Cephes Mathematical Library

Source code archives

[Documentation for single precision library.](#)
[Documentation for double precision library.](#)
[Documentation for 80-bit long double library.](#)
[Documentation for 128-bit long double library.](#)
[Documentation for extended precision library.](#)

Single Precision Special Functions

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

- acoshf, Inverse hyperbolic cosine
- airyf, Airy function
- asinf, Inverse circular sine
- acosf, Inverse circular cosine
- asinhf, Inverse hyperbolic sine
- atanf, Inverse circular tangent
- atan2f, Quadrant correct inverse circular tangent
- atanhf, Inverse hyperbolic tangent
- bdtrf, Binomial distribution
- bdtrcf, Complemented binomial distribution
- bdtrif, Inverse binomial distribution
- betaf, Beta function
- cbrtf, Cube root
- chbevlf, Evaluate Chebyshev series
- chdtrf, Chi-square distribution
- chdtrcf, Complemented Chi-square distribution
- chdtrif, Inverse of complemented Chi-square distribution
- clogf, Complex natural logarithm
- cexpf, Complex exponential
- csinf, Complex circular sine
- ccosf, Complex circular cosine
- ctanf, Complex circular tangent
- ccotf, Complex circular cotangent
- casinf, Complex circular arc sine
- cacosf, Complex circular arc cosine
- catanf, Complex circular arc tangent
- cmplxf, Complex arithmetic
- coshf, Hyperbolic cosine
- dawsnf, Dawson's Integral
- ellief, Incomplete elliptic integral of the second kind
- ellikf, Incomplete elliptic integral of the first kind
- ellpef, Complete elliptic integral of the second kind
- ellpjf, Jacobian Elliptic Functions
- ellpkf, Complete elliptic integral of the first kind
- exp10f, Base 10 exponential function
- exp2f, Base 2 exponential function
- expf, Exponential function
- expnf, Exponential integral En
- expx2f, Exponential of squared argument
- facf, Factorial function
- fdtrf, F distribution
- fdtrcf, Complemented F distribution
- fdtrif, Inverse of complemented F distribution
- ceilf, Round up to integer
- floorf, Round down to integer
- frexpf, Extract exponent and significand
- ldexpf, Apply exponent
- signbitf, Extract sign
- isnanf, Test for not a number
- isfinitef, Test for infinity
- fresnlf, Fresnel integral
- gammaf, Gamma function
- lgamf, Natural logarithm of gamma function
- gdtrf, Gamma distribution function
- gdtrcf, Complemented gamma distribution function
- hyp2f1f, Gauss hypergeometric function
- hypergf, Confluent hypergeometric function
- i0f, Modified Bessel function of order zero
- i0ef, Modified Bessel function of order zero, exponentially scaled
- i1f, Modified Bessel function of order one
- i1ef, Modified Bessel function of order one, exponentially scaled
- igamf, Incomplete gamma integral
- igamcf, Complemented incomplete gamma integral
- igamif, Inverse of complemented incomplete gamma integral
- incbetf, Incomplete beta integral
- incbif, Inverse of incomplete beta integral
- ivf, Modified Bessel function of noninteger order

```
/*                                                      acoshf.c
 *
 *      Inverse hyperbolic cosine
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, acoshf();
 *
 * y = acoshf( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns inverse hyperbolic cosine of argument.
 *
 * If 1 <= x < 1.5, a polynomial approximation
 *
 *      sqrt(z) * P(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         100000      1.8e-7       3.9e-8
 *    IEEE      1,2000      100000                   3.0e-8
 *
 *
 * ERROR MESSAGES:
```

```
 *
 *   message         condition      value returned
 * acoshf domain      |x| < 1            0.0
 *
 */



/*							airy.c
 *
 *	Airy function
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, ai, aip, bi, bip;
 * int airyf();
 *
 * airyf( x, &ai, &aip, &bi, &bip );
 *
 *
 *
 * DESCRIPTION:
 *
 * Solution of the differential equation
 *
 *	y"(x) = xy.
 *
 * The function returns the two independent solutions Ai, Bi
 * and their first derivatives Ai'(x), Bi'(x).
 *
 * Evaluation is by power series summation for small x,
 * by rational minimax approximations for large x.
 *
 *
 *
 * ACCURACY:
 * Error criterion is absolute when function <= 1, relative
 * when function > 1, except * denotes relative error criterion.
 * For large negative x, the absolute error increases as x^1.5.
 * For large positive x, the relative error increases as x^1.5.
 *
 * Arithmetic  domain   function  # trials      peak         rms
 * IEEE        -10, 0     Ai        50000       7.0e-7      1.2e-7
 * IEEE          0, 10    Ai        50000       9.9e-6*     6.8e-7*
 * IEEE        -10, 0     Ai'       50000       2.4e-6      3.5e-7
 * IEEE          0, 10    Ai'       50000       8.7e-6*     6.2e-7*
 * IEEE        -10, 10    Bi       100000       2.2e-6      2.6e-7
 * IEEE        -10, 10    Bi'       50000       2.2e-6      3.5e-7
 *
 */



/*							asinf.c
 *
 *	Inverse circular sine
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, asinf();
 *
 * y = asinf( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns radian angle between -pi/2 and +pi/2 whose sine is x.
 *
 * A polynomial of the form x + x**3 P(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       100000       2.5e-7       5.0e-8
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * asinf domain        |x| > 1           0.0
 *
 */



/*							acosf()
 *
 *	Inverse circular cosine
 *
```

```
 *
 *
 * SYNOPSIS:
 *
 * float x, y, acosf();
 *
 * y = acosf( 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      100000       1.4e-7      4.2e-8
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * acosf domain        |x| > 1           0.0
 */



/*                                                      asinhf.c
 *
 *      Inverse hyperbolic sine
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, asinhf();
 *
 * y = asinhf( 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     -3,3        100000       2.4e-7      4.1e-8
 *
 */



/*                                                      atanf.c
 *
 *      Inverse circular tangent
 *      (arctangent)
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, atanf();
 *
 * y = atanf( 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 ).  A polynomial approximates
 * the function in this basic interval.
 *
```

```
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10, 10     100000      1.9e-7      4.1e-8
 *
 */



/*                                                      atan2f()
 *
 *      Quadrant correct inverse circular tangent
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, z, atan2f();
 *
 * z = atan2f( 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     100000      1.9e-7      4.1e-8
 * See atan.c.
 *
 */



/*                                                      atanhf.c
 *
 *      Inverse hyperbolic tangent
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, atanhf();
 *
 * y = atanhf( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns inverse hyperbolic tangent of argument in the range
 * MINLOGF to MAXLOGF.
 *
 * If |x| < 0.5, a polynomial approximation is used.
 * Otherwise,
 *        atanh(x) = 0.5 * log( (1+x)/(1-x) ).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -1,1        100000      1.4e-7      3.1e-8
 *
 */



/*                                                      bdtrf.c
 *
 *      Binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k, n;
 * float p, y, bdtrf();
 *
 * y = bdtrf( k, n, p );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the sum of the terms 0 through k of the Binomial
 * probability density:
```

```
 *
 *   k
 *   --  ( n )   j      n-j
 *   >   (   )  p  (1-p)
 *   --  ( j )
 *  j=0
 *
 * The terms are not summed directly; instead the incomplete
 * beta integral is employed, according to the formula
 *
 * y = bdtr( k, n, p ) = incbet( n-k, k+1, 1-p ).
 *
 * The arguments must be positive, with p ranging from 0 to 1.
 *
 *
 *
 * ACCURACY:
 *
 *        Relative error (p varies from 0 to 1):
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,100       2000       6.9e-5      1.1e-5
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * bdtrf domain        k < 0            0.0
 *                     n < k
 *                     x < 0, x > 1
 *
 */



/*                                                      bdtrcf()
 *
 *      Complemented binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k, n;
 * float p, y, bdtrcf();
 *
 * y = bdtrcf( k, n, p );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the sum of the terms k+1 through n of the Binomial
 * probability density:
 *
 *   n
 *   --  ( n )   j      n-j
 *   >   (   )  p  (1-p)
 *   --  ( j )
 *  j=k+1
 *
 * The terms are not summed directly; instead the incomplete
 * beta integral is employed, according to the formula
 *
 * y = bdtrc( k, n, p ) = incbet( k+1, n-k, p ).
 *
 * The arguments must be positive, with p ranging from 0 to 1.
 *
 *
 *
 * ACCURACY:
 *
 *        Relative error (p varies from 0 to 1):
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,100       2000       6.0e-5      1.2e-5
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * bdtrcf domain     x<0, x>1, n<k       0.0
 */



/*                                                      bdtrif()
 *
 *      Inverse binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k, n;
 * float p, y, bdtrif();
 *
 * p = bdtrf( k, n, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Finds the event probability p such that the sum of the
```

```
 * terms 0 through k of the Binomial probability density
 * is equal to the given cumulative probability y.
 *
 * This is accomplished using the inverse beta integral
 * function and the relation
 *
 * 1 - p = incbi( n-k, k+1, y ).
 *
 *
 *
 *
 * ACCURACY:
 *
 *        Relative error (p varies from 0 to 1):
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,100       2000       3.5e-5      3.3e-6
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * bdtrif domain    k < 0, n <= k         0.0
 *                  x < 0, x > 1
 *
 */



/*                                                      betaf.c
 *
 *      Beta function
 *
 *
 *
 * SYNOPSIS:
 *
 * float a, b, y, betaf();
 *
 * y = betaf( a, b );
 *
 *
 *
 * DESCRIPTION:
 *
 *                   -     -
 *                  | (a) | (b)
 * beta( a, b )  =  -----------.
 *                     -
 *                    | (a+b)
 *
 * For large arguments the logarithm of the function is
 * evaluated using lgam(), then exponentiated.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,30       10000       4.0e-5      6.0e-6
 *    IEEE       -20,0      10000       4.9e-3      5.4e-5
 *
 * ERROR MESSAGES:
 *
 *   message         condition          value returned
 * betaf overflow   log(beta) > MAXLOG       0.0
 *                  a or b < 0 integer        0.0
 *
 */



/*                                                      cbrtf.c
 *
 *      Cube root
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, cbrtf();
 *
 * y = cbrtf( 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 to converge to an accurate result.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
```

```
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,1e38      100000      7.6e-8      2.7e-8
 *
 */



/*							chbevlf.c
 *
 *	Evaluate Chebyshev series
 *
 *
 *
 * SYNOPSIS:
 *
 * int N;
 * float x, y, coef[N], chebevlf();
 *
 * y = chbevlf( x, coef, N );
 *
 *
 *
 * DESCRIPTION:
 *
 * Evaluates the series
 *
 *        N-1
 *         - '
 *  y  =   >   coef[i] T (x/2)
 *         -            i
 *        i=0
 *
 * of Chebyshev polynomials Ti at argument x/2.
 *
 * Coefficients are stored in reverse order, i.e. the zero
 * order term is last in the array.  Note N is the number of
 * coefficients, not the order.
 *
 * If coefficients are for the interval a to b, x must
 * have been transformed to x -> 2(2x - b - a)/(b-a) before
 * entering the routine.  This maps x from (a, b) to (-1, 1),
 * over which the Chebyshev polynomials are defined.
 *
 * If the coefficients are for the inverted interval, in
 * which (a, b) is mapped to (1/b, 1/a), the transformation
 * required is x -> 2(2ab/x - b - a)/(b-a).  If b is infinity,
 * this becomes x -> 4a/x - 1.
 *
 *
 *
 * SPEED:
 *
 * Taking advantage of the recurrence properties of the
 * Chebyshev polynomials, the routine requires one more
 * addition per loop than evaluating a nested polynomial of
 * the same degree.
 *
 */



/*							chdtrf.c
 *
 *	Chi-square distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * float df, x, y, chdtrf();
 *
 * y = chdtrf( df, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the area under the left hand tail (from 0 to x)
 * of the Chi square probability density function with
 * v degrees of freedom.
 *
 *
 *                                  inf.
 *                                    -
 *                        1          | |  v/2-1  -t/2
 *  P( x | v )   =   -----------     |   t      e     dt
 *                    v/2  -       | |
 *                   2    | (v/2)   -
 *                                   x
 *
 * where x is the Chi-square variable.
 *
 * The incomplete gamma integral is used, according to the
 * formula
 *
 *	y = chdtr( v, x ) = igam( v/2.0, x/2.0 ).
 *
 *
 * The arguments must both be positive.
 *
```

```
 *
 *
 * ACCURACY:
 *
 *        Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,100       5000       3.2e-5      5.0e-6
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * chdtrf domain  x < 0 or v < 1        0.0
 */



/*                                                      chdtrcf()
 *
 *      Complemented Chi-square distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * float v, x, y, chdtrcf();
 *
 * y = chdtrcf( v, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the area under the right hand tail (from x to
 * infinity) of the Chi square probability density function
 * with v degrees of freedom:
 *
 *
 *                                  inf.
 *                                    -
 *                        1          | |  v/2-1  -t/2
 *  P( x | v )   =   -----------     |   t      e     dt
 *                    v/2  -       | |
 *                   2    | (v/2)   -
 *                                   x
 *
 * where x is the Chi-square variable.
 *
 * The incomplete gamma integral is used, according to the
 * formula
 *
 *      y = chdtr( v, x ) = igamc( v/2.0, x/2.0 ).
 *
 *
 * The arguments must both be positive.
 *
 *
 *
 * ACCURACY:
 *
 *        Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,100       5000       2.7e-5      3.2e-6
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * chdtrc domain  x < 0 or v < 1        0.0
 */



/*                                                      chdtrif()
 *
 *      Inverse of complemented Chi-square distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * float df, x, y, chdtrif();
 *
 * x = chdtrif( df, y );
 *
 *
 *
 *
 * DESCRIPTION:
 *
 * Finds the Chi-square argument x such that the integral
 * from x to infinity of the Chi-square density is equal
 * to the given cumulative probability y.
 *
 * This is accomplished using the inverse gamma integral
 * function and the relation
 *
 *    x/2 = igami( df/2, y );
 *
 *
 *
 *
```

```
 * ACCURACY:
 *
 *        Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,100       10000      2.2e-5      8.5e-7
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * chdtri domain   y < 0 or y > 1        0.0
 *                     v < 1
 *
 */



/*                                                      clogf.c
 *
 *      Complex natural logarithm
 *
 *
 *
 * SYNOPSIS:
 *
 * void clogf();
 * cmplxf z, w;
 *
 * clogf( &z, &w );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns complex logarithm to the base e (2.718...) of
 * the complex argument x.
 *
 * If z = x + iy, r = sqrt( x**2 + y**2 ),
 * then
 *       w = log(r) + i arctan(y/x).
 *
 * The arctangent ranges from -PI to +PI.
 *
 *
 * ACCURACY:
 *
 *                     Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10,+10     30000       1.9e-6       6.2e-8
 *
 * Larger relative error can be observed for z near 1 +i0.
 * In IEEE arithmetic the peak absolute error is 3.1e-7.
 *
 */



/*                                                      cexpf()
 *
 *      Complex exponential function
 *
 *
 *
 * SYNOPSIS:
 *
 * void cexpf();
 * cmplxf z, w;
 *
 * cexpf( &z, &w );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the exponential of the complex argument z
 * into the complex result w.
 *
 * If
 *     z = x + iy,
 *     r = exp(x),
 *
 * then
 *
 *     w = r cos y + i r sin y.
 *
 *
 * ACCURACY:
 *
 *                     Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10,+10     30000       1.4e-7      4.5e-8
 *
 */



/*                                                      csinf()
 *
 *      Complex circular sine
 *
 *
```

```
 *
 * SYNOPSIS:
 *
 * void csinf();
 * cmplxf z, w;
 *
 * csinf( &z, &w );
 *
 *
 *
 * DESCRIPTION:
 *
 * If
 *     z = x + iy,
 *
 * then
 *
 *     w = sin x  cosh y  +  i cos x sinh y.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10,+10     30000       1.9e-7      5.5e-8
 *
 */



/*                                                      ccosf()
 *
 *      Complex circular cosine
 *
 *
 *
 * SYNOPSIS:
 *
 * void ccosf();
 * cmplxf z, w;
 *
 * ccosf( &z, &w );
 *
 *
 *
 * DESCRIPTION:
 *
 * If
 *     z = x + iy,
 *
 * then
 *
 *     w = cos x  cosh y  -  i sin x sinh y.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10,+10     30000       1.8e-7       5.5e-8
 */



/*                                                      ctanf()
 *
 *      Complex circular tangent
 *
 *
 *
 * SYNOPSIS:
 *
 * void ctanf();
 * cmplxf z, w;
 *
 * ctanf( &z, &w );
 *
 *
 *
 * DESCRIPTION:
 *
 * If
 *     z = x + iy,
 *
 * then
 *
 *           sin 2x  +  i sinh 2y
 *     w  =  --------------------.
 *            cos 2x  +  cosh 2y
 *
 * On the real axis the denominator is zero at odd multiples
 * of PI/2.  The denominator is evaluated by its Taylor
 * series near these points.
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
```

```
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10,+10     30000       3.3e-7       5.1e-8
 */



/*							ccotf()
 *
 *	Complex circular cotangent
 *
 *
 *
 * SYNOPSIS:
 *
 * void ccotf();
 * cmplxf z, w;
 *
 * ccotf( &z, &w );
 *
 *
 *
 * DESCRIPTION:
 *
 * If
 *     z = x + iy,
 *
 * then
 *
 *           sin 2x  -  i sinh 2y
 *     w  =  --------------------.
 *            cosh 2y  -  cos 2x
 *
 * On the real axis, the denominator has zeros at even
 * multiples of PI/2.  Near these points it is evaluated
 * by a Taylor series.
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10,+10     30000       3.6e-7       5.7e-8
 * Also tested by ctan * ccot = 1 + i0.
 */



/*							casinf()
 *
 *	Complex circular arc sine
 *
 *
 *
 * SYNOPSIS:
 *
 * void casinf();
 * cmplxf z, w;
 *
 * casinf( &z, &w );
 *
 *
 *
 * DESCRIPTION:
 *
 * Inverse complex sine:
 *
 *                               2
 * w = -i clog( iz + csqrt( 1 - z ) ).
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10,+10     30000       1.1e-5      1.5e-6
 * Larger relative error can be observed for z near zero.
 *
 */



/*							cacosf()
 *
 *	Complex circular arc cosine
 *
 *
 *
 * SYNOPSIS:
 *
 * void cacosf();
 * cmplxf z, w;
 *
 * cacosf( &z, &w );
 *
 *
 *
 * DESCRIPTION:
 *
 *
 * w = arccos z  =  PI/2 - arcsin z.
```

```
 *
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10,+10     30000       9.2e-6       1.2e-6
 *
 */



/*                                                      catan()
 *
 *      Complex circular arc tangent
 *
 *
 *
 * SYNOPSIS:
 *
 * void catan();
 * cmplxf z, w;
 *
 * catan( &z, &w );
 *
 *
 *
 * DESCRIPTION:
 *
 * If
 *     z = x + iy,
 *
 * then
 *          1       (    2x     )
 * Re w  =  - arctan(-----------)  +  k PI
 *          2       (     2    2)
 *                  (1 - x  - y )
 *
 *               ( 2         2)
 *          1    (x  +  (y+1) )
 * Im w  =  - log(------------)
 *          4    ( 2         2)
 *               (x  +  (y-1) )
 *
 * Where k is an arbitrary integer.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10,+10     30000        2.3e-6      5.2e-8
 *
 */



/*                                                      cmplxf.c
 *
 *      Complex number arithmetic
 *
 *
 *
 * SYNOPSIS:
 *
 * typedef struct {
 *      float r;     real part
 *      float i;     imaginary part
 *     }cmplxf;
 *
 * cmplxf *a, *b, *c;
 *
 * caddf( a, b, c );     c = b + a
 * csubf( a, b, c );     c = b - a
 * cmulf( a, b, c );     c = b * a
 * cdivf( a, b, c );     c = b / a
 * cnegf( c );           c = -c
 * cmovf( b, c );        c = b
 *
 *
 *
 * DESCRIPTION:
 *
 * Addition:
 *    c.r  =  b.r + a.r
 *    c.i  =  b.i + a.i
 *
 * Subtraction:
 *    c.r  =  b.r - a.r
 *    c.i  =  b.i - a.i
 *
 * Multiplication:
 *    c.r  =  b.r * a.r  -  b.i * a.i
 *    c.i  =  b.r * a.i  +  b.i * a.r
 *
 * Division:
 *    d    =  a.r * a.r  +  a.i * a.i
```

```
 *    c.r  = (b.r * a.r  + b.i * a.i)/d
 *    c.i  = (b.i * a.r  -  b.r * a.i)/d
 * ACCURACY:
 *
 * In DEC arithmetic, the test (1/z) * z = 1 had peak relative
 * error 3.1e-17, rms 1.2e-17.  The test (y/z) * (z/y) = 1 had
 * peak relative error 8.3e-17, rms 2.1e-17.
 *
 * Tests in the rectangle {-10,+10}:
 *                      Relative error:
 * arithmetic   function  # trials      peak         rms
 *    IEEE       cadd       30000       5.9e-8      2.6e-8
 *    IEEE       csub       30000       6.0e-8      2.6e-8
 *    IEEE       cmul       30000       1.1e-7      3.7e-8
 *    IEEE       cdiv       30000       2.1e-7      5.7e-8
 */



/*                                                      coshf.c
 *
 *      Hyperbolic cosine
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, coshf();
 *
 * y = coshf( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns hyperbolic cosine of argument in the range MINLOGF to
 * MAXLOGF.
 *
 * cosh(x)  =  ( exp(x) + exp(-x) )/2.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     +-MAXLOGF    100000      1.2e-7      2.8e-8
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * coshf overflow  |x| > MAXLOGF       MAXNUMF
 *
 *
 */



/*                                                      dawsnf.c
 *
 *      Dawson's Integral
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, dawsnf();
 *
 * y = dawsnf( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Approximates the integral
 *
 *                             x
 *                             -
 *                      2     | |        2
 *  dawsn(x)  =  exp( -x  )   |    exp( t  ) dt
 *                          | |
 *                           -
 *                           0
 *
 * Three different rational approximations are employed, for
 * the intervals 0 to 3.25; 3.25 to 6.25; and 6.25 up.
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,10        50000       4.4e-7      6.3e-8
 *
 *
 */
```

```
/*                                                      ellief.c
 *
 *      Incomplete elliptic integral of the second kind
 *
 *
 *
 * SYNOPSIS:
 *
 * float phi, m, y, ellief();
 *
 * y = ellief( phi, m );
 *
 *
 *
 * DESCRIPTION:
 *
 * Approximates the integral
 *
 *
 *                phi
 *                 -
 *                | |
 *                |                   2
 * E(phi\m)  =    |    sqrt( 1 - m sin t ) dt
 *                |
 *              | |
 *               -
 *                0
 *
 * of amplitude phi and modulus m, using the arithmetic -
 * geometric mean algorithm.
 *
 *
 *
 * ACCURACY:
 *
 * Tested at random arguments with phi in [0, 2] and m in
 * [0, 1].
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,2        10000       4.5e-7      7.4e-8
 *
 *
 */


/*                                                      ellikf.c
 *
 *      Incomplete elliptic integral of the first kind
 *
 *
 *
 * SYNOPSIS:
 *
 * float phi, m, y, ellikf();
 *
 * y = ellikf( phi, m );
 *
 *
 *
 * DESCRIPTION:
 *
 * Approximates the integral
 *
 *
 *
 *                phi
 *                 -
 *                | |
 *                |           dt
 * F(phi\m)  =    |    ------------------
 *                |                   2
 *              | |    sqrt( 1 - m sin t )
 *               -
 *                0
 *
 * of amplitude phi and modulus m, using the arithmetic -
 * geometric mean algorithm.
 *
 *
 *
 *
 * ACCURACY:
 *
 * Tested at random points with phi in [0, 2] and m in
 * [0, 1].
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,2         10000       2.9e-7      5.8e-8
 *
 *
 */


/*                                                      ellpef.c
 *
 *      Complete elliptic integral of the second kind
 *
```

```
 *
 *
 * SYNOPSIS:
 *
 * float m1, y, ellpef();
 *
 * y = ellpef( m1 );
 *
 *
 *
 * DESCRIPTION:
 *
 * Approximates the integral
 *
 *
 *            pi/2
 *             -
 *            | |                 2
 * E(m)  =    |    sqrt( 1 - m sin t ) dt
 *          | |
 *           -
 *            0
 *
 * Where m = 1 - m1, using the approximation
 *
 *      P(x)  -  x log x Q(x).
 *
 * Though there are no singularities, the argument m1 is used
 * rather than m for compatibility with ellpk().
 *
 * E(1) = 1; E(0) = pi/2.
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0, 1       30000       1.1e-7      3.9e-8
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * ellpef domain     x<0, x>1            0.0
 *
 */



/*                                                      ellpjf.c
 *
 *      Jacobian Elliptic Functions
 *
 *
 *
 * SYNOPSIS:
 *
 * float u, m, sn, cn, dn, phi;
 * int ellpj();
 *
 * ellpj( u, m, &sn, &cn, &dn, &phi );
 *
 *
 *
 * DESCRIPTION:
 *
 *
 * Evaluates the Jacobian elliptic functions sn(u|m), cn(u|m),
 * and dn(u|m) of parameter m between 0 and 1, and real
 * argument u.
 *
 * These functions are periodic, with quarter-period on the
 * real axis equal to the complete elliptic integral
 * ellpk(1.0-m).
 *
 * Relation to incomplete elliptic integral:
 * If u = ellik(phi,m), then sn(u|m) = sin(phi),
 * and cn(u|m) = cos(phi).  Phi is called the amplitude of u.
 *
 * Computation is by means of the arithmetic-geometric mean
 * algorithm, except when m is within 1e-9 of 0 or 1.  In the
 * latter case with m close to 1, the approximation applies
 * only for phi < pi/2.
 *
 * ACCURACY:
 *
 * Tested at random points with u between 0 and 10, m between
 * 0 and 1.
 *
 *            Absolute error (* = relative error):
 * arithmetic   function   # trials      peak         rms
 *    IEEE      sn          10000       1.7e-6      2.2e-7
 *    IEEE      cn          10000       1.6e-6      2.2e-7
 *    IEEE      dn         100000       3.2e-6      2.6e-7
 *    IEEE      phi         10000       3.9e-7*     6.7e-8*
 *
 * Larger errors occur for m near 1.
 * Peak error observed in consistency check using addition
 * theorem for sn(u+v) was 4e-16 (absolute).  Also tested by
 * the above relation to the incomplete elliptic integral.
```

```
 * Accuracy deteriorates when u is large.
 *
 */



/*							ellpkf.c
 *
 *	Complete elliptic integral of the first kind
 *
 *
 *
 * SYNOPSIS:
 *
 * float m1, y, ellpkf();
 *
 * y = ellpkf( m1 );
 *
 *
 *
 * DESCRIPTION:
 *
 * Approximates the integral
 *
 *
 *
 *            pi/2
 *             -
 *            | |
 *            |           dt
 * K(m)  =    |    ------------------
 *            |                   2
 *          | |    sqrt( 1 - m sin t )
 *           -
 *            0
 *
 * where m = 1 - m1, using the approximation
 *
 *     P(x)  -  log x Q(x).
 *
 * The argument m1 is used rather than m so that the logarithmic
 * singularity at m = 1 will be shifted to the origin; this
 * preserves maximum accuracy.
 *
 * K(0) = pi/2.
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,1        30000       1.3e-7      3.4e-8
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * ellpkf domain      x<0, x>1           0.0
 *
 */



/*							exp10f.c
 *
 *	Base 10 exponential function
 *	(Common antilogarithm)
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, exp10f();
 *
 * y = exp10f( 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).
 * A polynomial approximates 10**f.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -38,+38     100000      9.8e-8      2.8e-8
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * exp10 underflow    x < -MAXL10        0.0
 * exp10 overflow     x > MAXL10       MAXNUM
 *
 * IEEE single arithmetic: MAXL10 = 38.230809449325611792.
```

```
 *
 */



/*                                                      exp2f.c
 *
 *      Base 2 exponential function
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, exp2f();
 *
 * y = exp2f( 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 polynomial approximates 2**x in the basic range [-0.5, 0.5].
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     -127,+127    100000      1.7e-7      2.8e-8
 *
 *
 * See exp.c for comments on error amplification.
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * exp underflow    x < -MAXL2        0.0
 * exp overflow     x > MAXL2         MAXNUMF
 *
 * For IEEE arithmetic, MAXL2 = 127.
 */



/*                                                      expf.c
 *
 *      Exponential function
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, expf();
 *
 * y = expf( 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 polynomial is used to approximate exp(f)
 * in the basic range [-0.5, 0.5].
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      +- MAXLOG   100000      1.7e-7      2.8e-8
 *
 *
 * 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 double precision
 * computer number, the result contains amplified roundoff
 * error for large arguments not exactly represented.
 *
 *
 * ERROR MESSAGES:
 *
```

```
 *   message         condition      value returned
 * expf underflow    x < MINLOGF         0.0
 * expf overflow     x > MAXLOGF         MAXNUMF
 *
 */



/*                                                      expnf.c
 *
 *              Exponential integral En
 *
 *
 *
 * SYNOPSIS:
 *
 * int n;
 * float x, y, expnf();
 *
 * y = expnf( n, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Evaluates the exponential integral
 *
 *                 inf.
 *                   -
 *                  | |   -xt
 *                  |    e
 *      E (x)  =    |    ----  dt.
 *       n          |      n
 *                | |     t
 *                 -
 *                  1
 *
 *
 * Both n and x must be nonnegative.
 *
 * The routine employs either a power series, a continued
 * fraction, or an asymptotic formula depending on the
 * relative values of n and x.
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0, 30       10000       5.6e-7      1.2e-7
 *
 */



/*                                                      expx2f.c
 *
 *      Exponential of squared argument
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, expx2f();
 *
 * y = expx2f( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Computes y = exp(x*x) while suppressing error amplification
 * that would ordinarily arise from the inexactness of the argument x*x.
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic    domain     # trials      peak       rms
 *   IEEE      -9.4, 9.4      10^7       1.7e-7     4.7e-8
 *
 */



/*                                                      facf.c
 *
 *      Factorial function
 *
 *
 *
 * SYNOPSIS:
 *
 * float y, facf();
 * int i;
 *
 * y = facf( i );
 *
 *
 *
 * DESCRIPTION:
```

```
 *
 * Returns factorial of i  =  1 * 2 * 3 * ... * i.
 * fac(0) = 1.0.
 *
 * Due to machine arithmetic bounds the largest value of
 * i accepted is 33 in single precision arithmetic.
 * Greater values, or negative ones,
 * produce an error message and return MAXNUM.
 *
 *
 *
 * ACCURACY:
 *
 * For i < 34 the values are simply tabulated, and have
 * full machine accuracy.
 *
 */



/*                                                      fdtrf.c
 *
 *      F distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int df1, df2;
 * float x, y, fdtrf();
 *
 * y = fdtrf( df1, df2, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the area from zero to x under the F density
 * function (also known as Snedcor's density or the
 * variance ratio density).  This is the density
 * of x = (u1/df1)/(u2/df2), where u1 and u2 are random
 * variables having Chi square distributions with df1
 * and df2 degrees of freedom, respectively.
 *
 * The incomplete beta integral is used, according to the
 * formula
 *
 *      P(x) = incbet( df1/2, df2/2, (df1*x/(df2 + df1*x) ).
 *
 *
 * The arguments a and b are greater than zero, and x
 * x is nonnegative.
 * ACCURACY:
 *
 *        Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,100       5000       2.2e-5      1.1e-6
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * fdtrf domain    a<0, b<0, x<0         0.0
 *
 */



/*                                                      fdtrcf()
 *
 *      Complemented F distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int df1, df2;
 * float x, y, fdtrcf();
 *
 * y = fdtrcf( df1, df2, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the area from x to infinity under the F density
 * function (also known as Snedcor's density or the
 * variance ratio density).
 *
 *
 *                       inf.
 *                        -
 *              1        | |  a-1      b-1
 * 1-P(x)  =  ------      |   t    (1-t)    dt
 *            B(a,b)    | |
 *                       -
 *                        x
 *
 * (See fdtr.c.)
 *
 * The incomplete beta integral is used, according to the
```

```
 * formula
 *
 *      P(x) = incbet( df2/2, df1/2, (df2/(df2 + df1*x) ).
 *
 *
 * ACCURACY:
 *
 *        Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,100       5000       7.3e-5      1.2e-5
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * fdtrcf domain   a<0, b<0, x<0         0.0
 *
 */



/*                                                      fdtrif()
 *
 *      Inverse of complemented F distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * float df1, df2, x, y, fdtrif();
 *
 * x = fdtrif( df1, df2, y );
 *
 *
 *
 *
 * DESCRIPTION:
 *
 * Finds the F density argument x such that the integral
 * from x to infinity of the F density is equal to the
 * given probability y.
 *
 * This is accomplished using the inverse beta integral
 * function and the relations
 *
 *      z = incbi( df2/2, df1/2, y )
 *      x = df2 (1-z) / (df1 z).
 *
 * Note: the following relations hold for the inverse of
 * the uncomplemented F distribution:
 *
 *      z = incbi( df1/2, df2/2, y )
 *      x = df2 z / (df1 (1-z)).
 *
 *
 *
 * ACCURACY:
 *
 * arithmetic   domain     # trials      peak         rms
 *        Absolute error:
 *    IEEE       0,100       5000       4.0e-5      3.2e-6
 *        Relative error:
 *    IEEE       0,100       5000       1.2e-3      1.8e-5
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * fdtrif domain  y <= 0 or y > 1       0.0
 *                     v < 1
 *
 */



/*                                                      ceilf()
 *                                                      floorf()
 *                                                      frexpf()
 *                                                      ldexpf()
 *                                                      signbitf()
 *                                                      isnanf()
 *                                                      isfinitef()
 *
 *      Single precision floating point numeric utilities
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y;
 * float ceilf(), floorf(), frexpf(), ldexpf();
 * int signbit(), isnan(), isfinite();
 * int expnt, n;
 *
 * y = floorf(x);
 * y = ceilf(x);
 * y = frexpf( x, &expnt );
 * y = ldexpf( x, n );
 * n = signbit(x);
 * n = isnan(x);
 * n = isfinite(x);
 *
```

```
 *
 *
 * DESCRIPTION:
 *
 * All four routines return a single precision floating point
 * result.
 *
 * sfloor() returns the largest integer less than or equal to x.
 * It truncates toward minus infinity.
 *
 * sceil() returns the smallest integer greater than or equal
 * to x.  It truncates toward plus infinity.
 *
 * sfrexp() 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.
 *
 * ldexpf() multiplies x by 2**n.
 *
 * signbit(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 many but not all C compilers.  The ones supplied are
 * written in C for either DEC or 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.
 */



/*                                                      fresnlf.c
 *
 *      Fresnel integral
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, S, C;
 * void fresnlf();
 *
 * fresnlf( x, &S, &C );
 *
 *
 * DESCRIPTION:
 *
 * Evaluates the Fresnel integrals
 *
 *           x
 *           -
 *          | |
 * C(x) =   |   cos(pi/2 t**2) dt,
 *        | |
 *         -
 *          0
 *
 *           x
 *           -
 *          | |
 * S(x) =   |   sin(pi/2 t**2) dt.
 *        | |
 *         -
 *          0
 *
 *
 * The integrals are evaluated by power series for small x.
 * For x >= 1 auxiliary functions f(x) and g(x) are employed
 * such that
 *
 * C(x) = 0.5 + f(x) sin( pi/2 x**2 ) - g(x) cos( pi/2 x**2 )
 * S(x) = 0.5 - f(x) cos( pi/2 x**2 ) - g(x) sin( pi/2 x**2 )
 *
 *
 *
 * ACCURACY:
 *
 *  Relative error.
 *
 * Arithmetic  function   domain     # trials      peak         rms
 *   IEEE       S(x)      0, 10       30000       1.1e-6      1.9e-7
 *   IEEE       C(x)      0, 10       30000       1.1e-6      2.0e-7
 */



/*                                                      gammaf.c
 *
 *      Gamma function
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, gammaf();
 * extern int sngamf;
 *
```

```
 * y = gammaf( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns gamma function of the argument.  The result is
 * correctly signed, and the sign (+1 or -1) is also
 * returned in a global (extern) variable named sgngamf.
 * This same variable is also filled in by the logarithmic
 * gamma function lgam().
 *
 * Arguments between 0 and 10 are reduced by recurrence and the
 * function is approximated by a polynomial function covering
 * the interval (2,3).  Large arguments are handled by Stirling's
 * formula. Negative arguments are made positive using
 * a reflection formula.
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,-33      100,000     5.7e-7      1.0e-7
 *    IEEE       -33,0      100,000     6.1e-7      1.2e-7
 *
 *
 */



/*                                                      lgamf()
 *
 *      Natural logarithm of gamma function
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, lgamf();
 * extern int sgngamf;
 *
 * y = lgamf( 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 sgngamf.
 *
 * For arguments greater than 6.5, the logarithm of the gamma
 * function is approximated by the logarithmic version of
 * Stirling's formula.  Arguments between 0 and +6.5 are reduced by
 * by recurrence to the interval [.75,1.25] or [1.5,2.5] of a rational
 * approximation.  The cosecant reflection formula is employed for
 * arguments less than zero.
 *
 * Arguments greater than MAXLGM = 2.035093e36 return MAXNUM and an
 * error message.
 *
 *
 *
 * ACCURACY:
 *
 *
 *
 * arithmetic      domain        # trials     peak         rms
 *    IEEE        -100,+100       500,000    7.4e-7       6.8e-8
 * The error criterion was relative when the function magnitude
 * was greater than one but absolute when it was less than one.
 * The routine has low relative error for positive arguments.
 *
 * The following test used the relative error criterion.
 *    IEEE    -2, +3              100000     4.0e-7      5.6e-8
 *
 */



/*                                                      gdtrf.c
 *
 *      Gamma distribution function
 *
 *
 *
 * SYNOPSIS:
 *
 * float a, b, x, y, gdtrf();
 *
 * y = gdtrf( a, b, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the integral from zero to x of the gamma probability
 * density function:
```

```
 *
 *
 *                x
 *        b       -
 *       a       | |   b-1  -at
 * y =  -----    |    t    e    dt
 *       -     | |
 *      | (b)   -
 *               0
 *
 *  The incomplete gamma integral is used, according to the
 * relation
 *
 * y = igam( b, ax ).
 *
 *
 * ACCURACY:
 *
 *        Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,100       5000       5.8e-5      3.0e-6
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * gdtrf domain        x < 0            0.0
 *
 */



/*                                                      gdtrcf.c
 *
 *      Complemented gamma distribution function
 *
 *
 *
 * SYNOPSIS:
 *
 * float a, b, x, y, gdtrcf();
 *
 * y = gdtrcf( a, b, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the integral from x to infinity of the gamma
 * probability density function:
 *
 *
 *               inf.
 *        b       -
 *       a       | |   b-1  -at
 * y =  -----    |    t    e    dt
 *       -     | |
 *      | (b)   -
 *               x
 *
 *  The incomplete gamma integral is used, according to the
 * relation
 *
 * y = igamc( b, ax ).
 *
 *
 * ACCURACY:
 *
 *        Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,100       5000       9.1e-5      1.5e-5
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * gdtrcf domain        x < 0            0.0
 *
 */



/*                                                      hyp2f1f.c
 *
 *      Gauss hypergeometric function   F
 *                                     2 1
 *
 *
 * SYNOPSIS:
 *
 * float a, b, c, x, y, hyp2f1f();
 *
 * y = hyp2f1f( a, b, c, x );
 *
 *
 * DESCRIPTION:
 *
 *
 *  hyp2f1( a, b, c, x )  =   F ( a, b; c; x )
 *                           2 1
 *
 *           inf.
```

```
 *            -   a(a+1)...(a+k) b(b+1)...(b+k)   k+1
 *   =  1 +   >   -----------------------------  x   .
 *            -         c(c+1)...(c+k) (k+1)!
 *          k = 0
 *
 *  Cases addressed are
 *      Tests and escapes for negative integer a, b, or c
 *      Linear transformation if c - a or c - b negative integer
 *      Special case c = a or c = b
 *      Linear transformation for  x near +1
 *      Transformation for x < -0.5
 *      Psi function expansion if x > 0.5 and c - a - b integer
 *      Conditionally, a recurrence on c to make c-a-b > 0
 *
 * |x| > 1 is rejected.
 *
 * The parameters a, b, c are considered to be integer
 * valued if they are within 1.0e-6 of the nearest integer.
 *
 * ACCURACY:
 *
 *                      Relative error (-1 < x < 1):
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,3         30000       5.8e-4      4.3e-6
 */



/*                                                      hypergf.c
 *
 *      Confluent hypergeometric function
 *
 *
 *
 * SYNOPSIS:
 *
 * float a, b, x, y, hypergf();
 *
 * y = hypergf( a, b, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Computes the confluent hypergeometric function
 *
 *                          1           2
 *                       a x    a(a+1) x
 *   F ( a,b;x )  =  1 + ---- + --------- + ...
 *  1 1                  b 1!   b(b+1) 2!
 *
 * Many higher transcendental functions are special cases of
 * this power series.
 *
 * As is evident from the formula, b must not be a negative
 * integer or zero unless a is an integer with 0 >= a > b.
 *
 * The routine attempts both a direct summation of the series
 * and an asymptotic expansion.  In each case error due to
 * roundoff, cancellation, and nonconvergence is estimated.
 * The result with smaller estimated error is returned.
 *
 *
 *
 * ACCURACY:
 *
 * Tested at random points (a, b, x), all three variables
 * ranging from 0 to 30.
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,5         10000       6.6e-7      1.3e-7
 *    IEEE      0,30        30000       1.1e-5      6.5e-7
 *
 * Larger errors can be observed when b is near a negative
 * integer or zero.  Certain combinations of arguments yield
 * serious cancellation error in the power series summation
 * and also are not in the region of near convergence of the
 * asymptotic series.  An error message is printed if the
 * self-estimated relative error is greater than 1.0e-3.
 *
 */



/*                                                      i0f.c
 *
 *      Modified Bessel function of order zero
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, i0();
 *
 * y = i0f( x );
 *
 *
 *
 * DESCRIPTION:
 *
```

```
 * Returns modified Bessel function of order zero of the
 * argument.
 *
 * The function is defined as i0(x) = j0( ix ).
 *
 * The range is partitioned into the two intervals [0,8] and
 * (8, infinity).  Chebyshev polynomial expansions are employed
 * in each interval.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,30        100000      4.0e-7      7.9e-8
 *
 */



/*                                                      i0ef.c
 *
 *      Modified Bessel function of order zero,
 *      exponentially scaled
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, i0ef();
 *
 * y = i0ef( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns exponentially scaled modified Bessel function
 * of order zero of the argument.
 *
 * The function is defined as i0e(x) = exp(-|x|) j0( ix ).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,30        100000      3.7e-7      7.0e-8
 * See i0f().
 *
 */



/*                                                      i1f.c
 *
 *      Modified Bessel function of order one
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, i1f();
 *
 * y = i1f( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns modified Bessel function of order one of the
 * argument.
 *
 * The function is defined as i1(x) = -i j1( ix ).
 *
 * The range is partitioned into the two intervals [0,8] and
 * (8, infinity).  Chebyshev polynomial expansions are employed
 * in each interval.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0, 30       100000      1.5e-6      1.6e-7
 *
 *
 */



/*                                                      i1ef.c
 *
 *      Modified Bessel function of order one,
 *      exponentially scaled
 *
 *
```

```
 *
 * SYNOPSIS:
 *
 * float x, y, i1ef();
 *
 * y = i1ef( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns exponentially scaled modified Bessel function
 * of order one of the argument.
 *
 * The function is defined as i1(x) = -i exp(-|x|) j1( ix ).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0, 30       30000       1.5e-6      1.5e-7
 * See i1().
 *
 */



/*                                                      igamf.c
 *
 *      Incomplete gamma integral
 *
 *
 *
 * SYNOPSIS:
 *
 * float a, x, y, igamf();
 *
 * y = igamf( a, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * The function is defined by
 *
 *                           x
 *                            -
 *                   1       | |  -t  a-1
 *  igam(a,x)  =   -----     |   e   t   dt.
 *                  -      | |
 *                 | (a)    -
 *                           0
 *
 *
 * In this implementation both arguments must be positive.
 * The integral is evaluated by either a power series or
 * continued fraction expansion, depending on the relative
 * values of a and x.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,30        20000       7.8e-6      5.9e-7
 *
 */



/*                                                      igamcf()
 *
 *      Complemented incomplete gamma integral
 *
 *
 *
 * SYNOPSIS:
 *
 * float a, x, y, igamcf();
 *
 * y = igamcf( a, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * The function is defined by
 *
 *
 *  igamc(a,x)   =   1 - igam(a,x)
 *
 *                            inf.
 *                              -
 *                     1       | |  -t  a-1
 *               =   -----     |   e   t   dt.
 *                    -      | |
 *                   | (a)    -
```

```
 *                              x
 *
 *
 * In this implementation both arguments must be positive.
 * The integral is evaluated by either a power series or
 * continued fraction expansion, depending on the relative
 * values of a and x.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,30        30000       7.8e-6      5.9e-7
 *
 */



/*                                                      igamif()
 *
 *      Inverse of complemented imcomplete gamma integral
 *
 *
 *
 * SYNOPSIS:
 *
 * float a, x, y, igamif();
 *
 * x = igamif( a, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Given y, the function finds x such that
 *
 *  igamc( a, x ) = y.
 *
 * It is valid in the right-hand tail of the distribution, y < 0.5.
 * Starting with the approximate value
 *
 *         3
 *  x = a t
 *
 *  where
 *
 *  t = 1 - d - ndtri(y) sqrt(d)
 *
 * and
 *
 *  d = 1/9a,
 *
 * the routine performs up to 10 Newton iterations to find the
 * root of igamc(a,x) - y = 0.
 *
 *
 * ACCURACY:
 *
 * Tested for a ranging from 0 to 100 and x from 0 to 1.
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,100         5000       1.0e-5      1.5e-6
 *
 */



/*                                                      incbetf.c
 *
 *      Incomplete beta integral
 *
 *
 * SYNOPSIS:
 *
 * float a, b, x, y, incbetf();
 *
 * y = incbetf( a, b, x );
 *
 *
 * DESCRIPTION:
 *
 * Returns incomplete beta integral of the arguments, evaluated
 * from zero to x.  The function is defined as
 *
 *                  x
 *     -            -
 *    | (a+b)      | |  a-1     b-1
 *  -----------    |   t   (1-t)   dt.
 *   -     -     | |
 *  | (a) | (b)   -
 *                 0
 *
 * The domain of definition is 0 <= x <= 1.  In this
 * implementation a and b are restricted to positive values.
 * The integral from x to 1 may be obtained by the symmetry
 * relation
 *
```

```
 *    1 - incbet( a, b, x )  =  incbet( b, a, 1-x ).
 *
 * The integral is evaluated by a continued fraction expansion.
 * If a < 1, the function calls itself recursively after a
 * transformation to increase a to a+1.
 *
 * ACCURACY:
 *
 * Tested at random points (a,b,x) with a and b in the indicated
 * interval and x between 0 and 1.
 *
 * arithmetic   domain     # trials      peak         rms
 * Relative error:
 *    IEEE       0,30       10000       3.7e-5      5.1e-6
 *    IEEE       0,100      10000       1.7e-4      2.5e-5
 * The useful domain for relative error is limited by underflow
 * of the single precision exponential function.
 * Absolute error:
 *    IEEE       0,30      100000       2.2e-5      9.6e-7
 *    IEEE       0,100      10000       6.5e-5      3.7e-6
 *
 * Larger errors may occur for extreme ratios of a and b.
 *
 * ERROR MESSAGES:
 *   message         condition      value returned
 * incbetf domain     x<0, x>1          0.0
 */



/*                                                      incbif()
 *
 *      Inverse of imcomplete beta integral
 *
 *
 *
 * SYNOPSIS:
 *
 * float a, b, x, y, incbif();
 *
 * x = incbif( a, b, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Given y, the function finds x such that
 *
 *  incbet( a, b, x ) = y.
 *
 * the routine performs up to 10 Newton iterations to find the
 * root of incbet(a,b,x) - y = 0.
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 *                x     a,b
 * arithmetic   domain  domain  # trials    peak       rms
 *    IEEE      0,1     0,100     5000     2.8e-4    8.3e-6
 *
 * Overflow and larger errors may occur for one of a or b near zero
 *  and the other large.
 */



/*                                                      ivf.c
 *
 *      Modified Bessel function of noninteger order
 *
 *
 *
 * SYNOPSIS:
 *
 * float v, x, y, ivf();
 *
 * y = ivf( v, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns modified Bessel function of order v of the
 * argument.  If x is negative, v must be integer valued.
 *
 * The function is defined as Iv(x) = Jv( ix ).  It is
 * here computed in terms of the confluent hypergeometric
 * function, according to the formula
 *
 *              v  -x
 * Iv(x) = (x/2)  e   hyperg( v+0.5, 2v+1, 2x ) / gamma(v+1)
 *
 * If v is a negative integer, then v is replaced by -v.
 *
 *
 * ACCURACY:
 *
 * Tested at random points (v, x), with v between 0 and
 * 30, x between 0 and 28.
```

```
 * arithmetic   domain     # trials      peak         rms
 *                      Relative error:
 *    IEEE      0,15          3000      4.7e-6      5.4e-7
 *          Absolute error (relative when function > 1)
 *    IEEE      0,30          5000      8.5e-6      1.3e-6
 *
 * Accuracy is diminished if v is near a negative integer.
 * The useful domain for relative error is limited by overflow
 * of the single precision exponential function.
 *
 * See also hyperg.c.
 *
 */



/*                                                      j0f.c
 *
 *      Bessel function of order zero
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, j0f();
 *
 * y = j0f( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of order zero of the argument.
 *
 * The domain is divided into the intervals [0, 2] and
 * (2, infinity). In the first interval the following polynomial
 * approximation is used:
 *
 *
 *        2         2         2
 * (w - r  ) (w - r  ) (w - r  ) P(w)
 *       1         2         3
 *
 *            2
 * where w = x  and the three r's are zeros of the function.
 *
 * In the second interval, the modulus and phase are approximated
 * by polynomials of the form Modulus(x) = sqrt(1/x) Q(1/x)
 * and Phase(x) = x + 1/x R(1/x^2) - pi/4.  The function is
 *
 *   j0(x) = Modulus(x) cos( Phase(x) ).
 *
 *
 *
 * ACCURACY:
 *
 *                      Absolute error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0, 2        100000      1.3e-7      3.6e-8
 *    IEEE      2, 32       100000      1.9e-7      5.4e-8
 *
 */



/*                                                      y0f.c
 *
 *      Bessel function of the second kind, order zero
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, y0f();
 *
 * y = y0f( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of the second kind, of order
 * zero, of the argument.
 *
 * The domain is divided into the intervals [0, 2] and
 * (2, infinity). In the first interval a rational approximation
 * R(x) is employed to compute
 *
 *                  2         2         2
 * y0(x)  =  (w - r  ) (w - r  ) (w - r  ) R(x)  +  2/pi ln(x) j0(x).
 *                 1         2         3
 *
 * Thus a call to j0() is required.  The three zeros are removed
 * from R(x) to improve its numerical stability.
 *
 * In the second interval, the modulus and phase are approximated
 * by polynomials of the form Modulus(x) = sqrt(1/x) Q(1/x)
 * and Phase(x) = x + 1/x S(1/x^2) - pi/4.  Then the function is
 *
 *   y0(x) = Modulus(x) sin( Phase(x) ).
```

```
 *
 *
 *
 *
 * ACCURACY:
 *
 *  Absolute error, when y0(x) < 1; else relative error:
 *
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,  2       100000      2.4e-7      3.4e-8
 *    IEEE      2, 32       100000      1.8e-7      5.3e-8
 *
 */



/*                                                      j1f.c
 *
 *      Bessel function of order one
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, j1f();
 *
 * y = j1f( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of order one of the argument.
 *
 * The domain is divided into the intervals [0, 2] and
 * (2, infinity). In the first interval a polynomial approximation
 *        2
 * (w - r  ) x P(w)
 *       1
 *                     2
 * is used, where w = x  and r is the first zero of the function.
 *
 * In the second interval, the modulus and phase are approximated
 * by polynomials of the form Modulus(x) = sqrt(1/x) Q(1/x)
 * and Phase(x) = x + 1/x R(1/x^2) - 3pi/4.  The function is
 *
 *   j0(x) = Modulus(x) cos( Phase(x) ).
 *
 *
 *
 * ACCURACY:
 *
 *                      Absolute error:
 * arithmetic   domain      # trials      peak       rms
 *    IEEE      0,  2       100000       1.2e-7     2.5e-8
 *    IEEE      2, 32       100000       2.0e-7     5.3e-8
 *
 *
 */



/*                                                      y1
 *
 *      Bessel function of second kind of order one
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, y1();
 *
 * y = y1( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of the second kind of order one
 * of the argument.
 *
 * The domain is divided into the intervals [0, 2] and
 * (2, infinity). In the first interval a rational approximation
 * R(x) is employed to compute
 *
 *                  2
 * y0(x)  =  (w - r  ) x R(x^2)  +  2/pi (ln(x) j1(x) - 1/x) .
 *                 1
 *
 * Thus a call to j1() is required.
 *
 * In the second interval, the modulus and phase are approximated
 * by polynomials of the form Modulus(x) = sqrt(1/x) Q(1/x)
 * and Phase(x) = x + 1/x S(1/x^2) - 3pi/4.  Then the function is
 *
 *   y0(x) = Modulus(x) sin( Phase(x) ).
 *
 *
 *
 *
```

```
 * ACCURACY:
 *
 *                      Absolute error:
 * arithmetic   domain      # trials      peak         rms
 *    IEEE      0,  2       100000       2.2e-7     4.6e-8
 *    IEEE      2, 32       100000       1.9e-7     5.3e-8
 *
 * (error criterion relative when |y1| > 1).
 *
 */



/*                                                      jnf.c
 *
 *      Bessel function of integer order
 *
 *
 *
 * SYNOPSIS:
 *
 * int n;
 * float x, y, jnf();
 *
 * y = jnf( 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   range      # trials      peak         rms
 *    IEEE      0, 15       30000       3.6e-7      3.6e-8
 *
 *
 * Not suitable for large n or x. Use jvf() instead.
 *
 */



/*                                                      jvf.c
 *
 *      Bessel function of noninteger order
 *
 *
 *
 * SYNOPSIS:
 *
 * float v, x, y, jvf();
 *
 * y = jvf( v, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of order v of the argument,
 * where v is real.  Negative x is allowed if v is an integer.
 *
 * Several expansions are included: the ascending power
 * series, the Hankel expansion, and two transitional
 * expansions for large v.  If v is not too large, it
 * is reduced by recurrence to a region of best accuracy.
 *
 * The single precision routine accepts negative v, but with
 * reduced accuracy.
 *
 *
 *
 * ACCURACY:
 * Results for integer v are indicated by *.
 * Error criterion is absolute, except relative when |jv()| > 1.
 *
 * arithmetic     domain      # trials      peak         rms
 *                v      x
 *    IEEE       0,125  0,125   30000      2.0e-6      2.0e-7
 *    IEEE     -17,0    0,125   30000      1.1e-5      4.0e-7
 *    IEEE    -100,0    0,125    3000      1.5e-4      7.8e-6
 */
```

```
/*                                                      k0f.c
 *
 *      Modified Bessel function, third kind, order zero
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, k0f();
 *
 * y = k0f( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns modified Bessel function of the third kind
 * of order zero of the argument.
 *
 * The range is partitioned into the two intervals [0,8] and
 * (8, infinity).  Chebyshev polynomial expansions are employed
 * in each interval.
 *
 *
 *
 * ACCURACY:
 *
 * Tested at 2000 random points between 0 and 8.  Peak absolute
 * error (relative when K0 > 1) was 1.46e-14; rms, 4.26e-15.
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0, 30       30000       7.8e-7      8.5e-8
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 *  K0 domain          x <= 0          MAXNUM
 *
 */



/*                                                      k0ef()
 *
 *      Modified Bessel function, third kind, order zero,
 *      exponentially scaled
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, k0ef();
 *
 * y = k0ef( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns exponentially scaled modified Bessel function
 * of the third kind of order zero of the argument.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0, 30       30000       8.1e-7      7.8e-8
 * See k0().
 *
 */



/*                                                      k1f.c
 *
 *      Modified Bessel function, third kind, order one
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, k1f();
 *
 * y = k1f( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Computes the modified Bessel function of the third kind
 * of order one of the argument.
 *
 * The range is partitioned into the two intervals [0,2] and
 * (2, infinity).  Chebyshev polynomial expansions are employed
 * in each interval.
 *
 *
 *
```

```
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0, 30       30000       4.6e-7      7.6e-8
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * k1 domain          x <= 0          MAXNUM
 *
 */



/*							k1ef.c
 *
 *	Modified Bessel function, third kind, order one,
 *	exponentially scaled
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, k1ef();
 *
 * y = k1ef( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns exponentially scaled modified Bessel function
 * of the third kind of order one of the argument:
 *
 *      k1e(x) = exp(x) * k1(x).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0, 30       30000       4.9e-7      6.7e-8
 * See k1().
 *
 */



/*							knf.c
 *
 *	Modified Bessel function, third kind, integer order
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, knf();
 * int n;
 *
 * y = knf( n, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns modified Bessel function of the third kind
 * of order n of the argument.
 *
 * The range is partitioned into the two intervals [0,9.55] and
 * (9.55, infinity).  An ascending power series is used in the
 * low range, and an asymptotic expansion in the high range.
 *
 *
 *
 * ACCURACY:
 *
 *          Absolute error, relative when function > 1:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,30        10000       2.0e-4      3.8e-6
 *
 *  Error is high only near the crossover point x = 9.55
 * between the two expansions used.
 */



/*							log10f.c
 *
 *	Common logarithm
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, log10f();
 *
 * y = log10f( x );
 *
```

```
 *
 *
 * DESCRIPTION:
 *
 * Returns logarithm to the base 10 of x.
 *
 * The argument is separated into its exponent and fractional
 * parts.  The logarithm of the fraction is approximated by
 *
 *     log(1+x) = x - 0.5 x**2 + x**3 P(x).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0.5, 2.0    100000      1.3e-7      3.4e-8
 *    IEEE      0, MAXNUMF  100000      1.3e-7      2.6e-8
 *
 * In the tests over the interval [0, MAXNUM], the logarithms
 * of the random arguments were uniformly distributed over
 * [-MAXL10, MAXL10].
 *
 * ERROR MESSAGES:
 *
 * log10f singularity:  x = 0; returns -MAXL10
 * log10f domain:       x < 0; returns -MAXL10
 * MAXL10 = 38.230809449325611792
 */



/*                                                      log2f.c
 *
 *      Base 2 logarithm
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, log2f();
 *
 * y = log2f( 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 base e
 * 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(+-88)   100000      1.1e-7      2.4e-8
 *    IEEE      0.5, 2.0    100000      1.1e-7      3.0e-8
 *
 * In the tests over the interval [exp(+-88)], the logarithms
 * of the random arguments were uniformly distributed.
 *
 * ERROR MESSAGES:
 *
 * log singularity:  x = 0; returns MINLOGF/log(2)
 * log domain:       x < 0; returns MINLOGF/log(2)
 */



/*                                                      logf.c
 *
 *      Natural logarithm
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, logf();
 *
 * y = logf( 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)
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0.5, 2.0    100000       7.6e-8     2.7e-8
 *    IEEE      1, MAXNUMF  100000                  2.6e-8
 *
 * In the tests over the interval [1, MAXNUM], the logarithms
 * of the random arguments were uniformly distributed over
 * [0, MAXLOGF].
 *
 * ERROR MESSAGES:
 *
 * logf singularity:  x = 0; returns MINLOG
 * logf domain:       x < 0; returns MINLOG
 */



/*                                                      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
 *
 */



/*                                                      nbdtrf.c
 *
 *      Negative binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k, n;
 * float p, y, nbdtrf();
 *
 * y = nbdtrf( k, n, p );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the sum of the terms 0 through k of the negative
 * binomial distribution:
 *
 *   k
 *   --  ( n+j-1 )   n      j
```

```
 *   >   (       )  p  (1-p)
 *   --  (   j   )
 *  j=0
 *
 * In a sequence of Bernoulli trials, this is the probability
 * that k or fewer failures precede the nth success.
 *
 * The terms are not computed individually; instead the incomplete
 * beta integral is employed, according to the formula
 *
 * y = nbdtr( k, n, p ) = incbet( n, k+1, p ).
 *
 * The arguments must be positive, with p ranging from 0 to 1.
 *
 *
 *
 * ACCURACY:
 *
 *        Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,100       5000       1.5e-4      1.9e-5
 *
 */



/*                                                      nbdtrcf.c
 *
 *      Complemented negative binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k, n;
 * float p, y, nbdtrcf();
 *
 * y = nbdtrcf( k, n, p );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the sum of the terms k+1 to infinity of the negative
 * binomial distribution:
 *
 *   inf
 *   --  ( n+j-1 )   n      j
 *   >   (       )  p  (1-p)
 *   --  (   j   )
 *  j=k+1
 *
 * The terms are not computed individually; instead the incomplete
 * beta integral is employed, according to the formula
 *
 * y = nbdtrc( k, n, p ) = incbet( k+1, n, 1-p ).
 *
 * The arguments must be positive, with p ranging from 0 to 1.
 *
 *
 *
 * ACCURACY:
 *
 *        Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,100       5000       1.4e-4      2.0e-5
 *
 */



/*                                                      ndtrf.c
 *
 *      Normal distribution function
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, ndtrf();
 *
 * y = ndtrf( 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.
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     -13,0        50000       1.5e-5      2.6e-6
 *
 *
 * ERROR MESSAGES:
 *
 * See erfcf().
 *
 */



/*                                                      erff.c
 *
 *      Error function
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, erff();
 *
 * y = erff( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * The integral is
 *
 *                           x
 *                            -
 *                 2         | |          2
 *   erf(x)  =  --------     |    exp( - t  ) dt.
 *              sqrt(pi)   | |
 *                          -
 *                           0
 *
 * The magnitude of x is limited to 9.231948545 for DEC
 * arithmetic; 1 or -1 is returned outside this range.
 *
 * For 0 <= |x| < 1, erf(x) = x * P(x**2); otherwise
 * erf(x) = 1 - erfc(x).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -9.3,9.3    50000       1.7e-7      2.8e-8
 *
 */



/*                                                      erfcf.c
 *
 *      Complementary error function
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, erfcf();
 *
 * y = erfcf( 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 polynomial
 * approximations 1/x P(1/x**2) are computed.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
```

```
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -9.3,9.3    50000       3.9e-6      7.2e-7
 *
 *
 * ERROR MESSAGES:
 *
 *   message           condition              value returned
 * erfcf underflow    x**2 > MAXLOGF              0.0
 *
 *
 */



/*                                                      ndtrif.c
 *
 *      Inverse of Normal distribution function
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, ndtrif();
 *
 * x = ndtrif( y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the argument, x, for which the area under the
 * Gaussian probability density function (integrated from
 * minus infinity to x) is equal to y.
 *
 *
 * For small arguments 0 < y < exp(-2), the program computes
 * z = sqrt( -2.0 * log(y) );  then the approximation is
 * x = z - log(z)/z  - (1/z) P(1/z) / Q(1/z).
 * There are two rational functions P/Q, one for 0 < y < exp(-32)
 * and the other for y up to exp(-2).  For larger arguments,
 * w = y - 0.5, and  x/sqrt(2pi) = w + w**3 R(w**2)/S(w**2)).
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain        # trials      peak         rms
 *    IEEE     1e-38, 1        30000       3.6e-7      5.0e-8
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition    value returned
 * ndtrif domain      x <= 0        -MAXNUM
 * ndtrif domain      x >= 1         MAXNUM
 *
 */



/*                                                      pdtrf.c
 *
 *      Poisson distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k;
 * float m, y, pdtrf();
 *
 * y = pdtrf( k, m );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the sum of the first k terms of the Poisson
 * distribution:
 *
 *   k         j
 *   --   -m  m
 *   >   e    --
 *   --       j!
 *  j=0
 *
 * The terms are not summed directly; instead the incomplete
 * gamma integral is employed, according to the relation
 *
 * y = pdtr( k, m ) = igamc( k+1, m ).
 *
 * The arguments must both be positive.
 *
 *
 *
 * ACCURACY:
 *
 *        Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,100       5000       6.9e-5      8.0e-6
```

```
 *
 */



/*                                                      pdtrcf()
 *
 *      Complemented poisson distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k;
 * float m, y, pdtrcf();
 *
 * y = pdtrcf( k, m );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the sum of the terms k+1 to infinity of the Poisson
 * distribution:
 *
 *  inf.       j
 *   --   -m  m
 *   >   e    --
 *   --       j!
 *  j=k+1
 *
 * The terms are not summed directly; instead the incomplete
 * gamma integral is employed, according to the formula
 *
 * y = pdtrc( k, m ) = igam( k+1, m ).
 *
 * The arguments must both be positive.
 *
 *
 *
 * ACCURACY:
 *
 *        Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,100       5000       8.4e-5      1.2e-5
 *
 */



/*                                                      pdtrif()
 *
 *      Inverse Poisson distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k;
 * float m, y, pdtrf();
 *
 * m = pdtrif( k, y );
 *
 *
 *
 *
 * DESCRIPTION:
 *
 * Finds the Poisson variable x such that the integral
 * from 0 to x of the Poisson density is equal to the
 * given probability y.
 *
 * This is accomplished using the inverse gamma integral
 * function and the relation
 *
 *    m = igami( k+1, y ).
 *
 *
 *
 *
 * ACCURACY:
 *
 *        Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,100       5000       8.7e-6      1.4e-6
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * pdtri domain    y < 0 or y >= 1       0.0
 *                     k < 0
 *
 */



/*                                                      polevlf.c
 *                                                      p1evlf.c
 *
 *      Evaluate polynomial
```

```
 *
 *
 *
 * SYNOPSIS:
 *
 * int N;
 * float x, y, coef[N+1], polevlf[];
 *
 * y = polevlf( 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 p1evl() assumes that coef[N] = 1.0 and is
 * omitted from the array.  Its calling arguments are
 * otherwise the same as polevl().
 *
 *
 * 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.
 *
 */



/*                                                      polynf.c
 *                                                      polyrf.c
 * Arithmetic operations on polynomials
 *
 * In the following descriptions a, b, c are polynomials of degree
 * na, nb, nc respectively.  The degree of a polynomial cannot
 * exceed a run-time value MAXPOLF.  An operation that attempts
 * to use or generate a polynomial of higher degree may produce a
 * result that suffers truncation at degree MAXPOL.  The value of
 * MAXPOL is set by calling the function
 *
 *     polinif( maxpol );
 *
 * where maxpol is the desired maximum degree.  This must be
 * done prior to calling any of the other functions in this module.
 * Memory for internal temporary polynomial storage is allocated
 * by polinif().
 *
 * Each polynomial is represented by an array containing its
 * coefficients, together with a separately declared integer equal
 * to the degree of the polynomial.  The coefficients appear in
 * ascending order; that is,
 *
 *                                        2                      na
 * a(x)  =  a[0]  +  a[1] * x  +  a[2] * x   +  ...  +  a[na] * x  .
 *
 *
 *
 * sum = poleva( a, na, x );    Evaluate polynomial a(t) at t = x.
 * polprtf( a, na, D );         Print the coefficients of a to D digits.
 * polclrf( a, na );            Set a identically equal to zero, up to a[na].
 * polmovf( a, na, b );         Set b = a.
 * poladdf( a, na, b, nb, c );  c = b + a, nc = max(na,nb)
 * polsubf( a, na, b, nb, c );  c = b - a, nc = max(na,nb)
 * polmulf( a, na, b, nb, c );  c = b * a, nc = na+nb
 *
 *
 * Division:
 *
 * i = poldivf( a, na, b, nb, c );      c = b / a, nc = MAXPOL
 *
 * returns i = the degree of the first nonzero coefficient of a.
 * The computed quotient c must be divided by x^i.  An error message
 * is printed if a is identically zero.
 *
 *
 * Change of variables:
 * If a and b are polynomials, and t = a(x), then
 *     c(t) = b(a(x))
 * is a polynomial found by substituting a(x) for t.  The
 * subroutine call for this is
 *
 * polsbtf( a, na, b, nb, c );
 *
 *
 * Notes:
 * poldivf() is an integer routine; polevaf() is float.
 * Any of the arguments a, b, c may refer to the same array.
```

```
 *
 */



/*                                                      powf.c
 *
 *      Power function
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, z, powf();
 *
 * z = powf( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Computes x raised to the yth power.  Analytically,
 *
 *      x**y  =  exp( y log(x) ).
 *
 * Following Cody and Waite, this program uses a lookup table
 * of 2**-i/16 and pseudo extended precision arithmetic to
 * obtain an extra three bits of accuracy in both the logarithm
 * and the exponential.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 *  arithmetic  domain     # trials      peak         rms
 *    IEEE     -10,10      100,000      1.4e-7      3.6e-8
 * 1/10 < x < 10, x uniformly distributed.
 * -10 < y < 10, y uniformly distributed.
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * powf overflow     x**y > MAXNUMF     MAXNUMF
 * powf underflow   x**y < 1/MAXNUMF      0.0
 * powf domain      x<0 and y noninteger  0.0
 *
 */



/*                                                      powif.c
 *
 *      Real raised to integer power
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, powif();
 * int n;
 *
 * y = powif( 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      .04,26     -26,26    100000       1.1e-6      2.0e-7
 *    IEEE        1,2      -128,128  100000       1.1e-5      1.0e-6
 *
 * Returns MAXNUMF on overflow, zero on underflow.
 *
 */



/*                                                      psif.c
 *
 *      Psi (digamma) function
 *
 *
 * SYNOPSIS:
 *
 * float x, y, psif();
 *
```

```
 * y = psif( x );
 *
 *
 * DESCRIPTION:
 *
 *              d      -
 *   psi(x)  =  -- ln | (x)
 *              dx
 *
 * is the logarithmic derivative of the gamma function.
 * For integer x,
 *                   n-1
 *                    -
 * psi(n) = -EUL  +   >  1/k.
 *                    -
 *                   k=1
 *
 * This formula is used for 0 < n <= 10.  If x is negative, it
 * is transformed to a positive argument by the reflection
 * formula  psi(1-x) = psi(x) + pi cot(pi x).
 * For general positive x, the argument is made greater than 10
 * using the recurrence  psi(x+1) = psi(x) + 1/x.
 * Then the following asymptotic expansion is applied:
 *
 *                           inf.   B
 *                            -      2k
 * psi(x) = log(x) - 1/2x -   >   -------
 *                            -        2k
 *                           k=1   2k x
 *
 * where the B2k are Bernoulli numbers.
 *
 * ACCURACY:
 *    Absolute error,  relative when |psi| > 1 :
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -33,0        30000      8.2e-7      1.2e-7
 *    IEEE      0,33        100000      7.3e-7      7.7e-8
 *
 * ERROR MESSAGES:
 *     message         condition      value returned
 * psi singularity    x integer <=0      MAXNUMF
 */



/*                                              rgammaf.c
 *
 *      Reciprocal gamma function
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, rgammaf();
 *
 * y = rgammaf( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns one divided by the gamma function of the argument.
 *
 * The function is approximated by a Chebyshev expansion in
 * the interval [0,1].  Range reduction is by recurrence
 * for arguments between -34.034 and +34.84425627277176174.
 * 1/MAXNUMF is returned for positive arguments outside this
 * range.
 *
 * The reciprocal gamma function has no singularities,
 * but overflow and underflow may occur for large arguments.
 * These conditions return either MAXNUMF or 1/MAXNUMF with
 * appropriate sign.
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     -34,+34      100000      8.9e-7      1.1e-7
 */



/*                                                      shichif.c
 *
 *      Hyperbolic sine and cosine integrals
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, Chi, Shi;
 *
 * shichi( x, &Chi, &Shi );
 *
 *
 * DESCRIPTION:
 *
 * Approximates the integrals
 *
```

```
 *                             x
 *                             -
 *                            | |   cosh t - 1
 *   Chi(x) = eul + ln x +    |    -----------  dt,
 *                          | |          t
 *                           -
 *                           0
 *
 *               x
 *               -
 *              | |  sinh t
 *   Shi(x) =   |    ------  dt
 *            | |       t
 *             -
 *             0
 *
 * where eul = 0.57721566490153286061 is Euler's constant.
 * The integrals are evaluated by power series for x < 8
 * and by Chebyshev expansions for x between 8 and 88.
 * For large x, both functions approach exp(x)/2x.
 * Arguments greater than 88 in magnitude return MAXNUM.
 *
 *
 * ACCURACY:
 *
 * Test interval 0 to 88.
 *                      Relative error:
 * arithmetic   function  # trials      peak         rms
 *    IEEE         Shi      20000       3.5e-7      7.0e-8
 *        Absolute error, except relative when |Chi| > 1:
 *    IEEE         Chi      20000       3.8e-7      7.6e-8
 */



/*                                                      sicif.c
 *
 *      Sine and cosine integrals
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, Ci, Si;
 *
 * sicif( x, &Si, &Ci );
 *
 *
 * DESCRIPTION:
 *
 * Evaluates the integrals
 *
 *                           x
 *                           -
 *                          |  cos t - 1
 *   Ci(x) = eul + ln x +   |  --------- dt,
 *                          |      t
 *                         -
 *                          0
 *             x
 *             -
 *            |  sin t
 *   Si(x) =  |  ----- dt
 *            |    t
 *           -
 *            0
 *
 * where eul = 0.57721566490153286061 is Euler's constant.
 * The integrals are approximated by rational functions.
 * For x > 8 auxiliary functions f(x) and g(x) are employed
 * such that
 *
 * Ci(x) = f(x) sin(x) - g(x) cos(x)
 * Si(x) = pi/2 - f(x) cos(x) - g(x) sin(x)
 *
 *
 * ACCURACY:
 *    Test interval = [0,50].
 * Absolute error, except relative when > 1:
 * arithmetic   function   # trials      peak         rms
 *    IEEE        Si        30000       2.1e-7      4.3e-8
 *    IEEE        Ci        30000       3.9e-7      2.2e-8
 */



/*                                                      sindgf.c
 *
 *      Circular sine of angle in degrees
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, sindgf();
 *
 * y = sindgf( x );
 *
 *
 *
```

```
 * DESCRIPTION:
 *
 * Range reduction is into intervals of 45 degrees.
 *
 * Two polynomial approximating functions are employed.
 * Between 0 and pi/4 the sine is approximated by
 *      x  +  x**3 P(x**2).
 * Between pi/4 and pi/2 the cosine is represented as
 *      1  -  x**2 Q(x**2).
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain      # trials      peak       rms
 *    IEEE      +-3600      100,000      1.2e-7     3.0e-8
 *
 * ERROR MESSAGES:
 *
 *   message           condition        value returned
 * sin total loss      x > 2^24              0.0
 *
 */



/*                                                      cosdgf.c
 *
 *      Circular cosine of angle in degrees
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, cosdgf();
 *
 * y = cosdgf( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Range reduction is into intervals of 45 degrees.
 *
 * Two polynomial approximating functions are employed.
 * Between 0 and pi/4 the cosine is approximated by
 *      1  -  x**2 Q(x**2).
 * Between pi/4 and pi/2 the sine is represented as
 *      x  +  x**3 P(x**2).
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain      # trials      peak         rms
 *    IEEE    -8192,+8192   100,000      3.0e-7     3.0e-8
 */



/*                                                      sinf.c
 *
 *      Circular sine
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, sinf();
 *
 * y = sinf( 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
 *      x  +  x**3 P(x**2).
 * Between pi/4 and pi/2 the cosine is represented as
 *      1  -  x**2 Q(x**2).
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain      # trials      peak       rms
 *    IEEE    -4096,+4096   100,000      1.2e-7     3.0e-8
 *    IEEE    -8192,+8192   100,000      3.0e-7     3.0e-8
 *
 * ERROR MESSAGES:
 *
 *   message           condition        value returned
 * sin total loss      x > 2^24              0.0
 *
 * Partial loss of accuracy begins to occur at x = 2^13
```

```
 * = 8192. Results may be meaningless for x >= 2^24
 * The routine as implemented flags a TLOSS error
 * for x >= 2^24 and returns 0.0.
 */



/*                                                      cosf.c
 *
 *      Circular cosine
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, cosf();
 *
 * y = cosf( 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  -  x**2 Q(x**2).
 * Between pi/4 and pi/2 the sine is represented as
 *      x  +  x**3 P(x**2).
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain      # trials      peak         rms
 *    IEEE    -8192,+8192   100,000      3.0e-7     3.0e-8
 */



/*                                                      sinhf.c
 *
 *      Hyperbolic sine
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, sinhf();
 *
 * y = sinhf( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns hyperbolic sine of argument in the range MINLOGF to
 * MAXLOGF.
 *
 * The range is partitioned into two segments.  If |x| <= 1, a
 * polynomial approximation is used.
 * Otherwise the calculation is sinh(x) = ( exp(x) - exp(-x) )/2.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     +-MAXLOG     100000      1.1e-7      2.9e-8
 *
 */



/*                                                      spencef.c
 *
 *      Dilogarithm
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, spencef();
 *
 * y = spencef( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Computes the integral
 *
 *                    x
 *                    -
 *                   | | log t
 * spence(x)  =  -   |   ----- dt
```

```
 *                 | |   t - 1
 *                  -
 *                  1
 *
 * for x >= 0.  A rational approximation gives the integral in
 * the interval (0.5, 1.5).  Transformation formulas for 1/x
 * and 1-x are employed outside the basic expansion range.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,4         30000       4.4e-7      6.3e-8
 *
 *
 */



/*                                                      sqrtf.c
 *
 *      Square root
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, sqrtf();
 *
 * y = sqrtf( 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.
 *
 *
 *
 * ACCURACY:
 *
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,1.e38     100000       8.7e-8     2.9e-8
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * sqrtf domain        x < 0            0.0
 *
 */



/*                                                      stdtrf.c
 *
 *      Student's t distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * float t, stdtrf();
 * short k;
 *
 * y = stdtrf( k, t );
 *
 *
 * DESCRIPTION:
 *
 * Computes the integral from minus infinity to t of the Student
 * t distribution with integer k > 0 degrees of freedom:
 *
 *                                      t
 *                                      -
 *                                     | |
 *              -                      |         2   -(k+1)/2
 *             | ( (k+1)/2 )           |  (     x   )
 *       ----------------------        |  ( 1 + --- )        dx
 *                     -               |  (      k  )
 *       sqrt( k pi ) | ( k/2 )        |
 *                                   | |
 *                                    -
 *                                   -inf.
 *
 * Relation to incomplete beta integral:
 *
 *        1 - stdtr(k,t) = 0.5 * incbet( k/2, 1/2, z )
 * where
 *        z = k/(k + t**2).
 *
```

```
 * For t < -1, this is the method of computation.  For higher t,
 * a direct method is derived from integration by parts.
 * Since the function is symmetric about t=0, the area under the
 * right tail of the density is found by calling the function
 * with -t instead of t.
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      +/- 100      5000       2.3e-5      2.9e-6
 */



/*                                                      struvef.c
 *
 *      Struve function
 *
 *
 *
 * SYNOPSIS:
 *
 * float v, x, y, struvef();
 *
 * y = struvef( v, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Computes the Struve function Hv(x) of order v, argument x.
 * Negative x is rejected unless v is an integer.
 *
 * This module also contains the hypergeometric functions 1F2
 * and 3F0 and a routine for the Bessel function Yv(x) with
 * noninteger v.
 *
 *
 *
 * ACCURACY:
 *
 *  v varies from 0 to 10.
 *    Absolute error (relative error when |Hv(x)| > 1):
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10,10      100000      9.0e-5      4.0e-6
 *
 */



/*                                                      tandgf.c
 *
 *      Circular tangent of angle in degrees
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, tandgf();
 *
 * y = tandgf( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the circular tangent of the radian argument x.
 *
 * Range reduction is into intervals of 45 degrees.
 *
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     +-2^24       50000       2.4e-7      4.8e-8
 *
 * ERROR MESSAGES:
 *
 *   message         condition          value returned
 * tanf total loss   x > 2^24              0.0
 *
 */



/*                                                      cotdgf.c
 *
 *      Circular cotangent of angle in degrees
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, cotdgf();
 *
 * y = cotdgf( x );
```

```
 *
 *
 *
 * DESCRIPTION:
 *
 * Range reduction is into intervals of 45 degrees.
 * A common routine computes either the tangent or cotangent.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     +-2^24       50000       2.4e-7      4.8e-8
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition          value returned
 * cot total loss   x > 2^24                0.0
 * cot singularity  x = 0                  MAXNUMF
 *
 */



/*                                                      tanf.c
 *
 *      Circular tangent
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, tanf();
 *
 * y = tanf( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the circular tangent of the radian argument x.
 *
 * Range reduction is modulo pi/4.  A polynomial approximation
 * is employed in the basic interval [0, pi/4].
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     +-4096        100000     3.3e-7      4.5e-8
 *
 * ERROR MESSAGES:
 *
 *   message         condition          value returned
 * tanf total loss   x > 2^24              0.0
 *
 */



/*                                                      cotf.c
 *
 *      Circular cotangent
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, cotf();
 *
 * y = cotf( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the circular cotangent of the radian argument x.
 * A common routine computes either the tangent or cotangent.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     +-4096        100000     3.0e-7      4.5e-8
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition          value returned
 * cot total loss   x > 2^24                0.0
 * cot singularity  x = 0                  MAXNUMF
 *
 */
```

```
/*                                                      tanhf.c
 *
 *      Hyperbolic tangent
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, tanhf();
 *
 * y = tanhf( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns hyperbolic tangent of argument in the range MINLOG to
 * MAXLOG.
 *
 * A polynomial approximation is used for |x| < 0.625.
 * Otherwise,
 *
 *    tanh(x) = sinh(x)/cosh(x) = 1  -  2/(exp(2x) + 1).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -2,2        100000      1.3e-7      2.6e-8
 *
 */



/*                                                      ynf.c
 *
 *      Bessel function of second kind of integer order
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, ynf();
 * int n;
 *
 * y = ynf( n, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of order n, where n is a
 * (possibly negative) integer.
 *
 * The function is evaluated by forward recurrence on
 * n, starting with values computed by the routines
 * y0() and y1().
 *
 * If n = 0 or 1 the routine for y0 or y1 is called
 * directly.
 *
 *
 *
 * ACCURACY:
 *
 *
 *  Absolute error, except relative when y > 1:
 *
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0, 30       10000       2.3e-6      3.4e-7
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * yn singularity   x = 0              MAXNUMF
 * yn overflow                         MAXNUMF
 *
 * Spot checked against tables for x, n between 0 and 100.
 *
 */



/*                                                      zetacf.c
 *
 *      Riemann zeta function
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, y, zetacf();
 *
 * y = zetacf( x );
 *
```

```
 *
 *
 * DESCRIPTION:
 *
 *
 *
 *               inf.
 *                -    -x
 *   zetac(x)  =   >   k   ,   x > 1,
 *                -
 *               k=2
 *
 * is related to the Riemann zeta function by
 *
 *      Riemann zeta(x) = zetac(x) + 1.
 *
 * Extension of the function definition for x < 1 is implemented.
 * Zero is returned for x > log2(MAXNUM).
 *
 * An overflow error may occur for large negative x, due to the
 * gamma function in the reflection formula.
 *
 * ACCURACY:
 *
 * Tabulated values have full machine accuracy.
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      1,50        30000       5.5e-7      7.5e-8
 *
 *
 */



/*                                                      zetaf.c
 *
 *      Riemann zeta function of two arguments
 *
 *
 *
 * SYNOPSIS:
 *
 * float x, q, y, zetaf();
 *
 * y = zetaf( x, q );
 *
 *
 *
 * DESCRIPTION:
 *
 *
 *
 *                inf.
 *                 -        -x
 *   zeta(x,q)  =   >   (k+q)
 *                 -
 *                k=0
 *
 * where x > 1 and q is not a negative integer or zero.
 * The Euler-Maclaurin summation formula is used to obtain
 * the expansion
 *
 *               n
 *               -       -x
 * zeta(x,q)  =   >  (k+q)
 *               -
 *              k=1
 *
 *           1-x                 inf.  B   x(x+1)...(x+2j)
 *      (n+q)           1         -     2j
 *  +  ---------  -  -------  +   >    --------------------
 *        x-1              x      -                   x+2j+1
 *                   2(n+q)      j=1       (2j)! (n+q)
 *
 * where the B2j are Bernoulli numbers.  Note that (see zetac.c)
 * zeta(x,1) = zetac(x) + 1.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,25        10000       6.9e-7      1.0e-7
 *
 * Large arguments may produce underflow in powf(), in which
 * case the results are inaccurate.
 *
 * REFERENCE:
 *
 * Gradshteyn, I. S., and I. M. Ryzhik, Tables of Integrals,
 * Series, and Products, p. 1073; Academic Press, 1980.
 *
 */
```

[To Cephes home page www.moshier.net](http://www.moshier.net):

Last update: 5 October 2014