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.

Long Double Precision Special Functions

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

- acoshl, Inverse hyperbolic cosine
- arcdotl, Angle between two vectors
- asinh, Inverse hyperbolic sine
- asin, Inverse circular sine
- acos, Inverse circular cosine
- atanh, Inverse hyperbolic tangent
- atan, Inverse circular tangent
- atan2, Quadrant correct inverse circular tangent
- bdtr, Binomial distribution
- bdtrc, Complemented binomial distribution
- bdtri, Inverse binomial distribution
- btdtr, Beta distribution
- cbrt, Cube root
- chdtr, Chi-square distribution
- chdtrc, Complemented Chi-square distribution
- chdtri, Inverse of complemented Chi-square distribution
- clog, Complex natural logarithm
- cexp, Complex exponential function
- csin, Complex circular sine
- ccos, Complex circular cosine
- ctan, Complex circular tangent
- ccot, Complex circular cotangent
- casin, Complex circular arc sine
- cacos, Complex circular arc cosine
- catan, Complex circular arc tangent
- cmplx, Complex number arithmetic
- cosh, Hyperbolic cosine
- ellie, Incomplete elliptic integral of the second kind
- ellik, Incomplete elliptic integral of the first kind
- ellpe, Complete elliptic integral of the second kind
- ellpj, Jacobian elliptic functions
- ellpk, Complete elliptic integral of the first kind
- exp10, Base 10 exponential function
- exp2, Base 2 exponential function
- exp, Exponential function
- expm1, Exponential function, minus 1
- expx2, Exponential function
- fdtr, F distribution
- fdtrc, Complemented F distribution
- fdtri, Inverse of complemented F distribution
- floor, Floor function
- ceil, Ceil function
- frexp, Extract exponent
- ldexp, Apply exponent
- fabs, Absolute value
- gamma, Gamma function
- lgam, Natural logarithm of gamma function
- gdtr, Gamma distribution function
- gdtrc, Complemented gamma distribution function
- gels, Linear system with symmetric coefficient matrix
- hyperg, Confluent hypergeometric function
- ieee, Extended precision arithmetic
- igami, Inverse of complemented imcomplete gamma integral
- igam, Incomplete gamma integral
- igamc, Complemented incomplete gamma integral
- incbet, Incomplete beta integral
- incbi, Inverse of imcomplete beta integral
- isnan, Test for not a number
- isfinite, Test for infinity
- signbit, Extract sign
- j0, Bessel function of order zero
- y0, Bessel function of the second kind, order zero
- j1, Bessel function of order one
- y1, Bessel function of the second kind, order one
- jn, Bessel function of integer order
- ldrand, Pseudorandom number generator
- log10, Common logarithm

```
/*                                                      acoshl.c
 *
 *      Inverse hyperbolic cosine, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, acoshl();
 *
 * y = acoshl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns inverse hyperbolic cosine of argument.
 *
 * If 1 <= x < 1.5, a rational approximation
 *
 *      sqrt(2z) * P(z)/Q(z)
 *
 * where z = x-1, is used.  Otherwise,
 *
 * acosh(x)  =  log( x + sqrt( (x-1)(x+1) ).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      1,3         30000       2.0e-19     3.9e-20
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * acoshl domain      |x| < 1            0.0
 *
 */
```

```
/*                                                      arcdot.c
 *
 *      Angle between two vectors
 *
 *
 *
 *
 * SYNOPSIS:
 *
 * long double p[3], q[3], arcdotl();
 *
 * y = arcdotl( p, q );
 *
 *
 *
 * DESCRIPTION:
 *
 * For two vectors p, q, the angle A between them is given by
```

```
 *
 *      p.q / (|p| |q|)  = cos A  .
 *
 * where "." represents inner product, "|x|" the length of vector x.
 * If the angle is small, an expression in sin A is preferred.
 * Set r = q - p.  Then
 *
 *     p.q = p.p + p.r ,
 *
 *     |p|^2 = p.p ,
 *
 *     |q|^2 = p.p + 2 p.r + r.r ,
 *
 *                  p.p^2 + 2 p.p p.r + p.r^2
 *     cos^2 A  =  ----------------------------
 *                    p.p (p.p + 2 p.r + r.r)
 *
 *                  p.p + 2 p.r + p.r^2 / p.p
 *              =  --------------------------- ,
 *                     p.p + 2 p.r + r.r
 *
 *     sin^2 A  =  1 - cos^2 A
 *
 *                   r.r - p.r^2 / p.p
 *              =  --------------------
 *                  p.p + 2 p.r + r.r
 *
 *              =   (r.r - p.r^2 / p.p) / q.q  .
 *
 * ACCURACY:
 *
 * About 1 ULP.  See arcdot.c.
 *
 */



/*                                                      asinhl.c
 *
 *      Inverse hyperbolic sine, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, asinhl();
 *
 * y = asinhl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns inverse hyperbolic sine of argument.
 *
 * If |x| < 0.5, the function is approximated by a rational
 * form  x + x**3 P(x)/Q(x).  Otherwise,
 *
 *     asinh(x) = log( x + sqrt(1 + x*x) ).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     -3,3         30000       1.7e-19     3.5e-20
 *
 */



/*                                                      asinl.c
 *
 *      Inverse circular sine, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, asinl();
 *
 * y = asinl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns radian angle between -pi/2 and +pi/2 whose sine is x.
 *
 * A rational function of the form x + x**3 P(x**2)/Q(x**2)
 * is used for |x| in the interval [0, 0.5].  If |x| > 0.5 it is
 * transformed by the identity
 *
 *    asin(x) = pi/2 - 2 asin( sqrt( (1-x)/2 ) ).
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
```

```
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     -1, 1        30000       2.7e-19     4.8e-20
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * asinl domain        |x| > 1           NANL
 *
 */



/*                                                      acosl()
 *
 *      Inverse circular cosine, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, acosl();
 *
 * y = acosl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns radian angle between -pi/2 and +pi/2 whose cosine
 * is x.
 *
 * Analytically, acos(x) = pi/2 - asin(x).  However if |x| is
 * near 1, there is cancellation error in subtracting asin(x)
 * from pi/2.  Hence if x < -0.5,
 *
 *    acos(x) =  pi - 2.0 * asin( sqrt((1+x)/2) );
 *
 * or if x > +0.5,
 *
 *    acos(x) =  2.0 * asin(  sqrt((1-x)/2) ).
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -1, 1       30000       1.4e-19     3.5e-20
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * acosl domain        |x| > 1           NANL
 */



/*                                                      atanhl.c
 *
 *      Inverse hyperbolic tangent, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, atanhl();
 *
 * y = atanhl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns inverse hyperbolic tangent of argument in the range
 * MINLOGL to MAXLOGL.
 *
 * If |x| < 0.5, the rational form x + x**3 P(x)/Q(x) is
 * employed.  Otherwise,
 *        atanh(x) = 0.5 * log( (1+x)/(1-x) ).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -1,1        30000       1.1e-19     3.3e-20
 *
 */



/*                                                      atanl.c
 *
 *      Inverse circular tangent, long double precision
 *      (arctangent)
 *
 *
 *
```

```
 * SYNOPSIS:
 *
 * long double x, y, atanl();
 *
 * y = atanl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns radian angle between -pi/2 and +pi/2 whose tangent
 * is x.
 *
 * Range reduction is from four intervals into the interval
 * from zero to  tan( pi/8 ).  The approximant uses a rational
 * function of degree 3/4 of the form x + x**3 P(x)/Q(x).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10, 10    150000       1.3e-19     3.0e-20
 *
 */



/*                                                      atan2l()
 *
 *      Quadrant correct inverse circular tangent,
 *      long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, z, atan2l();
 *
 * z = atan2l( y, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns radian angle whose tangent is y/x.
 * Define compile time symbol ANSIC = 1 for ANSI standard,
 * range -PI < z <= +PI, args (y,x); else ANSIC = 0 for range
 * 0 to 2PI, args (x,y).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10, 10     60000       1.7e-19     3.2e-20
 * See atan.c.
 *
 */



/*                                                      bdtrl.c
 *
 *      Binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k, n;
 * long double p, y, bdtrl();
 *
 * y = bdtrl( 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:
```

```
 *
 * Tested at random points (k,n,p) with a and b between 0
 * and 10000 and p between 0 and 1.
 *    Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,10000      3000       1.6e-14     2.2e-15
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * bdtrl domain        k < 0            0.0
 *                     n < k
 *                     x < 0, x > 1
 *
 */



/*                                                      bdtrcl()
 *
 *      Complemented binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k, n;
 * long double p, y, bdtrcl();
 *
 * y = bdtrcl( 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:
 *
 * See incbet.c.
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * bdtrcl domain     x<0, x>1, n<k       0.0
 */



/*                                                      bdtril()
 *
 *      Inverse binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k, n;
 * long double p, y, bdtril();
 *
 * p = bdtril( 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:
 *
 * See incbi.c.
 * Tested at random k, n between 1 and 10000.  The "domain" refers to p:
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,1        3500       2.0e-15     8.2e-17
 *
 * ERROR MESSAGES:
```

```
 *
 *   message         condition      value returned
 * bdtril domain     k < 0, n <= k         0.0
 *                  x < 0, x > 1
 */



/*                                                      btdtrl.c
 *
 *      Beta distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * long double a, b, x, y, btdtrl();
 *
 * y = btdtrl( a, b, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the area from zero to x under the beta density
 * function:
 *
 *
 *                          x
 *            -             -
 *           | (a+b)       | |  a-1      b-1
 * P(x)  =  ----------     |   t    (1-t)    dt
 *           -     -     | |
 *          | (a) | (b)   -
 *                         0
 *
 *
 * The mean value of this distribution is a/(a+b).  The variance
 * is ab/[(a+b)^2 (a+b+1)].
 *
 * This function is identical to the incomplete beta integral
 * function, incbetl(a, b, x).
 *
 * The complemented function is
 *
 * 1 - P(1-x)  =  incbetl( b, a, x );
 *
 *
 * ACCURACY:
 *
 * See incbetl.c.
 *
 */



/*                                                      cbrtl.c
 *
 *      Cube root, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, cbrtl();
 *
 * y = cbrtl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the cube root of the argument, which may be negative.
 *
 * Range reduction involves determining the power of 2 of
 * the argument.  A polynomial of degree 2 applied to the
 * mantissa, and multiplication by the cube root of 1, 2, or 4
 * approximates the root to within about 0.1%.  Then Newton's
 * iteration is used three times to converge to an accurate
 * result.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     .125,8        80000      7.0e-20     2.2e-20
 *    IEEE    exp(+-707)    100000      7.0e-20     2.4e-20
 *
 */



/*                                                      chdtrl.c
 *
 *      Chi-square distribution
 *
 *
 *
```

```
 * SYNOPSIS:
 *
 * long double df, x, y, chdtrl();
 *
 * y = chdtrl( 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:
 *
 * See igam().
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * chdtr domain   x < 0 or v < 1        0.0
 */


/*                                                      chdtrcl()
 *
 *      Complemented Chi-square distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * long double v, x, y, chdtrcl();
 *
 * y = chdtrcl( 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:
 *
 * See igamc().
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * chdtrc domain  x < 0 or v < 1        0.0
 */
```

```
/*							chdtril()
 *
 *	Inverse of complemented Chi-square distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * long double df, x, y, chdtril();
 *
 * x = chdtril( 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:
 *
 * See igami.c.
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * chdtri domain   y < 0 or y > 1        0.0
 *                     v < 1
 *
 */



/*							clogl.c
 *
 *	Complex natural logarithm
 *
 *
 *
 * SYNOPSIS:
 *
 * void clogl();
 * cmplxl z, w;
 *
 * clogl( &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
 *    DEC       -10,+10      7000       8.5e-17     1.9e-17
 *    IEEE      -10,+10     30000       5.0e-15     1.1e-16
 *
 * Larger relative error can be observed for z near 1 +i0.
 * In IEEE arithmetic the peak absolute error is 5.2e-16, rms
 * absolute error 1.0e-16.
 */



/*							cexpl()
 *
 *	Complex exponential function
 *
 *
 *
 * SYNOPSIS:
 *
 * void cexpl();
 * cmplxl z, w;
 *
 * cexpl( &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
 *    DEC       -10,+10      8700       3.7e-17     1.1e-17
 *    IEEE      -10,+10     30000       3.0e-16     8.7e-17
 *
 */



/*                                                      csinl()
 *
 *      Complex circular sine
 *
 *
 *
 * SYNOPSIS:
 *
 * void csinl();
 * cmplxl z, w;
 *
 * csinl( &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
 *    DEC       -10,+10      8400       5.3e-17     1.3e-17
 *    IEEE      -10,+10     30000       3.8e-16     1.0e-16
 * Also tested by csin(casin(z)) = z.
 *
 */



/*                                                      ccosl()
 *
 *      Complex circular cosine
 *
 *
 *
 * SYNOPSIS:
 *
 * void ccosl();
 * cmplxl z, w;
 *
 * ccosl( &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
 *    DEC       -10,+10      8400       4.5e-17     1.3e-17
 *    IEEE      -10,+10     30000       3.8e-16     1.0e-16
 */


/*                                                      ctanl()
 *
```

```
 *      Complex circular tangent
 *
 *
 *
 * SYNOPSIS:
 *
 * void ctanl();
 * cmplxl z, w;
 *
 * ctanl( &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
 *    DEC       -10,+10      5200       7.1e-17     1.6e-17
 *    IEEE      -10,+10     30000       7.2e-16     1.2e-16
 * Also tested by ctan * ccot = 1 and catan(ctan(z))  =  z.
 */



/*                                                      ccotl()
 *
 *      Complex circular cotangent
 *
 *
 *
 * SYNOPSIS:
 *
 * void ccotl();
 * cmplxl z, w;
 *
 * ccotl( &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
 *    DEC       -10,+10      3000       6.5e-17     1.6e-17
 *    IEEE      -10,+10     30000       9.2e-16     1.2e-16
 * Also tested by ctan * ccot = 1 + i0.
 */



/*                                                      casinl()
 *
 *      Complex circular arc sine
 *
 *
 *
 * SYNOPSIS:
 *
 * void casinl();
 * cmplxl z, w;
 *
 * casinl( &z, &w );
 *
 *
 *
 * DESCRIPTION:
 *
 * Inverse complex sine:
```

```
 *
 *                              2
 * w = -i clog( iz + csqrt( 1 - z ) ).
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    DEC       -10,+10     10100       2.1e-15     3.4e-16
 *    IEEE      -10,+10     30000       2.2e-14     2.7e-15
 * Larger relative error can be observed for z near zero.
 * Also tested by csin(casin(z)) = z.
 */



/*                                                      cacosl()
 *
 *      Complex circular arc cosine
 *
 *
 *
 * SYNOPSIS:
 *
 * void cacosl();
 * cmplxl z, w;
 *
 * cacosl( &z, &w );
 *
 *
 *
 * DESCRIPTION:
 *
 *
 * w = arccos z  =  PI/2 - arcsin z.
 *
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    DEC       -10,+10      5200      1.6e-15      2.8e-16
 *    IEEE      -10,+10     30000      1.8e-14      2.2e-15
 */



/*                                                      catanl()
 *
 *      Complex circular arc tangent
 *
 *
 *
 * SYNOPSIS:
 *
 * void catanl();
 * cmplxl z, w;
 *
 * catanl( &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
 *    DEC       -10,+10      5900       1.3e-16     7.8e-18
 *    IEEE      -10,+10     30000       2.3e-15     8.5e-17
 * The check catan( ctan(z) )  =  z, with |x| and |y| < PI/2,
 * had peak relative error 1.5e-16, rms relative error
 * 2.9e-17.  See also clog().
 */
```

```
/*                                                      cmplxl.c
 *
 *      Complex number arithmetic
 *
 *
 *
 * SYNOPSIS:
 *
 * typedef struct {
 *      long double r;     real part
 *      long double i;     imaginary part
 *     }cmplxl;
 *
 * cmplxl *a, *b, *c;
 *
 * caddl( a, b, c );     c = b + a
 * csubl( a, b, c );     c = b - a
 * cmull( a, b, c );     c = b * a
 * cdivl( a, b, c );     c = b / a
 * cnegl( c );           c = -c
 * cmovl( 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
 *    DEC        cadd       10000       1.4e-17     3.4e-18
 *    IEEE       cadd      100000       1.1e-16     2.7e-17
 *    DEC        csub       10000       1.4e-17     4.5e-18
 *    IEEE       csub      100000       1.1e-16     3.4e-17
 *    DEC        cmul        3000       2.3e-17     8.7e-18
 *    IEEE       cmul      100000       2.1e-16     6.9e-17
 *    DEC        cdiv       18000       4.9e-17     1.3e-17
 *    IEEE       cdiv      100000       3.7e-16     1.1e-16
 */



/*                                                      coshl.c
 *
 *      Hyperbolic cosine, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, coshl();
 *
 * y = coshl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns hyperbolic cosine of argument in the range MINLOGL to
 * MAXLOGL.
 *
 * cosh(x)  =  ( exp(x) + exp(-x) )/2.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     +-10000      30000       1.1e-19     2.8e-20
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition              value returned
 * cosh overflow    |x| > MAXLOGL+LOGE2L      INFINITYL
 *
 *
 */
```

```
/*                                                      elliel.c
 *
 *      Incomplete elliptic integral of the second kind
 *
 *
 *
 * SYNOPSIS:
 *
 * long double phi, m, y, elliel();
 *
 * y = elliel( 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 [-10, 10] and m in
 * [0, 1].
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     -10,10       50000       2.7e-18     2.3e-19
 *
 *
 */



/*                                                      ellikl.c
 *
 *      Incomplete elliptic integral of the first kind
 *
 *
 *
 * SYNOPSIS:
 *
 * long double phi, m, y, ellikl();
 *
 * y = ellikl( 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 m in [0, 1] and phi as indicated.
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     -10,10        30000      3.6e-18     4.1e-19
 *
 *
 */



/*                                                      ellpel.c
 *
```

```
 *      Complete elliptic integral of the second kind
 *
 *
 *
 * SYNOPSIS:
 *
 * long double m1, y, ellpel();
 *
 * y = ellpel( 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       10000       1.1e-19     3.5e-20
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * ellpel domain     x<0, x>1            0.0
 *
 */



/*                                                      ellpjl.c
 *
 *      Jacobian Elliptic Functions
 *
 *
 *
 * SYNOPSIS:
 *
 * long double u, m, sn, cn, dn, phi;
 * int ellpjl();
 *
 * ellpjl( 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-12 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-18     2.3e-19
 *    IEEE      cn          20000       1.6e-18     2.2e-19
 *    IEEE      dn         100000       2.9e-18     9.1e-20
 *    IEEE      phi         10000       4.0e-19*    6.6e-20*
 *
 * Accuracy deteriorates when u is large.
 * Larger errors occur for m near 1.
```

```
 *
 */



/*                                                      ellpkl.c
 *
 *      Complete elliptic integral of the first kind
 *
 *
 *
 * SYNOPSIS:
 *
 * long double m1, y, ellpkl();
 *
 * y = ellpkl( 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        10000       1.1e-19     3.3e-20
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * ellpkl domain      x<0, x>1           0.0
 *
 */



/*                                                      exp10l.c
 *
 *      Base 10 exponential function, long double precision
 *      (Common antilogarithm)
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, exp10l()
 *
 * y = exp10l( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns 10 raised to the x power.
 *
 * Range reduction is accomplished by expressing the argument
 * as 10**x = 2**n 10**f, with |f| < 0.5 log10(2).
 * The Pade' form
 *
 *     1 + 2x P(x**2)/( Q(x**2) - P(x**2) )
 *
 * is used to approximate 10**f.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      +-4900      30000       1.0e-19     2.7e-20
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * exp10l underflow    x < -MAXL10        0.0
 * exp10l overflow     x > MAXL10       MAXNUM
```

```
 *
 * IEEE arithmetic: MAXL10 = 4932.0754489586679023819
 *
 */



/*							exp2l.c
 *
 *	Base 2 exponential function, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, exp2l();
 *
 * y = exp2l( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns 2 raised to the x power.
 *
 * Range reduction is accomplished by separating the argument
 * into an integer k and fraction f such that
 *     x    k  f
 *    2  = 2  2.
 *
 * A Pade' form
 *
 *   1 + 2x P(x**2) / (Q(x**2) - x P(x**2) )
 *
 * approximates 2**x in the basic range [-0.5, 0.5].
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      +-16300     300000      9.1e-20     2.6e-20
 *
 *
 * See exp.c for comments on error amplification.
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * exp2l underflow   x < -16382        0.0
 * exp2l overflow    x >= 16384       MAXNUM
 *
 */



/*							expl.c
 *
 *	Exponential function, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, expl();
 *
 * y = expl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns e (2.71828...) raised to the x power.
 *
 * Range reduction is accomplished by separating the argument
 * into an integer k and fraction f such that
 *
 *     x    k  f
 *    e  = 2  e.
 *
 * A Pade' form of degree 2/3 is used to approximate exp(f) - 1
 * in the basic range [-0.5 ln 2, 0.5 ln 2].
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      +-10000     50000       1.12e-19    2.81e-20
 *
 *
 * Error amplification in the exponential function can be
 * a serious matter.  The error propagation involves
 * exp( X(1+delta) ) = exp(X) ( 1 + X*delta + ... ),
 * which shows that a 1 lsb error in representing X produces
 * a relative error of X times 1 lsb in the function.
 * While the routine gives an accurate result for arguments
 * that are exactly represented by a long double precision
 * computer number, the result contains amplified roundoff
```

```
 * error for large arguments not exactly represented.
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * exp underflow    x < MINLOG         0.0
 * exp overflow     x > MAXLOG         MAXNUM
 *
 */



/*                                                      expm1l.c
 *
 *      Exponential function, minus 1
 *      Long double precision
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, expm1l();
 *
 * y = expm1l( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns e (2.71828...) raised to the x power, minus 1.
 *
 * Range reduction is accomplished by separating the argument
 * into an integer k and fraction f such that
 *
 *     x    k  f
 *    e  = 2  e.
 *
 * An expansion x + .5 x^2 + x^3 R(x) approximates exp(f) - 1
 * in the basic range [-0.5 ln 2, 0.5 ln 2].
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE    -45,+MAXLOG   200,000     1.2e-19     2.5e-20
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * expm1l overflow   x > MAXLOG         MAXNUM
 *
 */



/*                                                      expx2l.c
 *
 *      Exponential of squared argument
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, expx2l();
 * int sign;
 *
 * y = expx2l( x, sign );
 *
 *
 *
 * DESCRIPTION:
 *
 * Computes y = exp(x*x) while suppressing error amplification
 * that would ordinarily arise from the inexactness of the
 * exponential argument x*x.
 *
 * If sign < 0, the result is inverted; i.e., y = exp(-x*x) .
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic      domain        # trials      peak         rms
 *   IEEE     -106.566, 106.566    10^5       1.6e-19     4.4e-20
 *
 */



/*                                                      fdtrl.c
 *
 *      F distribution, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * int df1, df2;
 * long double x, y, fdtrl();
```

```
 *
 * y = fdtrl( 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) = incbetl( df1/2, df2/2, (df1*x/(df2 + df1*x) ).
 *
 *
 * The arguments a and b are greater than zero, and x
 * x is nonnegative.
 *
 * ACCURACY:
 *
 * Tested at random points (a,b,x) in the indicated intervals.
 *                x     a,b                     Relative error:
 * arithmetic  domain  domain     # trials      peak         rms
 *    IEEE      0,1    1,100       10000       9.3e-18     2.9e-19
 *    IEEE      0,1    1,10000     10000       1.9e-14     2.9e-15
 *    IEEE      1,5    1,10000     10000       5.8e-15     1.4e-16
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * fdtrl domain     a<0, b<0, x<0         0.0
 *
 */



/*                                                      fdtrcl()
 *
 *      Complemented F distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int df1, df2;
 * long double x, y, fdtrcl();
 *
 * y = fdtrcl( 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:
 *
 * See incbet.c.
 * Tested at random points (a,b,x).
 *
 *                x     a,b                     Relative error:
 * arithmetic  domain  domain     # trials      peak         rms
 *    IEEE      0,1    0,100       10000       4.2e-18     3.3e-19
 *    IEEE      0,1    1,10000     10000       7.2e-15     2.6e-16
 *    IEEE      1,5    1,10000     10000       1.7e-14     3.0e-15
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * fdtrcl domain    a<0, b<0, x<0         0.0
 *
 */
```

```
/*							fdtril()
 *
 *	Inverse of complemented F distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int df1, df2;
 * long double x, p, fdtril();
 *
 * x = fdtril( df1, df2, p );
 *
 * 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 p.
 *
 * This is accomplished using the inverse beta integral
 * function and the relations
 *
 *      z = incbi( df2/2, df1/2, p )
 *      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, p )
 *      x = df2 z / (df1 (1-z)).
 *
 * ACCURACY:
 *
 * See incbi.c.
 * Tested at random points (a,b,p).
 *
 *              a,b                     Relative error:
 * arithmetic  domain     # trials      peak         rms
 *  For p between .001 and 1:
 *    IEEE     1,100       40000       4.6e-18     2.7e-19
 *    IEEE     1,10000     30000       1.7e-14     1.4e-16
 *  For p between 10^-6 and .001:
 *    IEEE     1,100       20000       1.9e-15     3.9e-17
 *    IEEE     1,10000     30000       2.7e-15     4.0e-17
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * fdtril domain   p <= 0 or p > 1       0.0
 *                     v < 1
 */



/*							ceill()
 *							floorl()
 *							frexpl()
 *							ldexpl()
 *							fabsl()
 *							signbitl()
 *							isnanl()
 *							isfinitel()
 *
 *	Floating point numeric utilities
 *
 *
 *
 * SYNOPSIS:
 *
 * long double ceill(), floorl(), frexpl(), ldexpl(), fabsl();
 * int signbitl(), isnanl(), isfinitel();
 * long double x, y;
 * int expnt, n;
 *
 * y = floorl(x);
 * y = ceill(x);
 * y = frexpl( x, &expnt );
 * y = ldexpl( x, n );
 * y = fabsl( x );
 * n = signbitl(x);
 * n = isnanl(x);
 * n = isfinitel(x);
 *
 *
 *
 * DESCRIPTION:
 *
 * The following routines return a long double precision floating point
 * result:
 *
 * floorl() returns the largest integer less than or equal to x.
 * It truncates toward minus infinity.
 *
 * ceill() returns the smallest integer greater than or equal
 * to x.  It truncates toward plus infinity.
 *
 * frexpl() extracts the exponent from x.  It returns an integer
 * power of two to expnt and the significand between 0.5 and 1
 * to y.  Thus  x = y * 2**expn.
 *
```

```
 * ldexpl() multiplies x by 2**n.
 *
 * fabsl() returns the absolute value of its argument.
 *
 * These functions are part of the standard C run time library
 * for some but not all C compilers.  The ones supplied are
 * written in C for IEEE arithmetic.  They should
 * be used only if your compiler library does not already have
 * them.
 *
 * The IEEE versions assume that denormal numbers are implemented
 * in the arithmetic.  Some modifications will be required if
 * the arithmetic has abrupt rather than gradual underflow.
 */



/*                                                      gammal.c
 *
 *      Gamma function
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, gammal();
 * extern int sgngam;
 *
 * y = gammal( 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 sgngam.
 * This variable is also filled in by the logarithmic gamma
 * function lgam().
 *
 * Arguments |x| <= 13 are reduced by recurrence and the function
 * approximated by a rational function of degree 7/8 in the
 * interval (2,3).  Large arguments are handled by Stirling's
 * formula. Large negative arguments are made positive using
 * a reflection formula.
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     -40,+40      10000       3.6e-19     7.9e-20
 *    IEEE    -1755,+1755   10000       4.8e-18     6.5e-19
 *
 * Accuracy for large arguments is dominated by error in powl().
 *
 */



/*                                                      lgaml()
 *
 *      Natural logarithm of gamma function
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, lgaml();
 * extern int sgngam;
 *
 * y = lgaml( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the base e (2.718...) logarithm of the absolute
 * value of the gamma function of the argument.
 * The sign (+1 or -1) of the gamma function is returned in a
 * global (extern) variable named sgngam.
 *
 * For arguments greater than 33, the logarithm of the gamma
 * function is approximated by the logarithmic version of
 * Stirling's formula using a polynomial approximation of
 * degree 4. Arguments between -33 and +33 are reduced by
 * recurrence to the interval [2,3] of a rational approximation.
 * The cosecant reflection formula is employed for arguments
 * less than -33.
 *
 * Arguments greater than MAXLGML (10^4928) return MAXNUML.
 *
 *
 *
 * ACCURACY:
 *
 *
 * arithmetic      domain        # trials     peak         rms
 *    IEEE         -40, 40        100000     2.2e-19     4.6e-20
 *    IEEE    10^-2000,10^+2000    20000     1.6e-19     3.3e-20
```

```
 * The error criterion was relative when the function magnitude
 * was greater than one but absolute when it was less than one.
 *
 */



/*                                                      gdtrl.c
 *
 *      Gamma distribution function
 *
 *
 *
 * SYNOPSIS:
 *
 * long double a, b, x, y, gdtrl();
 *
 * y = gdtrl( 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:
 *
 * See igam().
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * gdtrl domain        x < 0            0.0
 *
 */



/*                                                      gdtrcl.c
 *
 *      Complemented gamma distribution function
 *
 *
 *
 * SYNOPSIS:
 *
 * long double a, b, x, y, gdtrcl();
 *
 * y = gdtrcl( 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:
 *
 * See igamc().
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * gdtrcl domain        x < 0            0.0
 *
 */
```

```
/*
C
C     ..................................................................
C
C        SUBROUTINE GELS
C
C        PURPOSE
C           TO SOLVE A SYSTEM OF SIMULTANEOUS LINEAR EQUATIONS WITH
C           SYMMETRIC COEFFICIENT MATRIX UPPER TRIANGULAR PART OF WHICH
C           IS ASSUMED TO BE STORED COLUMNWISE.
C
C        USAGE
C           CALL GELS(R,A,M,N,EPS,IER,AUX)
C
C        DESCRIPTION OF PARAMETERS
C           R      - M BY N RIGHT HAND SIDE MATRIX.  (DESTROYED)
C                    ON RETURN R CONTAINS THE SOLUTION OF THE EQUATIONS.
C           A      - UPPER TRIANGULAR PART OF THE SYMMETRIC
C                    M BY M COEFFICIENT MATRIX.  (DESTROYED)
C           M      - THE NUMBER OF EQUATIONS IN THE SYSTEM.
C           N      - THE NUMBER OF RIGHT HAND SIDE VECTORS.
C           EPS    - AN INPUT CONSTANT WHICH IS USED AS RELATIVE
C                    TOLERANCE FOR TEST ON LOSS OF SIGNIFICANCE.
C           IER    - RESULTING ERROR PARAMETER CODED AS FOLLOWS
C                    IER=0  - NO ERROR,
C                    IER=-1 - NO RESULT BECAUSE OF M LESS THAN 1 OR
C                             PIVOT ELEMENT AT ANY ELIMINATION STEP
C                             EQUAL TO 0,
C                    IER=K  - WARNING DUE TO POSSIBLE LOSS OF SIGNIFI-
C                             CANCE INDICATED AT ELIMINATION STEP K+1,
C                             WHERE PIVOT ELEMENT WAS LESS THAN OR
C                             EQUAL TO THE INTERNAL TOLERANCE EPS TIMES
C                             ABSOLUTELY GREATEST MAIN DIAGONAL
C                             ELEMENT OF MATRIX A.
C           AUX    - AN AUXILIARY STORAGE ARRAY WITH DIMENSION M-1.
C
C        REMARKS
C           UPPER TRIANGULAR PART OF MATRIX A IS ASSUMED TO BE STORED
C           COLUMNWISE IN M*(M+1)/2 SUCCESSIVE STORAGE LOCATIONS, RIGHT
C           HAND SIDE MATRIX R COLUMNWISE IN N*M SUCCESSIVE STORAGE
C           LOCATIONS. ON RETURN SOLUTION MATRIX R IS STORED COLUMNWISE
C           TOO.
C           THE PROCEDURE GIVES RESULTS IF THE NUMBER OF EQUATIONS M IS
C           GREATER THAN 0 AND PIVOT ELEMENTS AT ALL ELIMINATION STEPS
C           ARE DIFFERENT FROM 0. HOWEVER WARNING IER=K - IF GIVEN -
C           INDICATES POSSIBLE LOSS OF SIGNIFICANCE. IN CASE OF A WELL
C           SCALED MATRIX A AND APPROPRIATE TOLERANCE EPS, IER=K MAY BE
C           INTERPRETED THAT MATRIX A HAS THE RANK K. NO WARNING IS
C           GIVEN IN CASE M=1.
C           ERROR PARAMETER IER=-1 DOES NOT NECESSARILY MEAN THAT
C           MATRIX A IS SINGULAR, AS ONLY MAIN DIAGONAL ELEMENTS
C           ARE USED AS PIVOT ELEMENTS. POSSIBLY SUBROUTINE GELG (WHICH
C           WORKS WITH TOTAL PIVOTING) WOULD BE ABLE TO FIND A SOLUTION.
C
C        SUBROUTINES AND FUNCTION SUBPROGRAMS REQUIRED
C           NONE
C
C        METHOD
C           SOLUTION IS DONE BY MEANS OF GAUSS-ELIMINATION WITH
C           PIVOTING IN MAIN DIAGONAL, IN ORDER TO PRESERVE
C           SYMMETRY IN REMAINING COEFFICIENT MATRICES.
C
C     ..................................................................
C
*/


/*                                                      hypergl.c
 *
 *      Confluent hypergeometric function
 *
 *
 *
 * SYNOPSIS:
 *
 * long double a, b, x, y, hypergl();
 *
 * y = hypergl( 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      0,30        100000      3.3e-18     5.0e-19
 *
 * 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-12.
 *
 */
```

```
/*                                                      ieee.c
 *
 *    Extended precision IEEE binary floating point arithmetic routines
 *
 * Numbers are stored in C language as arrays of 16-bit unsigned
 * short integers.  The arguments of the routines are pointers to
 * the arrays.
 *
 *
 * External e type data structure, simulates Intel 8087 chip
 * temporary real format but possibly with a larger significand:
 *
 *      NE-1 significand words  (least significant word first,
 *                               most significant bit is normally set)
 *      exponent                (value = EXONE for 1.0,
 *                              top bit is the sign)
 *
 *
 * Internal data structure of a number (a "word" is 16 bits):
 *
 * ei[0]        sign word       (0 for positive, 0xffff for negative)
 * ei[1]        biased exponent (value = EXONE for the number 1.0)
 * ei[2]        high guard word (always zero after normalization)
 * ei[3]
 * to ei[NI-2]  significand     (NI-4 significand words,
 *                               most significant word first,
 *                               most significant bit is set)
 * ei[NI-1]     low guard word  (0x8000 bit is rounding place)
 *
 *
 *
 *              Routines for external format numbers
 *
 *      asctoe( string, e )     ASCII string to extended double e type
 *      asctoe64( string, &d )  ASCII string to long double
 *      asctoe53( string, &d )  ASCII string to double
 *      asctoe24( string, &f )  ASCII string to single
 *      asctoeg( string, e, prec ) ASCII string to specified precision
 *      e24toe( &f, e )         IEEE single precision to e type
 *      e53toe( &d, e )         IEEE double precision to e type
 *      e64toe( &d, e )         IEEE long double precision to e type
 *      eabs(e)                 absolute value
 *      eadd( a, b, c )         c = b + a
 *      eclear(e)               e = 0
 *      ecmp (a, b)             Returns 1 if a > b, 0 if a == b,
 *                              -1 if a < b, -2 if either a or b is a NaN.
 *      ediv( a, b, c )         c = b / a
 *      efloor( a, b )          truncate to integer, toward -infinity
 *      efrexp( a, exp, s )     extract exponent and significand
 *      eifrac( e, &l, frac )   e to long integer and e type fraction
 *      euifrac( e, &l, frac )  e to unsigned long integer and e type fraction
 *      einfin( e )             set e to infinity, leaving its sign alone
 *      eldexp( a, n, b )       multiply by 2**n
 *      emov( a, b )            b = a
 *      emul( a, b, c )         c = b * a
 *      eneg(e)                 e = -e
 *      eround( a, b )          b = nearest integer value to a
 *      esub( a, b, c )         c = b - a
 *      e24toasc( &f, str, n )  single to ASCII string, n digits after decimal
 *      e53toasc( &d, str, n )  double to ASCII string, n digits after decimal
 *      e64toasc( &d, str, n )  long double to ASCII string
 *      etoasc( e, str, n )     e to ASCII string, n digits after decimal
 *      etoe24( e, &f )         convert e type to IEEE single precision
 *      etoe53( e, &d )         convert e type to IEEE double precision
 *      etoe64( e, &d )         convert e type to IEEE long double precision
 *      ltoe( &l, e )           long (32 bit) integer to e type
 *      ultoe( &l, e )          unsigned long (32 bit) integer to e type
 *      eisneg( e )             1 if sign bit of e != 0, else 0
 *      eisinf( e )             1 if e has maximum exponent (non-IEEE)
 *                              or is infinite (IEEE)
 *      eisnan( e )             1 if e is a NaN
 *      esqrt( a, b )           b = square root of a
 *
 *
 *              Routines for internal format numbers
 *
 *      eaddm( ai, bi )         add significands, bi = bi + ai
```

```
 *      ecleaz(ei)              ei = 0
 *      ecleazs(ei)             set ei = 0 but leave its sign alone
 *      ecmpm( ai, bi )         compare significands, return 1, 0, or -1
 *      edivm( ai, bi )         divide  significands, bi = bi / ai
 *      emdnorm(ai,l,s,exp)     normalize and round off
 *      emovi( a, ai )          convert external a to internal ai
 *      emovo( ai, a )          convert internal ai to external a
 *      emovz( ai, bi )         bi = ai, low guard word of bi = 0
 *      emulm( ai, bi )         multiply significands, bi = bi * ai
 *      enormlz(ei)             left-justify the significand
 *      eshdn1( ai )            shift significand and guards down 1 bit
 *      eshdn8( ai )            shift down 8 bits
 *      eshdn6( ai )            shift down 16 bits
 *      eshift( ai, n )         shift ai n bits up (or down if n < 0)
 *      eshup1( ai )            shift significand and guards up 1 bit
 *      eshup8( ai )            shift up 8 bits
 *      eshup6( ai )            shift up 16 bits
 *      esubm( ai, bi )         subtract significands, bi = bi - ai
 *
 *
 * The result is always normalized and rounded to NI-4 word precision
 * after each arithmetic operation.
 *
 * Exception flags are NOT fully supported.
 *
 * Define INFINITY in mconf.h for support of infinity; otherwise a
 * saturation arithmetic is implemented.
 *
 * Define NANS for support of Not-a-Number items; otherwise the
 * arithmetic will never produce a NaN output, and might be confused
 * by a NaN input.
 * If NaN's are supported, the output of ecmp(a,b) is -2 if
 * either a or b is a NaN. This means asking if(ecmp(a,b) < 0)
 * may not be legitimate. Use if(ecmp(a,b) == -1) for less-than
 * if in doubt.
 * Signaling NaN's are NOT supported; they are treated the same
 * as quiet NaN's.
 *
 * Denormals are always supported here where appropriate (e.g., not
 * for conversion to DEC numbers).
 */

/*
 * Revision history:
 *
 *  5 Jan 84    PDP-11 assembly language version
 *  2 Mar 86    fixed bug in asctoq()
 *  6 Dec 86    C language version
 * 30 Aug 88    100 digit version, improved rounding
 * 15 May 92    80-bit long double support
 *
 * Author:  S. L. Moshier.
 */



/*                                                      igamil()
 *
 *      Inverse of complemented imcomplete gamma integral
 *
 *
 *
 * SYNOPSIS:
 *
 * long double a, x, y, igamil();
 *
 * x = igamil( 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 0.5 to 30 and x from 0 to 0.5.
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
```

```
 *    DEC       0,0.5         3400       8.8e-16     1.3e-16
 *    IEEE      0,0.5        10000       1.1e-14     1.0e-15
 *
 */



/*                                                      igaml.c
 *
 *      Incomplete gamma integral
 *
 *
 *
 * SYNOPSIS:
 *
 * long double a, x, y, igaml();
 *
 * y = igaml( 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
 *    DEC       0,30         4000       4.4e-15     6.3e-16
 *    IEEE      0,30        10000       3.6e-14     5.1e-15
 *
 */



/*                                                      igamcl()
 *
 *      Complemented incomplete gamma integral
 *
 *
 *
 * SYNOPSIS:
 *
 * long double a, x, y, igamcl();
 *
 * y = igamcl( 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
 *    DEC       0,30         2000       2.7e-15     4.0e-16
 *    IEEE      0,30        60000       1.4e-12     6.3e-15
 *
 */
```

```
/*                                                      incbetl.c
 *
 *      Incomplete beta integral
 *
 *
 * SYNOPSIS:
 *
 * long double a, b, x, y, incbetl();
 *
 * y = incbetl( 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
 * or, when b*x is small, by a power series.
 *
 * ACCURACY:
 *
 * Tested at random points (a,b,x) with x between 0 and 1.
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,5       20000        4.5e-18     2.4e-19
 *    IEEE       0,100    100000        3.9e-17     1.0e-17
 * Half-integer a, b:
 *    IEEE      .5,10000  100000        3.9e-14     4.4e-15
 * Outputs smaller than the IEEE gradual underflow threshold
 * were excluded from these statistics.
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * incbetl domain     x<0, x>1          0.0
 */



/*                                                      incbil()
 *
 *      Inverse of imcomplete beta integral
 *
 *
 *
 * SYNOPSIS:
 *
 * long double a, b, x, y, incbil();
 *
 * x = incbil( 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    .5,10000    10000    1.1e-14   1.4e-16
 */



/*                                                      isnanl()
 *                                                      isfinitel()
 *                                                      signbitl()
 *
 *      Floating point IEEE special number tests
 *
 *
 *
 * SYNOPSIS:
 *
 * int signbitl(), isnanl(), isfinitel();
 * long double x, y;
```

```
 *
 * n = signbitl(x);
 * n = isnanl(x);
 * n = isfinitel(x);
 *
 *
 *
 * DESCRIPTION:
 *
 * These functions are part of the standard C run time library
 * for some but not all C compilers.  The ones supplied are
 * written in C for IEEE arithmetic.  They should
 * be used only if your compiler library does not already have
 * them.
 *
 */
```

```
/*                                                      j0l.c
 *
 *      Bessel function of order zero
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, j0l();
 *
 * y = j0l( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of first kind, order zero of the argument.
 *
 * The domain is divided into the intervals [0, 9] and
 * (9, infinity). In the first interval the rational approximation
 * is (x^2 - r^2) (x^2 - s^2) (x^2 - t^2) P7(x^2) / Q8(x^2),
 * where r, s, t are the first three zeros of the function.
 * In the second interval the expansion is in terms of the
 * modulus M0(x) = sqrt(J0(x)^2 + Y0(x)^2) and phase  P0(x)
 * = atan(Y0(x)/J0(x)).  M0 is approximated by sqrt(1/x)P7(1/x)/Q7(1/x).
 * The approximation to J0 is M0 * cos(x -  pi/4 + 1/x P5(1/x^2)/Q6(1/x^2)).
 *
 *
 * ACCURACY:
 *
 *                      Absolute error:
 * arithmetic   domain      # trials      peak         rms
 *    IEEE      0, 30       100000      2.8e-19      7.4e-20
 *
 *
 */
```

```
/*                                                      y0l.c
 *
 *      Bessel function of the second kind, order zero
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, y0l();
 *
 * y = y0l( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of the second kind, of order
 * zero, of the argument.
 *
 * The domain is divided into the intervals [0, 5>, [5,9> and
 * [9, infinity). In the first interval a rational approximation
 * R(x) is employed to compute y0(x)  = R(x) + 2/pi * log(x) * j0(x).
 *
 * In the second interval, the approximation is
 *     (x - p)(x - q)(x - r)(x - s)P7(x)/Q7(x)
 * where p, q, r, s are zeros of y0(x).
 *
 * The third interval uses the same approximations to modulus
 * and phase as j0(x), whence y0(x) = modulus * sin(phase).
 *
 * ACCURACY:
 *
 *  Absolute error, when y0(x) < 1; else relative error:
 *
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0, 30       100000      3.4e-19     7.6e-20
 *
 */
```

```
/*                                                      j1l.c
 *
 *      Bessel function of order one
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, j1l();
 *
 * y = j1l( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of order one of the argument.
 *
 * The domain is divided into the intervals [0, 9] and
 * (9, infinity). In the first interval the rational approximation
 * is (x^2 - r^2) (x^2 - s^2) (x^2 - t^2) x P8(x^2) / Q8(x^2),
 * where r, s, t are the first three zeros of the function.
 * In the second interval the expansion is in terms of the
 * modulus M1(x) = sqrt(J1(x)^2 + Y1(x)^2) and phase  P1(x)
 * = atan(Y1(x)/J1(x)).  M1 is approximated by sqrt(1/x)P7(1/x)/Q8(1/x).
 * The approximation to j1 is M1 * cos(x -  3 pi/4 + 1/x P5(1/x^2)/Q6(1/x^2)).
 *
 *
 * ACCURACY:
 *
 *                      Absolute error:
 * arithmetic   domain      # trials      peak         rms
 *    IEEE      0, 30        40000      1.8e-19      5.0e-20
 *
 *
 */



/*                                                      y1l.c
 *
 *      Bessel function of the second kind, order zero
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, y1l();
 *
 * y = y1l( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of the second kind, of order
 * zero, of the argument.
 *
 * The domain is divided into the intervals [0, 4.5>, [4.5,9> and
 * [9, infinity). In the first interval a rational approximation
 * R(x) is employed to compute y0(x)  = R(x) + 2/pi * log(x) * j0(x).
 *
 * In the second interval, the approximation is
 *     (x - p)(x - q)(x - r)(x - s)P9(x)/Q10(x)
 * where p, q, r, s are zeros of y1(x).
 *
 * The third interval uses the same approximations to modulus
 * and phase as j1(x), whence y1(x) = modulus * sin(phase).
 *
 * ACCURACY:
 *
 *  Absolute error, when y0(x) < 1; else relative error:
 *
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0, 30       36000       2.7e-19     5.3e-20
 *
 */



/*                                                      jnl.c
 *
 *      Bessel function of integer order
 *
 *
 *
 * SYNOPSIS:
 *
 * int n;
 * long double x, y, jnl();
 *
 * y = jnl( n, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of order n, where n is a
 * (possibly negative) integer.
 *
```

```
 * The ratio of jn(x) to j0(x) is computed by backward
 * recurrence.  First the ratio jn/jn-1 is found by a
 * continued fraction expansion.  Then the recurrence
 * relating successive orders is applied until j0 or j1 is
 * reached.
 *
 * If n = 0 or 1 the routine for j0 or j1 is called
 * directly.
 *
 *
 *
 * ACCURACY:
 *
 *                      Absolute error:
 * arithmetic   domain      # trials      peak         rms
 *    IEEE     -30, 30        5000       3.3e-19     4.7e-20
 *
 *
 * Not suitable for large n or x.
 *
 */



/*                                                      ldrand.c
 *
 *      Pseudorandom number generator
 *
 *
 *
 * SYNOPSIS:
 *
 * double y;
 * int ldrand();
 *
 * ldrand( &y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Yields a random number 1.0 < = y < 2.0.
 *
 * The three-generator congruential algorithm by Brian
 * Wichmann and David Hill (BYTE magazine, March, 1987,
 * pp 127-8) is used.
 *
 * Versions invoked by the different arithmetic compile
 * time options IBMPC, and MIEEE, produce the same sequences.
 *
 */



/*                                                      log10l.c
 *
 *      Common logarithm, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, log10l();
 *
 * y = log10l( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the base 10 logarithm of x.
 *
 * The argument is separated into its exponent and fractional
 * parts.  If the exponent is between -1 and +1, the logarithm
 * of the fraction is approximated by
 *
 *     log(1+x) = x - 0.5 x**2 + x**3 P(x)/Q(x).
 *
 * Otherwise, setting  z = 2(x-1)/x+1),
 *
 *     log(x) = z + z**3 P(z)/Q(z).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0.5, 2.0     30000      9.0e-20     2.6e-20
 *    IEEE     exp(+-10000)  30000      6.0e-20     2.3e-20
 *
 * In the tests over the interval exp(+-10000), the logarithms
 * of the random arguments were uniformly distributed over
 * [-10000, +10000].
 *
 * ERROR MESSAGES:
 *
 * log singularity:  x = 0; returns MINLOG
 * log domain:       x < 0; returns MINLOG
 */
```

```
/*							log1pl.c
 *
 *	Relative error logarithm
 *	Natural logarithm of 1+x, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, log1pl();
 *
 * y = log1pl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the base e (2.718...) logarithm of 1+x.
 *
 * The argument 1+x is separated into its exponent and fractional
 * parts.  If the exponent is between -1 and +1, the logarithm
 * of the fraction is approximated by
 *
 *     log(1+x) = x - 0.5 x^2 + x^3 P(x)/Q(x).
 *
 * Otherwise, setting  z = 2(x-1)/x+1),
 *
 *     log(x) = z + z^3 P(z)/Q(z).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     -1.0, 9.0    100000      8.2e-20    2.5e-20
 *
 * ERROR MESSAGES:
 *
 * log singularity:  x-1 = 0; returns -INFINITYL
 * log domain:       x-1 < 0; returns NANL
 */



/*							log2l.c
 *
 *	Base 2 logarithm, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, log2l();
 *
 * y = log2l( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the base 2 logarithm of x.
 *
 * The argument is separated into its exponent and fractional
 * parts.  If the exponent is between -1 and +1, the (natural)
 * logarithm of the fraction is approximated by
 *
 *     log(1+x) = x - 0.5 x**2 + x**3 P(x)/Q(x).
 *
 * Otherwise, setting  z = 2(x-1)/x+1),
 *
 *     log(x) = z + z**3 P(z)/Q(z).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0.5, 2.0     30000      9.8e-20     2.7e-20
 *    IEEE     exp(+-10000)  70000      5.4e-20     2.3e-20
 *
 * In the tests over the interval exp(+-10000), the logarithms
 * of the random arguments were uniformly distributed over
 * [-10000, +10000].
 *
 * ERROR MESSAGES:
 *
 * log singularity:  x = 0; returns -INFINITYL
 * log domain:       x < 0; returns NANL
 */



/*							logl.c
 *
 *	Natural logarithm, long double precision
 *
```

```
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, logl();
 *
 * y = logl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the base e (2.718...) logarithm of x.
 *
 * The argument is separated into its exponent and fractional
 * parts.  If the exponent is between -1 and +1, the logarithm
 * of the fraction is approximated by
 *
 *     log(1+x) = x - 0.5 x**2 + x**3 P(x)/Q(x).
 *
 * Otherwise, setting  z = 2(x-1)/x+1),
 *
 *     log(x) = z + z**3 P(z)/Q(z).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0.5, 2.0    150000      8.71e-20    2.75e-20
 *    IEEE     exp(+-10000) 100000      5.39e-20    2.34e-20
 *
 * In the tests over the interval exp(+-10000), the logarithms
 * of the random arguments were uniformly distributed over
 * [-10000, +10000].
 *
 * ERROR MESSAGES:
 *
 * log singularity:  x = 0; returns -INFINITYL
 * log domain:       x < 0; returns NANL
 */



/*                                                      mtherr.c
 *
 *      Library common error handling routine
 *
 *
 *
 * SYNOPSIS:
 *
 * char *fctnam;
 * int code;
 * int 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
 *
 */



/*                                                      nbdtrl.c
 *
 *      Negative binomial distribution
```

```
 *
 *
 *
 * SYNOPSIS:
 *
 * int k, n;
 * long double p, y, nbdtrl();
 *
 * y = nbdtrl( 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:
 *
 * Tested at random points (k,n,p) with k and n between 1 and 10,000
 * and p between 0 and 1.
 *
 * arithmetic   domain     # trials      peak         rms
 *    Absolute error:
 *    IEEE      0,10000     10000       9.8e-15     2.1e-16
 *
 */



/*                                                      nbdtrcl.c
 *
 *      Complemented negative binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k, n;
 * long double p, y, nbdtrcl();
 *
 * y = nbdtrcl( 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:
 *
 * See incbetl.c.
 *
 */



/*                                                      nbdtril
 *
 *      Functional inverse of negative binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k, n;
```

```
 * long double p, y, nbdtril();
 *
 * p = nbdtril( k, n, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Finds the argument p such that nbdtr(k,n,p) is equal to y.
 *
 * ACCURACY:
 *
 * Tested at random points (a,b,y), with y between 0 and 1.
 *
 *               a,b                     Relative error:
 * arithmetic  domain     # trials      peak         rms
 *    IEEE     0,100
 * See also incbil.c.
 */



/*                                                      ndtril.c
 *
 *      Inverse of Normal distribution function
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, ndtril();
 *
 * x = ndtril( 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 log(y) );  then the approximation is
 * x = z - log(z)/z  - (1/z) P(1/z) / Q(1/z) .
 * For larger arguments,  x/sqrt(2 pi) = w + w^3 R(w^2)/S(w^2)) ,
 * where w = y - 0.5 .
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain        # trials      peak         rms
 *  Arguments uniformly distributed:
 *    IEEE       0, 1           5000       7.8e-19     9.9e-20
 *  Arguments exponentially distributed:
 *    IEEE     exp(-11355),-1  30000       1.7e-19     4.3e-20
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition    value returned
 * ndtril domain      x <= 0        -MAXNUML
 * ndtril domain      x >= 1         MAXNUML
 *
 */



/*                                                      ndtrl.c
 *
 *      Normal distribution function
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, ndtrl();
 *
 * y = ndtrl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the area under the Gaussian probability density
 * function, integrated from minus infinity to x:
 *
 *                            x
 *                             -
 *                   1        | |          2
 *    ndtr(x)  = ---------    |    exp( - t /2 ) dt
 *               sqrt(2pi)  | |
 *                           -
 *                          -inf.
 *
 *             =  ( 1 + erf(z) ) / 2
 *             =  erfc(z) / 2
 *
 * where z = x/sqrt(2). Computation is via the functions
```

```
 * erf and erfc with care to avoid error amplification in computing exp(-x^2).
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     -13,0        30000       7.7e-19     1.0e-19
 *    IEEE     -106.5,-2    30000       4.2e-19     7.2e-20
 *    IEEE       0,3        30000       1.0e-19     2.4e-20
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition           value returned
 * erfcl underflow    x^2 / 2 > MAXLOGL        0.0
 *
 */



/*                                                     erfl.c
 *
 *      Error function
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, erfl();
 *
 * y = erfl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * The integral is
 *
 *                           x
 *                            -
 *                 2         | |          2
 *   erf(x)  =  --------     |    exp( - t  ) dt.
 *              sqrt(pi)   | |
 *                          -
 *                           0
 *
 * The magnitude of x is limited to about 106.56 for IEEE
 * arithmetic; 1 or -1 is returned outside this range.
 *
 * For 0 <= |x| < 1, erf(x) = x * P6(x^2)/Q6(x^2); otherwise
 * erf(x) = 1 - erfc(x).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,1         50000       2.0e-19     5.7e-20
 *
 */



/*                                                     erfcl.c
 *
 *      Complementary error function
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, erfcl();
 *
 * y = erfcl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 *
 *  1 - erf(x) =
 *
 *                           inf.
 *                             -
 *                  2         | |          2
 *   erfc(x)  =  --------     |    exp( - t  ) dt
 *               sqrt(pi)   | |
 *                           -
 *                            x
 *
 *
 * For small x, erfc(x) = 1 - erf(x); otherwise rational
 * approximations are computed.
 *
 * A special function expx2l.c is used to suppress error amplification
 * in computing exp(-x^2).
 *
 *
```

```
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,13        50000      8.4e-19      9.7e-20
 *    IEEE      6,106.56    20000      2.9e-19      7.1e-20
 *
 *
 * ERROR MESSAGES:
 *
 *   message          condition              value returned
 * erfcl underflow    x^2 > MAXLOGL              0.0
 *
 *
 */



/*                                                     pdtrl.c
 *
 *      Poisson distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k;
 * long double m, y, pdtrl();
 *
 * y = pdtrl( 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:
 *
 * See igamc().
 *
 */



/*                                                     pdtrcl()
 *
 *      Complemented poisson distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k;
 * long double m, y, pdtrcl();
 *
 * y = pdtrcl( 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:
 *
 * See igam.c.
```

```
 *
 */



/*                                                      pdtril()
 *
 *      Inverse Poisson distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k;
 * long double m, y, pdtrl();
 *
 * m = pdtril( 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:
 *
 * See igami.c.
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * pdtri domain    y < 0 or y >= 1       0.0
 *                     k < 0
 *
 */



/*                                                      polevll.c
 *                                                      p1evll.c
 *
 *      Evaluate polynomial
 *
 *
 *
 * SYNOPSIS:
 *
 * int N;
 * long double x, y, coef[N+1], polevl[];
 *
 * y = polevll( x, coef, N );
 *
 *
 *
 * DESCRIPTION:
 *
 * Evaluates polynomial of degree N:
 *
 *                     2          N
 * y  =  C  + C x + C x  +...+ C x
 *        0    1     2          N
 *
 * Coefficients are stored in reverse order:
 *
 * coef[0] = C  , ..., coef[N] = C  .
 *            N                   0
 *
 *  The function p1evll() assumes that coef[N] = 1.0 and is
 * omitted from the array.  Its calling arguments are
 * otherwise the same as polevll().
 *
 *  This module also contains the following globally declared constants:
 * MAXNUML = 1.189731495357231765021263853E4932L;
 * MACHEPL = 5.42101086242752217003726400434970855712890625E-20L;
 * MAXLOGL =  1.1356523406294143949492E4L;
 * MINLOGL = -1.135513711193302405887 3E4L;
 * LOGE2L  = 6.9314718055994530941723E-1L;
 * LOG2EL  = 1.4426950408889634073599E0L;
 * PIL     = 3.1415926535897932384626L;
 * PIO2L   = 1.5707963267948966192313L;
 * PIO4L   = 7.8539816339744830961566E-1L;
 *
 * SPEED:
 *
 * In the interest of speed, there are no checks for out
 * of bounds arithmetic.  This routine is used by most of
 * the functions in the library.  Depending on available
 * equipment features, the user may wish to rewrite the
 * program in microcode or assembly language.
```

```
 *
 */



/*							powil.c
 *
 *	Real raised to integer power, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, powil();
 * int n;
 *
 * y = powil( x, n );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns argument x raised to the nth power.
 * The routine efficiently decomposes n as a sum of powers of
 * two. The desired power is a product of two-to-the-kth
 * powers of x.  Thus to compute the 32767 power of x requires
 * 28 multiplications instead of 32767 multiplications.
 *
 *
 *
 * ACCURACY:
 *
 *
 *                      Relative error:
 * arithmetic   x domain   n domain  # trials      peak         rms
 *    IEEE     .001,1000  -1022,1023  50000       4.3e-17     7.8e-18
 *    IEEE        1,2     -1022,1023  20000       3.9e-17     7.6e-18
 *    IEEE     .99,1.01     0,8700    10000       3.6e-16     7.2e-17
 *
 * Returns MAXNUM on overflow, zero on underflow.
 *
 */



/*							powl.c
 *
 *	Power function, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, z, powl();
 *
 * z = powl( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Computes x raised to the yth power.  Analytically,
 *
 *      x**y  =  exp( y log(x) ).
 *
 * Following Cody and Waite, this program uses a lookup table
 * of 2**-i/32 and pseudo extended precision arithmetic to
 * obtain several extra bits of accuracy in both the logarithm
 * and the exponential.
 *
 *
 *
 * ACCURACY:
 *
 * The relative error of pow(x,y) can be estimated
 * by   y dl ln(2),   where dl is the absolute error of
 * the internally computed base 2 logarithm.  At the ends
 * of the approximation interval the logarithm equal 1/32
 * and its relative error is about 1 lsb = 1.1e-19.  Hence
 * the predicted relative error in the result is 2.3e-21 y .
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *
 *    IEEE     +-1000       40000      2.8e-18      3.7e-19
 * .001 < x < 1000, with log(x) uniformly distributed.
 * -1000 < y < 1000, y uniformly distributed.
 *
 *    IEEE     0,8700       60000      6.5e-18      1.0e-18
 * 0.99 < x < 1.01, 0 < y < 8700, uniformly distributed.
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * pow overflow     x**y > MAXNUM      INFINITY
 * pow underflow   x**y < 1/MAXNUM       0.0
 * pow domain      x<0 and y noninteger  0.0
 *
 */
```

```
/*                                                      sinhl.c
 *
 *      Hyperbolic sine, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, sinhl();
 *
 * y = sinhl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns hyperbolic sine of argument in the range MINLOGL to
 * MAXLOGL.
 *
 * The range is partitioned into two segments.  If |x| <= 1, a
 * rational function of the form x + x**3 P(x)/Q(x) is employed.
 * Otherwise the calculation is sinh(x) = ( exp(x) - exp(-x) )/2.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       -2,2       10000       1.5e-19     3.9e-20
 *    IEEE     +-10000      30000       1.1e-19     2.8e-20
 *
 */



/*                                                      sinl.c
 *
 *      Circular sine, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, sinl();
 *
 * y = sinl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Range reduction is into intervals of pi/4.  The reduction
 * error is nearly eliminated by contriving an extended precision
 * modular arithmetic.
 *
 * Two polynomial approximating functions are employed.
 * Between 0 and pi/4 the sine is approximated by the Cody
 * and Waite polynomial form
 *      x + x**3 P(x**2) .
 * Between pi/4 and pi/2 the cosine is represented as
 *      1 - .5 x**2 + x**4 Q(x**2) .
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain      # trials      peak         rms
 *    IEEE     +-5.5e11      200,000    1.2e-19     2.9e-20
 *
 * ERROR MESSAGES:
 *
 *   message           condition        value returned
 * sin total loss   x > 2**39               0.0
 *
 * Loss of precision occurs for x > 2**39 = 5.49755813888e11.
 * The routine as implemented flags a TLOSS error for
 * x > 2**39 and returns 0.0.
 */



/*                                                      cosl.c
 *
 *      Circular cosine, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, cosl();
 *
 * y = cosl( x );
 *
 *
 *
 * DESCRIPTION:
 *
```

```
 * Range reduction is into intervals of pi/4.  The reduction
 * error is nearly eliminated by contriving an extended precision
 * modular arithmetic.
 *
 * Two polynomial approximating functions are employed.
 * Between 0 and pi/4 the cosine is approximated by
 *      1 - .5 x**2 + x**4 Q(x**2) .
 * Between pi/4 and pi/2 the sine is represented by the Cody
 * and Waite polynomial form
 *      x  +  x**3 P(x**2) .
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain      # trials      peak         rms
 *    IEEE     +-5.5e11       50000      1.2e-19     2.9e-20
 */



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



/*                                                      stdtrl.c
 *
 *      Student's t distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * long double p, t, stdtrl();
 * int k;
 *
 * p = stdtrl( 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 < -1.6, 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:
 *
 * Tested at random 1 <= k <= 100.  The "domain" refers to t.
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     -100,-1.6    10000       5.7e-18     9.8e-19
 *    IEEE     -1.6,100     10000       3.8e-18     1.0e-19
 */



/*                                                      stdtril.c
 *
 *      Functional inverse of Student's t distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * long double p, t, stdtril();
 * int k;
 *
 * t = stdtril( k, p );
 *
 *
 * DESCRIPTION:
 *
 * Given probability p, finds the argument t such that stdtrl(k,t)
 * is equal to p.
 *
 * ACCURACY:
 *
 * Tested at random 1 <= k <= 100.  The "domain" refers to p:
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,1        3500       4.2e-17     4.1e-18
 */



/*                                                      tanhl.c
 *
 *      Hyperbolic tangent, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, tanhl();
 *
 * y = tanhl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns hyperbolic tangent of argument in the range MINLOGL to
 * MAXLOGL.
 *
 * A rational function is used for |x| < 0.625.  The form
 * x + x**3 P(x)/Q(x) of Cody & Waite is employed.
 * Otherwise,
 *    tanh(x) = sinh(x)/cosh(x) = 1  -  2/(exp(2x) + 1).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -2,2        30000       1.3e-19     2.4e-20
 *
 */



/*                                                      tanl.c
 *
 *      Circular tangent, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, tanl();
 *
 * y = tanl( x );
 *
 *
 *
 * DESCRIPTION:
 *
```

```
 * Returns the circular tangent of the radian argument x.
 *
 * Range reduction is modulo pi/4.  A rational function
 *       x + x**3 P(x**2)/Q(x**2)
 * is employed in the basic interval [0, pi/4].
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     +-1.07e9       30000     1.9e-19     4.8e-20
 *
 * ERROR MESSAGES:
 *
 *   message         condition          value returned
 * tan total loss   x > 2^39                0.0
 *
 */



/*                                                      cotl.c
 *
 *      Circular cotangent, long double precision
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, cotl();
 *
 * y = cotl( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the circular cotangent of the radian argument x.
 *
 * Range reduction is modulo pi/4.  A rational function
 *       x + x**3 P(x**2)/Q(x**2)
 * is employed in the basic interval [0, pi/4].
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     +-1.07e9      30000      1.9e-19     5.1e-20
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition          value returned
 * cot total loss   x > 2^39                0.0
 * cot singularity  x = 0                  INFINITYL
 *
 */



/*                                                      unityl.c
 *
 * Relative error approximations for function arguments near
 * unity.
 *
 *    cosm1(x) = cos(x) - 1
 *
 */



/*                                                      ynl.c
 *
 *      Bessel function of second kind of integer order
 *
 *
 *
 * SYNOPSIS:
 *
 * long double x, y, ynl();
 * int n;
 *
 * y = ynl( 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
 * y0l() and y1l().
 *
 * If n = 0 or 1 the routine for y0l or y1l is called
```

```
 * directly.
 *
 *
 *
 * ACCURACY:
 *
 *
 *       Absolute error, except relative error when y > 1.
 *       x >= 0,  -30 <= n <= +30.
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     -30, 30       10000       1.3e-18     1.8e-19
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * ynl singularity   x = 0              MAXNUML
 * ynl overflow                         MAXNUML
 *
 * Spot checked against tables for x, n between 0 and 100.
 *
 */
```

To Cephes home page www.moshier.net:

Get ldouble.zip:

Last update: 5 October 2014