occur as coefficients while summing series, this routine returns the product of an FM number and a Bernoulli number. The often-used values B_1 through B_{26} are stored in rational form as quotients of 32-bit integers. This saves some space in keeping the previously computed values, and allows the product to be done using one multiply by a one-word integer and one division by a one-word integer. These are both $O(t)$ operations, so it is faster than the $O(t^2)$ multiplication by a stored quotient. For B_n with n greater than 26, the numerator is too large to store as one 32-bit word, so B_n is stored as one FM number.

To compute B_n for n larger than 26, FM uses recurrence relations that follow from some identities of Ramanujan [1911]. One of these equations is

$$3\left(B_0 + B_6\frac{x^6}{6!} + B_{12}\frac{x^{12}}{12!} + \cdots\right) = -x\,\frac{\frac{x^2}{2!} - \frac{x^8}{8!} + \frac{x^{14}}{14!} - \cdots}{\frac{x^3}{3!} - \frac{x^9}{9!} + \frac{x^{15}}{15!} - \cdots}\,.$$

Ramanujan defined the Bernoulli numbers so that B_n was nonnegative when $n \geq 2$. FM follows the convention of Abramowitz and Stegun [1965] that B_1, B_4, B_8, $B_{12}\ldots$ are negative.

Changing to this notation and simplifying the equation above leads to the following recurrence when n is a multiple of 6:

$$\binom{n+3}{n} B_n = -\left[\binom{n+3}{n-6} B_{n-6} + \binom{n+3}{n-12} B_{n-12} + \cdots + \binom{n+3}{0} B_0\right] + \frac{n+3}{3} .$$

For $n = 6k+2$ the equation is

$$\binom{n+3}{n} B_n = -\left[\binom{n+3}{n-6} B_{n-6} + \binom{n+3}{n-12} B_{n-12} + \cdots + \binom{n+3}{2} B_2\right] + \frac{n+3}{3} ,$$

and $n = 6k+4$ gives

$$\binom{n+3}{n} B_n = -\left[\binom{n+3}{n-6} B_{n-6} + \binom{n+3}{n-12} B_{n-12} + \cdots + \binom{n+3}{4} B_4\right] - \frac{n+3}{6} .$$

If $M(t)$ denotes the time required for a t-digit multiplication, the algorithm used in Brent's MP [1978] for computing all the Bernoulli numbers up to B_n takes $O(n^2 t + nM(t))$ time, while the FM algorithm is $O(n^2 M(t))$. Thus the MP method is asymptotically more efficient, but the FM version is faster for the range of n needed in computation of the special functions.

3.2 Gamma and Log Gamma Functions

$$\Gamma(x) = \int_0^\infty t^{x-1}\, e^{-t}\, dt .$$

$\ln(\Gamma(x))$ is evaluated using Stirling's approximation unless x is a small integer.

$$\ln(\Gamma(x)) \sim (x - \tfrac{1}{2}) \ln(x) - x + \tfrac{1}{2} \ln(2\pi) + \sum_{n=1}^{\infty} \frac{B_{2n}}{2n\,(2n-1)\,x^{2n-1}} .$$

If x is negative the reflection formula is used, and x is replaced by $1 - x$.

Since this asymptotic series diverges, t digits of precision can be attained only for x greater than a threshold value, x_t. The input value can be shifted by any j for which $x + j > x_t$ and then the series can be summed. To improve the efficiency of the routine, an integer j is chosen with $x + j > x_t$, so that the total computation time is roughly minimized. This balances the faster summing of the series for larger j against the slower reversal of the argument shifting.

So $\ln(\Gamma(x))$ is found using the equation $\Gamma(x+j) = (x+j-1)(x+j-2)\cdots(x)\Gamma(x)$. If the original x was negative, then $\Gamma(x)\Gamma(1-x) = \pi \csc(\pi x)$ is used to reverse the initial reflection. Stirling's series is summed using the decreasing precision and concurrent sum techniques discussed in [Smith 1991].

$\Gamma(x)$ is done using $\ln(\Gamma(x))$ and e^x. The most recent argument and result of this routine are saved. If the argument used in the next call differs by a small integer from the previous argument, then $\Gamma(x)$ can be found quickly using the recurrence and the saved result. When the argument is a

small integer the result is obtained using multiplication, with much of the computation done with one-word integers.

For example, to get 60 significant digit accuracy for $\Gamma(0.5)$ the routine uses $n = 37$ terms in the log gamma expansion after choosing $j = 32$ and shifting the argument to $x + j = 32.5$. After B_{26}, the routine switches to using the full FM multiply routine for multiplying the Bernoulli numbers while computing the next term in the series.

$x!$ is computed using $x! = \Gamma(x + 1)$, so the time for the factorial function is essentially the same as Gamma.

3.3 Pochhammer's Symbol

$$(x)_n = x(x + 1)(x + 2) \cdots (x + n - 1) = \frac{\Gamma(x + n)}{\Gamma(x)} \ .$$

This calculation occurs in the Gamma and Log Gamma functions, and similar calculations occur in several other functions. In the case of Gamma, $\Gamma(x+n)$ is computed instead of $\Gamma(x)$ to increase the accuracy of the asymptotic expansion and to speed up convergence in the series. Then the result is divided by $(x)_n$ to recover $\Gamma(x)$.

As precision increases, multiplication becomes slow compared with addition, subtraction, or multiplication or division by a one-word integer. FM groups four terms together and updates the product of the next four terms from the previous four. This computes $(x)_n$ in about $n/4$ full multiplications and $7n/4$ faster operations.

If one group of four terms is $(x + k)(x + k + 1)(x + k + 2)(x + k + 3)$, then the next group is $(x + k + 4)(x + k + 5)(x + k + 6)(x + k + 7)$. The difference between these two groups is $16x^3 + 24(2k + 7)x^2 + 8(6k^2 + 42k + 79)x + 8(2k + 7)(k^2 + 7k + 15)$. After the terms x^2 and $16x^3$ are computed once before starting the product, the product of the next four terms can be obtained with two multiplications by small integers, one integer-to-FM conversion, and four additions. Then one full multiplication accumulates the product of the next four terms.

If k becomes so large that some of the coefficients in the cubic polynomial for this difference cannot be expressed as one word integers, then the terms are built up using a few more $O(t)$ operations.

When n is large enough FM computes $(x)_n$ with two Log Gamma calls, since this is faster than multiplying. For large values of x all the terms in the product are identical to working precision, so $(x)_n = x^n$.

3.4 Psi (Digamma) Function and Polygamma Functions

$$\psi(x) = \frac{d}{dx} \ \ln(\Gamma(x)) = \frac{\Gamma\,'(x)}{\Gamma(x)} \ .$$

Psi is computed from the asymptotic series

$$\psi(x) \sim \ln(x) - \frac{1}{2x} - \sum_{n=1}^{\infty} \frac{B_{2n}}{2n\, x^{2n}} \ .$$

This algorithm is very similar to that of $\ln(\Gamma(x))$. The reflection formula used for negative arguments is $\psi(1-x) = \psi(x) + \pi\, \cot(\pi\, x)$. For recovering the original function value after summing the series using a shifted argument the formula is

$$\psi(x+j) = \frac{1}{x+j-1} + \frac{1}{x+j-2} + \cdots + \frac{1}{x} + \psi(x) \ .$$

These terms are grouped in sets of four as with Pochhammer's symbol to reduce the number of $O(t^2)$ operations.

The Polygamma functions are the derivatives of Psi:

$$\psi^{(k)}(x) = \left(\frac{d}{dx}\right)^k \psi(x) = \left(\frac{d}{dx}\right)^{k+1} \ln(\Gamma(x))$$

$$\sim (-1)^{k-1} \left(\frac{(k-1)!}{x^k} + \frac{k!}{2\, x^{k+1}} + \sum_{n=1}^{\infty} \frac{B_{2n}\, (2n+k-1)!}{(2n)!\, x^{2n+k}} \right) \ .$$

Both the recurrence relation and the reflection formula are slightly more complicated than with Psi.

$$\psi^{(k)}(x+j) = (-1)^k\, k! \left(\frac{1}{(x+j-1)^{k+1}} + \frac{1}{(x+j-2)^{k+1}} + \cdots + \frac{1}{x^{k+1}} \right) + \psi^{(k)}(x) \ ,$$

$$\psi^{(k)}(1-x) = (-1)^k\, \psi^{(k)}(x) + (-1)^k\, \pi \left(\frac{d}{dx}\right)^k \cot(\pi\, x) \ .$$

To compute the derivatives of $\cot(\pi\, x)$ a recurrence relation is used to express the k^{th} derivative as a polynomial of degree $k+1$ with powers of $\cot(\pi\, x)$. For $k \le 14$ the coefficients needed to evaluate these polynomials are stored in the program as 32-bit integers. For larger values of k the coefficients are kept as FM numbers and the recurrence relation is used to generate the polynomials from degree 15 through $k+1$.

The time for Polygamma increases fairly slowly with increasing k. For precision about 40 digits and small positive x, the time is about twice the $k = 1$ value when $k = 30$. For small negative x it takes about three times as long when $k = 30$.

3.5 Euler's Constant

When the precision is not too high, Euler's constant is obtained from input conversion of a character string containing the value of the constant. When the arithmetic base being used in

FM is a power of ten (the usual case), this returns γ in $O(t)$ time. Like the other commonly used constants, γ is stored at the highest precision previously obtained. If the current precision is greater than that of both the string value and the saved value, then the Psi function is used: $\gamma = -\psi(1) = 0.5772\ldots$.

3.6 Beta Function and Binomial Coefficients

Each of these functions can be computed using $x!$ or $\Gamma(x)$:

$$B(x,y) = \int_0^1 t^{x-1}(1-t)^{y-1}\,dt = \frac{\Gamma(x)\,\Gamma(y)}{\Gamma(x+y)}\ ,$$

$$\binom{x}{y} = \frac{x!}{y!\,(x-y)!}\ .$$

Using the Gamma function, these routines are about three times slower than $\Gamma(x)$ unless an argument is a small integer or x and y differ by a small integer. Both functions handle small integer cases by essentially cancelling some terms and multiplying the remaining terms. This is often much faster since it avoids calling the Gamma function. Because the Gamma function saves its last result, these functions may also be faster if one of the arguments differs by a small integer from the argument for the previous call to $\Gamma(x)$.

There are several special cases in which the formulas above cannot be used directly. In cases where the Gamma function would overflow, the Log Gamma function is used to compute the logarithm of the desired result and then the final answer is obtained from the exponential. This usually requires increasing the number of guard digits being used to compensate for cancellation error.

When one argument, say x, is much larger than the other, y, there is too much cancellation in $\ln\Gamma(x) - \ln\Gamma(x+y)$. Stirling's formula is used in many such cases to simplify this difference and avoid the cancellation, giving $B(x,y) = \exp(\ln\Gamma(y) - y\,\ln(x+y))$.

Another special case involves negative integer arguments in the binomial coefficient function. When any of the three values are negative integers, FM uses the definitions in Graham, Knuth, and Patashnik [1989]. These lead to results such as $\binom{-10}{3} = -220$ and $\binom{-4}{-4} = 0$, which are useful in some combinatorial applications. However, many standard identities such as $\binom{x}{y} = \binom{x}{x-y}$ and $\binom{x}{x} = 1$ may not be true when negative integers are present.

3.7 Incomplete Gamma Function

There are several functions that are closely related to the incomplete gamma function. FM provides two from which the others can be found:

$$\gamma(a,x) = \int_0^x e^{-t}\, t^{a-1}\, dt,$$

$$\Gamma(a,x) = \int_x^\infty e^{-t}\, t^{a-1}\, dt.$$

The routine for $\gamma(a,x)$ uses one of three methods, depending on the current base, precision, and the values of a and x. The first method uses a convergent series, the second uses an asymptotic series, and the third uses a continued fraction expansion.

$$\begin{aligned}
\gamma(a,x) &= \frac{x^a}{a\, e^x} \sum_{n=0}^{\infty} \frac{x^n}{(a+1)_n} \\
&\sim \Gamma(a) - \frac{x^{a-1}}{e^x} \sum_{n=0}^{\infty} \frac{(a-n)_n}{x^n} \\
&= \Gamma(a) - \frac{x^a}{e^x} \left(\frac{1}{x+} \quad \frac{1-a}{1+} \quad \frac{1}{x+} \quad \frac{2-a}{1+} \quad \frac{2}{x+} \ \cdots \right).
\end{aligned}$$

Since $\Gamma(a,x) = \Gamma(a) - \gamma(a,x)$, these give three corresponding methods for $\Gamma(a,x)$. Because of cancellation error or poor rates of convergence, none of these methods work well when a is very close to zero and x is not too far from zero. The routine for $\Gamma(a,x)$ uses a fourth approximation in this case:

$$\Gamma(a,x) = -\ln(x) - \frac{\ln^2(x)\, a}{2} - \gamma + \left(\frac{\gamma^2}{2} + \frac{\pi^2}{12} \right) a - \sum_{n=1}^{\infty} \frac{(-1)^n\, x^{a+n}}{(a+n)\, n!} + O(a^2).$$

When a is a small negative integer this formula is used with the recurrence

$$\Gamma(-n,x) = \frac{(-1)^n}{n!} \left(\Gamma(0,x) - e^{-x} \sum_{j=0}^{n-1} \frac{(-1)^j j!}{x^{j+1}} \right).$$

The computation becomes slower for these functions when a and x are large and nearly equal. It takes about four times longer (using 40 significant digit precision) when a and x are around 10,000 than when they are near one.

3.8 Incomplete Beta Function

$$B_x(a,b) = \int_0^x t^{a-1}\,(1-t)^{b-1}\,dt\ .$$

The logic for selecting the method to be used in $B_x(a,b)$ is fairly complicated. The number of different cases is comparable to that of the carefully crafted subroutine `BRATIO` of Didonato and Morris [1992]. `BRATIO` computes $B_x(a,b)/B(a,b)$ for machine-precision numbers.

The exact list of methods used and the regions where they apply are different in FM than in `BRATIO`, because FM can dynamically raise precision during the calculation. It is sometimes more efficient in FM to raise precision in anticipation of losing 20 or 30 digits to cancellation in a faster but less stable method.

The routine for $B_x(a,b)$ primarily uses one of six methods, depending on the current base, precision, and the values of a, b, and x. The first method uses a convergent series, the second uses a continued fraction, and the third uses a more recent continued fraction [Didonato and Morris 1992]. The other three methods use the symmetry relation $B_x(a,b) = B(a,b) - B_{1-x}(b,a)$ followed by one of the first three.

$$B_x(a,b) = x^a \sum_{n=0}^{\infty} \frac{(1-b)_n\ x^n}{(a+n)\ n!}$$

$$= \frac{x^a\ (1-x)^b}{a} \left(\frac{1}{1+} \quad \frac{d_1}{1+} \quad \frac{d_2}{1+} \ \cdots \right)$$

$$= x^a\ (1-x)^b \left(\frac{\alpha_1}{\beta_1+} \quad \frac{\alpha_2}{\beta_2+} \quad \frac{\alpha_3}{\beta_3+} \ \cdots \right),$$

where

$$d_{2m+1} = \frac{-(a+m)(a+b+m)x}{(a+2m)(a+2m+1)}\ , \qquad d_{2m} = \frac{m(b-m)x}{(a+2m-1)(a+2m)}\ ,$$

$$\alpha_1 = 1, \qquad \alpha_{m+1} = \frac{(a+m-1)(a+b+m-1)m(b-m)x^2}{(a+2m-1)^2}\ , \quad m \geq 1\ ,$$

$$\beta_{m+1} = m + \frac{m(b-m)x}{a+2m-1} + \frac{(a+m)(a-(a+b)x+1+m(2-x))}{a+2m+1}\ , \quad m \geq 0\ .$$

In addition to these, there are several special case methods to handle rare situations where the methods above are unstable or slow.

Translation formulas are used in some methods to improve convergence and to avoid regions of instability. They can also be useful in testing. DiDonato and Morris use

$$t^{a-1}(1-t)^{b-1} = \frac{1}{a}\left(\frac{d}{dt}\ \left[t^a(1-t)^b\right] + (a+b)\,t^a\,(1-t)^{b-1} \right)$$

to get the translation formula

$$B_x(a,b) = \frac{1}{a}\, x^a\, (1-x)^b + \frac{a+b}{a}\, B_x(a+1,b).$$

Integrating by parts in the definition of $B_x(a,b)$ gives a diagonal step:

$$B_x(a,b) = -\frac{1}{b}\, x^{a-1}\, (1-x)^b + \frac{a-1}{b}\, B_x(a-1,b+1),$$

and combining these two formulas gives a step in the b direction:

$$B_x(a,b) = -\frac{1}{b}\, x^a\, (1-x)^b + \frac{a+b}{b}\, B_x(a,b+1).$$

For cases where several steps are needed, these formulas give

$$B_x(a,b) = \frac{x^a\,(1-x)^b}{a} \sum_{n=0}^{k-1} \frac{(a+b)_n}{(a+1)_n}\, x^n \;+\; \frac{(a+b)_k}{(a)_k}\, B_x(a+k,b),$$

$$B_x(a,b) = -\frac{x^a\,(1-x)^b}{b} \sum_{n=0}^{k-1} \frac{(a+b)_n}{(b+1)_n}\, (1-x)^n \;+\; \frac{(a+b)_k}{(b)_k}\, B_x(a,b+k).$$

For small b, two of the formulas used are similar to those in functions `FPSER` and `APSER` used by the subroutine `BRATIO` of Didonato and Morris. The first is used when $x \le 1/2$,

$$\begin{aligned} B_x(a,b) &= B_x(a,b+1) - x^a\, \ln(1-x) - a\, x^a \sum_{n=1}^{\infty} \frac{x^n}{n(a+n)} \;+\; O(b) \\ &= x^a \left(\frac{1}{a} - \ln(1-x) - a \sum_{n=1}^{\infty} \frac{x^n}{n(a+n)} \right) \;+\; O(b). \\ &= x^a \left(\frac{1}{a} \;+\; \sum_{n=1}^{\infty} \frac{x^n}{a+n} \right) \;+\; O(b). \end{aligned}$$

The second is used when $x > 1/2$ and $a < 20$,

$$\begin{aligned} B_x(a,b) &= B(a,b) - B_{1-x}(b,a) \\ &= \frac{1}{b} + \frac{1}{a} - \gamma - \psi(a+1) - B_{1-x}(b,a) \;+\; O(b) \\ &= \frac{1}{a} - \ln(1-x) - \gamma - \psi(a+1) - (1-x)^b \sum_{n=1}^{\infty} \frac{(1-a)_n\,(1-x)^n}{n\,n!} \;+\; O(b). \end{aligned}$$

A third method for small b, used when $x > 1/2$ and $a \ge 20$, is an asymptotic series derived from the one in Didonato and Morris' `BGRAT` routine.

$$\begin{aligned} B_x(a,b) &\sim x^T (-\ln(x))^b \sum_{n=0}^{\infty} p_n\, J_n(b,u) \;+\; O(b)\,, \quad \text{where} \\ T &= a - 1/2\,, \\ u &= -T\, \ln(x). \end{aligned}$$

The $J_n(b,u)$ terms are computed from a recurrence,

$$J_0(b,u) = \frac{\Gamma(b)\,\Gamma(b,u)}{x^T\,u^b}\,,$$
$$J_{n+1}(b,u) = \frac{2n(2n+1)}{4\,T^2}\,J_n(b,u) + \frac{u+2n+1}{4\,T^2}\,\left(\frac{\ln(x)}{2}\right)^{2n}$$

In `BGRAT`, p_n is the nth nonzero coefficient of the Maclauren series for $(\sinh(x)/x)^{b-1}$, and is also found using a recurrence,

$$p_0 = 1\,,$$
$$p_n = \frac{(b-1)}{(2n+1)!} + \frac{1}{n}\sum_{m=1}^{n-1}\frac{m\,b-n}{(2m+1)!}\,p_{n-m}.$$

This formula is not used in FM, to avoid having to store a large number of p_n values. When FM uses the asymptotic series, b is small enough so that $b-1=-1$ at the precision being used. Then

$$\left(\frac{\sinh(x)}{x}\right)^{b-1} = x\,\mathrm{csch}(x) \;+\; O(b) = 1 - \sum_{n=1}^{\infty}\frac{(2^{2n}-2)\,B_{2n}}{(2n)!}\,x^{2n} \;+\; O(b).$$

This gives a formula for p_n involving Bernoulli numbers. Since the B_n values are already saved by several of the other functions in FM, the formula used for small n is

$$p_n = \frac{2-2^{2n}}{(2n)!}\,B_{2n}.$$

For larger n, a formula relating B_n to the Riemann Zeta function is used.

$$|B_{2n}| = \frac{2\,(2n)!}{(2\pi)^{2n}}\,\zeta(2n) = \frac{2\,(2n)!}{(2\pi)^{2n}}\prod_{\substack{P_i \text{ is}\\ \text{prime}}}\frac{P_i^{2n}}{P_i^{2n}-1}.$$

When n becomes large enough so that only a few terms are needed in the product, p_n is updated using the formula

$$p_n = \frac{-p_{n-1}}{\pi^2}\,\left(\frac{2^{2n}-2}{2^{2n}-8}\right)\prod_{\substack{P_i \text{ is}\\ \text{prime}}}\left(1-\frac{P_i^2-1}{P_i^{2n}-1}\right).$$

Other special cases involve detecting input values for which $B_x(a,b) = B(a,b)$ at the current precision and then using the Beta routine.

4. TESTING

To test the accuracy of the functions in this package many random arguments were generated and results compared at a sequence of increasing precisions. Many tests were also done at various

boundaries between regions where the FM routine uses different methods. At the boundary the two different methods should agree.

As independent tests for the functions, FM results were compared to values tabulated in [Abramowitz and Stegun 1965] and to values generated by MP [Brent 1978], routine `BGRAT` [DiDonato and Morris 1992], and the Mathematica computer algebra system [Wolfram 1988].

References

1. Abramowitz, M., and Stegun, I.A. (Eds.) 1965. *Handbook of Mathematical Functions*, Dover, New York.

2. Brent, R.P. 1978. A Fortran Multiple-Precision Arithmetic Package. *ACM Trans. Math. Softw. 4*, 1 (March), 57–70.

3. DiDonato, A.R., and Morris, Jr., A.H. 1992. Significant Digit Computation of the Incomplete Beta Function Ratios. *ACM Trans. Math. Softw. 18*, 3 (September), 360–373.

4. Graham, R., Knuth, D., and Patashnik, O. 1989. *Concrete Mathematics*, Addison-Wesley, Redwood City, Calif.

5. Ramanujan, S. 1911. Some Properties of Bernoulli's Numbers. *Journal Indian Math. Soc.* 3 (1911), 219-234. Also in *Collected papers of Srinivasa Ramanujan*, G.H. Hardy, P.V. Seshu Aiyar, and B.M. Wilson, Eds., Cambridge University Press, 1927, pp. 1–14.

6. Smith, D.M. 1989. Efficient Multiple-Precision Evaluation of Elementary Functions. *Math. Comp.* 52 (January) 131–134.

7. Smith, D.M. 1991. A Fortran Package for Floating-Point Multiple-Precision Arithmetic. *ACM Trans. Math. Softw. 17*, 2 (June), 273–283.

8. Smith, D.M. 1998. Multiple Precision Complex Arithmetic and Functions. *ACM Trans. Math. Softw. 24*, 4 (December), 359–367.

9. Wolfram, S. 1988. *Mathematica: A System for Doing Mathematics by Computer*, Addison-Wesley, Redwood City, Calif.