Cephes Mathematical Library

[Up to home page](#):

[Get 128bit.tgz](#):

Documentation for single precision functions.
Documentation for double precision functions.
Documentation for 80-bit long double functions.
Documentation for 128-bit long double functions.
Documentation for extended precision functions.

128-bit Long Double Precision Functions

Select function name for additional information. For other precisions, see the archives and descriptions listed above.


```
/*                                                      acoshl.c
 *
 *      Inverse hyperbolic cosine, 128-bit long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, acoshl();
 *
 * y = acoshl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns inverse hyperbolic cosine of argument.
```

```
 *
 * If 1 <= x < 1.5, a rational approximation
 *
 *      sqrt(2z) * P(z)/Q(z)
 *
 * where z = x-1, is used.  Otherwise,
 *
 * acosh(x)  =  log( x + sqrt( (x-1)(x+1) ).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      1,3        100,000      4.1e-34     7.3e-35
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * acoshl domain      |x| < 1            0.0
 *
 */



/*                                                      asinhl.c
 *
 *      Inverse hyperbolic sine, 128-bit long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, asinhl();
 *
 * y = asinhl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns inverse hyperbolic sine of argument.
 *
 * If |x| < 0.5, the function is approximated by a rational
 * form  x + x**3 P(x)/Q(x).  Otherwise,
 *
 *     asinh(x) = log( x + sqrt(1 + x*x) ).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     -2,2         100,000     2.8e-34     6.7e-35
 *
 */



/*                                                      asinl.c
 *
 *      Inverse circular sine, 128-bit long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, asinl();
 *
 * y = asinl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns radian angle between -pi/2 and +pi/2 whose sine is x.
 *
 * A rational function of the form x + x**3 P(x**2)/Q(x**2)
 * is used for |x| in the interval [0, 0.5].  If |x| > 0.5 it is
 * transformed by the identity
 *
 *    asin(x) = pi/2 - 2 asin( sqrt( (1-x)/2 ) ).
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     -1, 1       100,000      3.7e-34      6.4e-35
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * asin domain        |x| > 1           0.0
 *
 */
```

```
/*                                                      acosl()
 *
 *      Inverse circular cosine, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, acosl();
 *
 * y = acosl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns radian angle between -pi/2 and +pi/2 whose cosine
 * is x.
 *
 * Analytically, acos(x) = pi/2 - asin(x).  However if |x| is
 * near 1, there is cancellation error in subtracting asin(x)
 * from pi/2.  Hence if x < -0.5,
 *
 *    acos(x) =  pi - 2.0 * asin( sqrt((1+x)/2) );
 *
 * or if x > +0.5,
 *
 *    acos(x) =  2.0 * asin(  sqrt((1-x)/2) ).
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     -1, 1       100,000      2.1e-34      5.6e-35
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * asin domain        |x| > 1           0.0
 */



/*                                                      atanhl.c
 *
 *      Inverse hyperbolic tangent, 128-bit long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, atanhl();
 *
 * y = atanhl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns inverse hyperbolic tangent of argument in the range
 * MINLOGL to MAXLOGL.
 *
 * If |x| < 0.5, the rational form x + x**3 P(x)/Q(x) is
 * employed.  Otherwise,
 *        atanh(x) = 0.5 * log( (1+x)/(1-x) ).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -1,1       100,000      2.0e-34     4.6e-35
 *
 */



/*                                                      atanl.c
 *
 *      Inverse circular tangent, 128-bit long double precision
 *      (arctangent)
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, atanl();
 *
 * y = atanl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns radian angle between -pi/2 and +pi/2 whose tangent
```

```
 * is x.
 *
 * Range reduction is from four intervals into the interval
 * from zero to  tan( pi/8 ).  The approximant uses a rational
 * function of degree 3/4 of the form x + x**3 P(x)/Q(x).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10, 10    100,000      2.6e-34     6.5e-35
 *
 */



/*                                                      atan2l()
 *
 *      Quadrant correct inverse circular tangent,
 *      long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, z, atan2l();
 *
 * z = atan2l( y, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns radian angle whose tangent is y/x.
 * Define compile time symbol ANSIC = 1 for ANSI standard,
 * range -PI < z <= +PI, args (y,x); else ANSIC = 0 for range
 * 0 to 2PI, args (x,y).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10, 10    100,000      3.2e-34      5.9e-35
 * See atan.c.
 *
 */



/*                                                      cbrtl.c
 *
 *      Cube root, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, cbrtl();
 *
 * y = cbrtl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the cube root of the argument, which may be negative.
 *
 * Range reduction involves determining the power of 2 of
 * the argument.  A polynomial of degree 2 applied to the
 * mantissa, and multiplication by the cube root of 1, 2, or 4
 * approximates the root to within about 0.1%.  Then Newton's
 * iteration is used three times to converge to an accurate
 * result.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     .125,8        80000      1.2e-34     3.8e-35
 *    IEEE    exp(+-707)    100000      1.3e-34     4.3e-35
 *
 */



/*                                                      coshl.c
 *
 *      Hyperbolic cosine, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, coshl();
```

```
 *
 * y = coshl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns hyperbolic cosine of argument in the range MINLOGL to
 * MAXLOGL.
 *
 * cosh(x)  =  ( exp(x) + exp(-x) )/2.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     +-10000      26,000       2.5e-34     8.6e-35
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * cosh overflow    |x| > MAXLOGL       MAXNUML
 *
 *
 */



/*                                                      exp10l.c
 *
 *      Base 10 exponential function, long double precision
 *      (Common antilogarithm)
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, exp10l()
 *
 * y = exp10l( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns 10 raised to the x power.
 *
 * Range reduction is accomplished by expressing the argument
 * as 10**x = 2**n 10**f, with |f| < 0.5 log10(2).
 * The Pade' form
 *
 *     1 + 2x P(x**2)/( Q(x**2) - P(x**2) )
 *
 * is used to approximate 10**f.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      +-4900     100,000      2.1e-34     4.7e-35
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * exp10l underflow    x < -MAXL10        0.0
 * exp10l overflow     x > MAXL10       MAXNUM
 *
 * IEEE arithmetic: MAXL10 = 4932.0754489586679023819
 *
 */



/*                                                      exp2l.c
 *
 *      Base 2 exponential function, 128-bit long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, exp2l();
 *
 * y = exp2l( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns 2 raised to the x power.
 *
 * Range reduction is accomplished by separating the argument
 * into an integer k and fraction f such that
 *     x    k  f
 *    2  = 2  2.
```

```
 *
 * A Pade' form
 *
 *   1 + 2x P(x**2) / (Q(x**2) - x P(x**2) )
 *
 * approximates 2**x in the basic range [-0.5, 0.5].
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      +-16300    100,000      2.0e-34     4.8e-35
 *
 *
 * See exp.c for comments on error amplification.
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * exp2l underflow   x < -16382        0.0
 * exp2l overflow    x >= 16384       MAXNUM
 *
 */



/*                                                       expl.c
 *
 *      Exponential function, 128-bit long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, expl();
 *
 * y = expl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns e (2.71828...) raised to the x power.
 *
 * Range reduction is accomplished by separating the argument
 * into an integer k and fraction f such that
 *
 *     x    k  f
 *    e  = 2  e.
 *
 * A Pade' form of degree 2/3 is used to approximate exp(f) - 1
 * in the basic range [-0.5 ln 2, 0.5 ln 2].
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      +-MAXLOG    100,000     2.6e-34     8.6e-35
 *
 *
 * Error amplification in the exponential function can be
 * a serious matter.  The error propagation involves
 * exp( X(1+delta) ) = exp(X) ( 1 + X*delta + ... ),
 * which shows that a 1 lsb error in representing X produces
 * a relative error of X times 1 lsb in the function.
 * While the routine gives an accurate result for arguments
 * that are exactly represented by a long double precision
 * computer number, the result contains amplified roundoff
 * error for large arguments not exactly represented.
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * exp underflow    x < MINLOG         0.0
 * exp overflow     x > MAXLOG         MAXNUM
 *
 */



/*                                                       expm1l.c
 *
 *      Exponential function, minus 1
 *      128-bit long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, expm1l();
 *
 * y = expm1l( x );
 *
 *
 *
 * DESCRIPTION:
```

```
 *
 * Returns e (2.71828...) raised to the x power, minus 1.
 *
 * Range reduction is accomplished by separating the argument
 * into an integer k and fraction f such that
 *
 *     x    k  f
 *    e  = 2  e.
 *
 * An expansion x + .5 x^2 + x^3 R(x) approximates exp(f) - 1
 * in the basic range [-0.5 ln 2, 0.5 ln 2].
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE    -79,+MAXLOG    100,000     1.7e-34     4.5e-35
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * expm1 overflow    x > MAXLOG         MAXNUM
 *
 */



/*                                                      ceill()
 *                                                      floorl()
 *                                                      frexpl()
 *                                                      ldexpl()
 *                                                      fabsl()
 *                                                      signbitl()
 *                                                      isnanl()
 *                                                      isfinitel()
 *
 *      Floating point numeric utilities
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y;
 * long double ceill(), floorl(), frexpl(), ldexpl(), fabsl();
 * int signbitl(), isnanl(), isfinitel();
 * int expnt, n;
 *
 * y = floorl(x);
 * y = ceill(x);
 * y = frexpl( x, &expnt );
 * y = ldexpl( x, n );
 * y = fabsl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * All four routines return a long double precision floating point
 * result.
 *
 * floorl() returns the largest integer less than or equal to x.
 * It truncates toward minus infinity.
 *
 * ceill() returns the smallest integer greater than or equal
 * to x.  It truncates toward plus infinity.
 *
 * frexpl() extracts the exponent from x.  It returns an integer
 * power of two to expnt and the significand between 0.5 and 1
 * to y.  Thus  x = y * 2**expn.
 *
 * ldexpl() multiplies x by 2**n.
 *
 * fabsl() returns the absolute value of its argument.
 *
 * signbitl(x) returns 1 if the sign bit of x is 1, else 0.
 *
 * These functions are part of the standard C run time library
 * for some but not all C compilers.  The ones supplied are
 * written in C for IEEE arithmetic.  They should
 * be used only if your compiler library does not already have
 * them.
 *
 * The IEEE versions assume that denormal numbers are implemented
 * in the arithmetic.  Some modifications will be required if
 * the arithmetic has abrupt rather than gradual underflow.
 */



/*                                                      ieee.c
 *
 *    Extended precision IEEE binary floating point arithmetic routines
 *
 * Numbers are stored in C language as arrays of 16-bit unsigned
 * short integers.  The arguments of the routines are pointers to
 * the arrays.
 *
 *
 * External e type data structure, simulates Intel 8087 chip
```

```
* temporary real format but possibly with a larger significand:
*
*      NE-1 significand words  (least significant word first,
*                               most significant bit is normally set)
*      exponent                (value = EXONE for 1.0,
*                              top bit is the sign)
*
*
* Internal data structure of a number (a "word" is 16 bits):
*
* ei[0]        sign word       (0 for positive, 0xffff for negative)
* ei[1]        biased exponent (value = EXONE for the number 1.0)
* ei[2]        high guard word (always zero after normalization)
* ei[3]
* to ei[NI-2]  significand     (NI-4 significand words,
*                               most significant word first,
*                               most significant bit is set)
* ei[NI-1]     low guard word  (0x8000 bit is rounding place)
*
*
*
*              Routines for external format numbers
*
*      asctoe( string, e )     ASCII string to extended double e type
*      asctoe64( string, &d )  ASCII string to long double
*      asctoe53( string, &d )  ASCII string to double
*      asctoe24( string, &f )  ASCII string to single
*      asctoeg( string, e, prec ) ASCII string to specified precision
*      e24toe( &f, e )         IEEE single precision to e type
*      e53toe( &d, e )         IEEE double precision to e type
*      e64toe( &d, e )         IEEE long double precision to e type
*      eabs(e)                 absolute value
*      eadd( a, b, c )         c = b + a
*      eclear(e)               e = 0
*      ecmp (a, b)             Returns 1 if a > b, 0 if a == b,
*                              -1 if a < b, -2 if either a or b is a NaN.
*      ediv( a, b, c )         c = b / a
*      efloor( a, b )          truncate to integer, toward -infinity
*      efrexp( a, exp, s )     extract exponent and significand
*      eifrac( e, &l, frac )   e to long integer and e type fraction
*      euifrac( e, &l, frac )  e to unsigned long integer and e type fraction
*      einfin( e )             set e to infinity, leaving its sign alone
*      eldexp( a, n, b )       multiply by 2**n
*      emov( a, b )            b = a
*      emul( a, b, c )         c = b * a
*      eneg(e)                 e = -e
*      eround( a, b )          b = nearest integer value to a
*      esub( a, b, c )         c = b - a
*      e24toasc( &f, str, n )  single to ASCII string, n digits after decimal
*      e53toasc( &d, str, n )  double to ASCII string, n digits after decimal
*      e64toasc( &d, str, n )  long double to ASCII string
*      etoasc( e, str, n )     e to ASCII string, n digits after decimal
*      etoe24( e, &f )         convert e type to IEEE single precision
*      etoe53( e, &d )         convert e type to IEEE double precision
*      etoe64( e, &d )         convert e type to IEEE long double precision
*      ltoe( &l, e )           long (32 bit) integer to e type
*      ultoe( &l, e )          unsigned long (32 bit) integer to e type
*      eisneg( e )             1 if sign bit of e != 0, else 0
*      eisinf( e )             1 if e has maximum exponent (non-IEEE)
*                              or is infinite (IEEE)
*      eisnan( e )             1 if e is a NaN
*      esqrt( a, b )           b = square root of a
*
*
*              Routines for internal format numbers
*
*      eaddm( ai, bi )         add significands, bi = bi + ai
*      ecleaz(ei)              ei = 0
*      ecleazs(ei)             set ei = 0 but leave its sign alone
*      ecmpm( ai, bi )         compare significands, return 1, 0, or -1
*      edivm( ai, bi )         divide  significands, bi = bi / ai
*      emdnorm(ai,l,s,exp)     normalize and round off
*      emovi( a, ai )          convert external a to internal ai
*      emovo( ai, a )          convert internal ai to external a
*      emovz( ai, bi )         bi = ai, low guard word of bi = 0
*      emulm( ai, bi )         multiply significands, bi = bi * ai
*      enormlz(ei)             left-justify the significand
*      eshdn1( ai )            shift significand and guards down 1 bit
*      eshdn8( ai )            shift down 8 bits
*      eshdn6( ai )            shift down 16 bits
*      eshift( ai, n )         shift ai n bits up (or down if n < 0)
*      eshup1( ai )            shift significand and guards up 1 bit
*      eshup8( ai )            shift up 8 bits
*      eshup6( ai )            shift up 16 bits
*      esubm( ai, bi )         subtract significands, bi = bi - ai
*
*
* The result is always normalized and rounded to NI-4 word precision
* after each arithmetic operation.
*
* Exception flags are NOT fully supported.
*
* Define INFINITIES in mconf.h for support of infinity; otherwise a
* saturation arithmetic is implemented.
*
* Define NANS for support of Not-a-Number items; otherwise the
* arithmetic will never produce a NaN output, and might be confused
* by a NaN input.
* If NaN's are supported, the output of ecmp(a,b) is -2 if
* either a or b is a NaN. This means asking if(ecmp(a,b) < 0)
```

```
 * may not be legitimate. Use if(ecmp(a,b) == -1) for less-than
 * if in doubt.
 * Signaling NaN's are NOT supported; they are treated the same
 * as quiet NaN's.
 *
 * Denormals are always supported here where appropriate (e.g., not
 * for conversion to DEC numbers).
 */



/*                                                      j0l.c
 *
 *      Bessel function of order zero
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, j0l();
 *
 * y = j0l( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of first kind, order zero of the argument.
 *
 * The domain is divided into two major intervals [0, 2] and
 * (2, infinity). In the first interval the rational approximation
 * is J0(x) = 1 - x^2 / 4 + x^4 R(x^2)
 * The second interval is further partitioned into eight equal segments
 * of 1/x.
 *
 * J0(x) = sqrt(2/(pi x)) (P0(x) cos(X) - Q0(x) sin(X)),
 * X = x - pi/4,
 *
 * and the auxiliary functions are given by
 *
 * J0(x)cos(X) + Y0(x)sin(X) = sqrt( 2/(pi x)) P0(x),
 * P0(x) = 1 + 1/x^2 R(1/x^2)
 *
 * Y0(x)cos(X) - J0(x)sin(X) = sqrt( 2/(pi x)) Q0(x),
 * Q0(x) = 1/x (-.125 + 1/x^2 R(1/x^2))
 *
 *
 *
 * ACCURACY:
 *
 *                      Absolute error:
 * arithmetic   domain      # trials      peak         rms
 *    IEEE      0, 30       100000      1.7e-34      2.4e-35
 *
 */



/*                                                      y0l
 *
 *      Bessel function of the second kind, order zero
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, y0l();
 *
 * y = y0l( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of the second kind, of order
 * zero, of the argument.
 *
 * The approximation is the same as for J0(x), and
 * Y0(x) = sqrt(2/(pi x)) (P0(x) sin(X) + Q0(x) cos(X)).
 *
 * ACCURACY:
 *
 *  Absolute error, when y0(x) < 1; else relative error:
 *
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0, 30       100000      3.0e-34     2.7e-35
 *
 */



/*                                                      j1l.c
 *
 *      Bessel function of order one
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, j1l();
```

```
 *
 * y = j1l( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of first kind, order one of the argument.
 *
 * The domain is divided into two major intervals [0, 2] and
 * (2, infinity). In the first interval the rational approximation is
 * J1(x) = .5x + x x^2 R(x^2)
 *
 * The second interval is further partitioned into eight equal segments
 * of 1/x.
 * J1(x) = sqrt(2/(pi x)) (P1(x) cos(X) - Q1(x) sin(X)),
 * X = x - 3 pi / 4,
 *
 * and the auxiliary functions are given by
 *
 * J1(x)cos(X) + Y1(x)sin(X) = sqrt( 2/(pi x)) P1(x),
 * P1(x) = 1 + 1/x^2 R(1/x^2)
 *
 * Y1(x)cos(X) - J1(x)sin(X) = sqrt( 2/(pi x)) Q1(x),
 * Q1(x) = 1/x (.375 + 1/x^2 R(1/x^2)).
 *
 *
 *
 * ACCURACY:
 *
 *                      Absolute error:
 * arithmetic   domain      # trials      peak         rms
 *    IEEE      0, 30       100000      2.8e-34      2.7e-35
 *
 *
 */



/*                                                      y1l
 *
 *      Bessel function of the second kind, order one
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, y1l();
 *
 * y = y1l( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of the second kind, of order
 * one, of the argument.
 *
 * The domain is divided into two major intervals [0, 2] and
 * (2, infinity). In the first interval the rational approximation is
 * Y1(x) = 2/pi * (log(x) * J1(x) - 1/x) + x R(x^2) .
 * In the second interval the approximation is the same as for J1(x), and
 * Y1(x) = sqrt(2/(pi x)) (P1(x) sin(X) + Q1(x) cos(X)),
 * X = x - 3 pi / 4.
 *
 * ACCURACY:
 *
 *  Absolute error, when y0(x) < 1; else relative error:
 *
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0, 30       100000      2.7e-34     2.9e-35
 *
 */



/*                                                      jnll.c
 *
 *      Bessel function of integer order
 *
 *
 *
 * SYNOPSIS:
 *
 * int n;
 * long double x, y, jnl();
 *
 * y = jnl( n, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of order n, where n is a
 * (possibly negative) integer.
 *
 * The ratio of jn(x) to j0(x) is computed by backward
 * recurrence.  First the ratio jn/jn-1 is found by a
 * continued fraction expansion.  Then the recurrence
 * relating successive orders is applied until j0 or j1 is
```

```
 * reached.
 *
 * If n = 0 or 1 the routine for j0 or j1 is called
 * directly.
 *
 *
 *
 * ACCURACY:
 *
 *                       Absolute error:
 * arithmetic   domain      # trials      peak         rms
 *    IEEE     -30, 30        10000      2.6e-34     4.6e-35
 *
 *
 * Not suitable for large n or x.
 *
 */



/*                                                      lgammall.c
 *
 *      Natural logarithm of gamma function
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, lgammal();
 * extern int sgngam;
 *
 * y = lgammal(x);
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the base e (2.718...) logarithm of the absolute
 * value of the gamma function of the argument.
 * The sign (+1 or -1) of the gamma function is returned in a
 * global (extern) variable named sgngam.
 *
 * The positive domain is partitioned into numerous segments for approximation.
 * For x > 10,
 *   log gamma(x) = (x - 0.5) log(x) - x + log sqrt(2 pi) + 1/x R(1/x^2)
 * Near the minimum at x = x0 = 1.46... the approximation is
 *   log gamma(x0 + z) = log gamma(x0) + z^2 P(z)/Q(z)
 * for small z.
 * Elsewhere between 0 and 10,
 *   log gamma(n + z) = log gamma(n) + z P(z)/Q(z)
 * for various selected n and small z.
 *
 * The cosecant reflection formula is employed for negative arguments.
 *
 * Arguments greater than MAXLGML (10^4928) return MAXNUML.
 *
 *
 * ACCURACY:
 *
 *
 * arithmetic      domain        # trials     peak         rms
 *                                            Relative error:
 *    IEEE         10, 30         100000     3.9e-34     9.8e-35
 *    IEEE          0, 10         100000     3.8e-34     5.3e-35
 *                                            Absolute error:
 *    IEEE         -10, 0         100000     8.0e-34     8.0e-35
 *    IEEE         -30, -10       100000     4.4e-34     1.0e-34
 *    IEEE        -100, 100       100000                 1.0e-34
 *
 * The absolute error criterion is the same as relative error
 * when the function magnitude is greater than one but it is absolute
 * when the magnitude is less than one.
 *
 */



/*                                                      log10l.c
 *
 *      Common logarithm, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, log10l();
 *
 * y = log10l( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the base 10 logarithm of x.
 *
 * The argument is separated into its exponent and fractional
 * parts.  If the exponent is between -1 and +1, the logarithm
 * of the fraction is approximated by
 *
 *     log(1+x) = x - 0.5 x**2 + x**3 P(x)/Q(x).
```

```
 *
 * Otherwise, setting  z = 2(x-1)/x+1),
 *
 *     log(x) = z + z**3 P(z)/Q(z).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0.5, 2.0     30000      2.3e-34     4.9e-35
 *    IEEE     exp(+-10000)  30000      1.0e-34     4.1e-35
 *
 * In the tests over the interval exp(+-10000), the logarithms
 * of the random arguments were uniformly distributed over
 * [-10000, +10000].
 *
 * ERROR MESSAGES:
 *
 * log singularity:  x = 0; returns MINLOG
 * log domain:       x < 0; returns MINLOG
 */



/*                                                      log1pl.c
 *
 *      Relative error logarithm
 *      Natural logarithm of 1+x, 128-bit long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, log1pl();
 *
 * y = log1pl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the base e (2.718...) logarithm of 1+x.
 *
 * The argument 1+x is separated into its exponent and fractional
 * parts.  If the exponent is between -1 and +1, the logarithm
 * of the fraction is approximated by
 *
 *     log(1+x) = x - 0.5 x^2 + x^3 P(x)/Q(x).
 *
 * Otherwise, setting  z = 2(x-1)/x+1),
 *
 *     log(x) = z + z^3 P(z)/Q(z).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -1, 8       100000      1.9e-34     4.3e-35
 */



/*                                                      log2l.c
 *
 *      Base 2 logarithm, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, log2l();
 *
 * y = log2l( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the base 2 logarithm of x.
 *
 * The argument is separated into its exponent and fractional
 * parts.  If the exponent is between -1 and +1, the (natural)
 * logarithm of the fraction is approximated by
 *
 *     log(1+x) = x - 0.5 x**2 + x**3 P(x)/Q(x).
 *
 * Otherwise, setting  z = 2(x-1)/x+1),
 *
 *     log(x) = z + z**3 P(z)/Q(z).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
```

```
 *    IEEE      0.5, 2.0     100,000    1.3e-34     4.5e-35
 *    IEEE     exp(+-10000)  100,000    9.6e-35     4.0e-35
 *
 * In the tests over the interval exp(+-10000), the logarithms
 * of the random arguments were uniformly distributed over
 * [-10000, +10000].
 *
 * ERROR MESSAGES:
 *
 * log singularity:  x = 0; returns MINLOG
 * log domain:       x < 0; returns MINLOG
 */



/*                                                      logl.c
 *
 *      Natural logarithm, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, logl();
 *
 * y = logl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the base e (2.718...) logarithm of x.
 *
 * The argument is separated into its exponent and fractional
 * parts.  If the exponent is between -1 and +1, the logarithm
 * of the fraction is approximated by
 *
 *     log(1+x) = x - 0.5 x**2 + x**3 P(x)/Q(x).
 *
 * Otherwise, setting  z = 2(x-1)/x+1),
 *
 *     log(x) = z + z**3 P(z)/Q(z).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE   exp(+-MAXLOGL) 36,000      9.5e-35     4.1e-35
 *
 *
 * ERROR MESSAGES:
 *
 * log singularity:  x = 0; returns MINLOGL
 * log domain:       x < 0; returns MINLOGL
 */



/*                                                      ndtrll.c
 *
 *      Normal distribution function
 *      128-bit long double version
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, ndtrl();
 *
 * y = ndtrl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the area under the Gaussian probability density
 * function, integrated from minus infinity to x:
 *
 *                            x
 *                             -
 *                  1         | |          2
 *    ndtr(x)  = ---------    |    exp( - t /2 ) dt
 *               sqrt(2pi)  | |
 *                           -
 *                          -inf.
 *
 *             =  ( 1 + erf(z) ) / 2
 *             =  erfc(z) / 2
 *
 * where z = x/sqrt(2). Computation is via the functions
 * erf and erfc with care to avoid error amplification in computing exp(-x^2).
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     -13,0        50000       7.7e-34     1.7e-34
```

```
 *    IEEE     -106.5,-2    50000       6.1e-34     1.9e-34
 *    IEEE       0,3        50000       1.5e-34     3.9e-35
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition           value returned
 * erfcl underflow    x^2 / 2 > MAXLOGL        0.0
 *
 */



/*                                                     ndtrll.c
 *
 *      Error function
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, erfl();
 *
 * y = erfl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * The integral is
 *
 *                           x
 *                            -
 *                 2         | |          2
 *   erf(x)  =  --------     |    exp( - t  ) dt.
 *              sqrt(pi)   | |
 *                          -
 *                           0
 *
 * The magnitude of x is limited to about 106.56 for IEEE
 * arithmetic; 1 or -1 is returned outside this range.
 *
 * For 0 <= |x| < 1, erf(x) is computed by rational approximations; otherwise
 * erf(x) = 1 - erfc(x).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,1         50000       1.5e-34     4.4e-35
 *
 */



/*                                                     ndtrll.c
 *
 *      Complementary error function
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, erfcl();
 *
 * y = erfcl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 *
 *  1 - erf(x) =
 *
 *                           inf.
 *                             -
 *                  2         | |          2
 *   erfc(x)  =  --------     |    exp( - t  ) dt
 *               sqrt(pi)   | |
 *                           -
 *                            x
 *
 *
 * For small x, erfc(x) = 1 - erf(x); otherwise rational
 * approximations are computed.
 *
 * A special function expx2l.c is used to suppress error amplification
 * in computing exp(-x^2).
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,13        100000     5.8e-34      1.5e-34
 *    IEEE      6,106.56    100000     5.9e-34      1.5e-34
 *
 *
```

```
 * ERROR MESSAGES:
 *
 *   message          condition              value returned
 * erfcl underflow    x^2 > MAXLOGL              0.0
 *
 *
 */



/*                                                      mtherr.c
 *
 *      Library common error handling routine
 *
 *
 *
 * SYNOPSIS:
 *
 * char *fctnam;
 * int code;
 * void mtherr();
 *
 * mtherr( fctnam, code );
 *
 *
 *
 * DESCRIPTION:
 *
 * This routine may be called to report one of the following
 * error conditions (in the include file mconf.h).
 *
 *   Mnemonic        Value          Significance
 *
 *    DOMAIN            1       argument domain error
 *    SING              2       function singularity
 *    OVERFLOW          3       overflow range error
 *    UNDERFLOW         4       underflow range error
 *    TLOSS             5       total loss of precision
 *    PLOSS             6       partial loss of precision
 *    EDOM             33       Unix domain error code
 *    ERANGE           34       Unix range error code
 *
 * The default version of the file prints the function name,
 * passed to it by the pointer fctnam, followed by the
 * error condition.  The display is directed to the standard
 * output device.  The routine then returns to the calling
 * program.  Users may wish to modify the program to abort by
 * calling exit() under severe error conditions such as domain
 * errors.
 *
 * Since all error conditions pass control to this function,
 * the display may be easily changed, eliminated, or directed
 * to an error logging device.
 *
 * SEE ALSO:
 *
 * mconf.h
 *
 */



/*                                                      polevll.c
 *                                                      p1evll.c
 *
 *      Evaluate polynomial
 *
 *
 *
 * SYNOPSIS:
 *
 * int N;
 * long double x, y, coef[N+1], polevl[];
 *
 * y = polevll( x, coef, N );
 *
 *
 *
 * DESCRIPTION:
 *
 * Evaluates polynomial of degree N:
 *
 *                     2          N
 * y  =  C  + C x + C x  +...+ C x
 *        0    1     2          N
 *
 * Coefficients are stored in reverse order:
 *
 * coef[0] = C  , ..., coef[N] = C  .
 *            N                   0
 *
 *  The function p1evll() assumes that coef[N] = 1.0 and is
 * omitted from the array.  Its calling arguments are
 * otherwise the same as polevll().
 *
 *
 * SPEED:
 *
 * In the interest of speed, there are no checks for out
 * of bounds arithmetic.  This routine is used by most of
```

```
 * the functions in the library.  Depending on available
 * equipment features, the user may wish to rewrite the
 * program in microcode or assembly language.
 *
 */



/*							powil.c
 *
 *	Real raised to integer power, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, powil();
 * int n;
 *
 * y = powil( x, n );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns argument x raised to the nth power.
 * The routine efficiently decomposes n as a sum of powers of
 * two. The desired power is a product of two-to-the-kth
 * powers of x.  Thus to compute the 32767 power of x requires
 * 28 multiplications instead of 32767 multiplications.
 *
 *
 *
 * ACCURACY:
 *
 *
 *                      Relative error:
 * arithmetic   x domain   n domain  # trials      peak         rms
 *    IEEE     .001,1000  -1022,1023  100,000     7.5e-32     1.4e-32
 *    IEEE     .99,1.01     0,8700    100,000     4.6e-31     9.1e-32
 *
 * Returns MAXNUM on overflow, zero on underflow.
 *
 */



/*							powl.c
 *
 *	Power function, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, z, powl();
 *
 * z = powl( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Computes x raised to the yth power.  For noninteger y,
 *
 *      x^y  =  exp2( y log2(x) ).
 *
 * using the base 2 logarithm and exponential functions.  If y
 * is an integer, |y| < 32768, the function is computed by powil.
 *
 *
 *
 * ACCURACY:
 *
 * The relative error of pow(x,y) can be estimated
 * by   y dl ln(2),   where dl is the absolute error of
 * the internally computed base 2 logarithm.
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *
 *    IEEE     +-1000      100,000     1.0e-30      1.4e-31
 * .001 < x < 1000, with log(x) uniformly distributed.
 * -1000 < y < 1000, y uniformly distributed.
 *
 *    IEEE     0,8700      100,000     1.4e-30      3.1e-31
 * 0.99 < x < 1.01, 0 < y < 8700, uniformly distributed.
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * pow overflow     x^y > MAXNUM      MAXNUM
 * pow underflow   x^y < 1/MAXNUM       0.0
 * pow domain      x<0 and y noninteger  0.0
 *
 */
```

```
/*                                                      sinhl.c
 *
 *      Hyperbolic sine, 128-bit long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, sinhl();
 *
 * y = sinhl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns hyperbolic sine of argument in the range MINLOGL to
 * MAXLOGL.
 *
 * The range is partitioned into two segments.  If |x| <= 1, a
 * rational function of the form x + x**3 P(x)/Q(x) is employed.
 * Otherwise the calculation is sinh(x) = ( exp(x) - exp(-x) )/2.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       -2,2      100,000      4.1e-34     7.9e-35
 *
 */



/*                                                      sinl.c
 *
 *      Circular sine, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, sinl();
 *
 * y = sinl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Range reduction is into intervals of pi/4.  The reduction
 * error is nearly eliminated by contriving an extended precision
 * modular arithmetic.
 *
 * Two polynomial approximating functions are employed.
 * Between 0 and pi/4 the sine is approximated by the Cody
 * and Waite polynomial form
 *      x + x^3 P(x^2) .
 * Between pi/4 and pi/2 the cosine is represented as
 *      1 - .5 x^2 + x^4 Q(x^2) .
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain      # trials      peak         rms
 *    IEEE     +-3.6e16      100,000    2.0e-34     5.3e-35
 *
 * ERROR MESSAGES:
 *
 *   message           condition        value returned
 * sin total loss      x > 2^55              0.0
 *
 */



/*                                                      cosl.c
 *
 *      Circular cosine, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, cosl();
 *
 * y = cosl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Range reduction is into intervals of pi/4.  The reduction
 * error is nearly eliminated by contriving an extended precision
 * modular arithmetic.
 *
 * Two polynomial approximating functions are employed.
 * Between 0 and pi/4 the cosine is approximated by
```

```
 *      1 - .5 x^2 + x^4 Q(x^2) .
 * Between pi/4 and pi/2 the sine is represented by the Cody
 * and Waite polynomial form
 *      x  +  x^3 P(x^2) .
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain      # trials      peak         rms
 *    IEEE     +-3.6e16     100,000      2.0e-34     5.2e-35
 *
 * ERROR MESSAGES:
 *
 *   message           condition        value returned
 * cos total loss      x > 2^55              0.0
 */



/*                                                      sqrtl.c
 *
 *      Square root, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, sqrtl();
 *
 * y = sqrtl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the square root of x.
 *
 * Range reduction involves isolating the power of two of the
 * argument and using a polynomial approximation to obtain
 * a rough value for the square root.  Then Heron's iteration
 * is used three times to converge to an accurate value.
 *
 * Note, some arithmetic coprocessors such as the 8087 and
 * 68881 produce correctly rounded square roots, which this
 * routine will not.
 *
 * ACCURACY:
 *
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,10        30000       8.1e-20     3.1e-20
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * sqrt domain        x < 0            0.0
 *
 */



/*                                                      tanhl.c
 *
 *      Hyperbolic tangent, 128-bit long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, tanhl();
 *
 * y = tanhl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns hyperbolic tangent of argument in the range MINLOGL to
 * MAXLOGL.
 *
 * A rational function is used for |x| < 0.625.  The form
 * x + x**3 P(x)/Q(x) of Cody & Waite is employed.
 * Otherwise,
 *    tanh(x) = sinh(x)/cosh(x) = 1  -  2/(exp(2x) + 1).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -2,2       100,000      2.1e-34     4.5e-35
 *
 */
```

```
/*							tanl.c
 *
 *	Circular tangent, 128-bit long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, tanl();
 *
 * y = tanl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the circular tangent of the radian argument x.
 *
 * Range reduction is modulo pi/4.  A rational function
 *       x + x**3 P(x**2)/Q(x**2)
 * is employed in the basic interval [0, pi/4].
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     +-3.6e16    100,000      3.0e-34      7.2e-35
 *
 * ERROR MESSAGES:
 *
 *   message         condition          value returned
 * tan total loss   x > 2^55                0.0
 *
 */
```

```
/*							cotl.c
 *
 *	Circular cotangent, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, cotl();
 *
 * y = cotl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the circular cotangent of the radian argument x.
 *
 * Range reduction is modulo pi/4.  A rational function
 *       x + x**3 P(x**2)/Q(x**2)
 * is employed in the basic interval [0, pi/4].
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     +-3.6e16    100,000      2.9e-34     7.2e-35
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition          value returned
 * cot total loss   x > 2^55                0.0
 * cot singularity  x = 0                  MAXNUM
 *
 */
```

To Cephes home page www.moshier.net:

Last update: 27 January 2002