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.

Double Precision Special Functions

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

- acosh, Inverse hyperbolic cosine
- airy, Airy functions
- asin, Inverse circular sine
- acos, Inverse circular cosine
- asinh, Inverse hyperbolic sine
- atan, Inverse circular tangent
- atan2, Quadrant correct inverse circular tangent
- atanh, Inverse hyperbolic tangent
- bdtr, Binomial distribution
- bdtrc, Complemented binomial distribution
- bdtri, Inverse binomial distribution
- beta, Beta function
- btdtr, Beta distribution
- cbrt, Cube root
- chbevl, Evaluate Chebyshev series
- chdtr, Chi-square distribution
- chdtrc, Complemented Chi-square distribution
- chdtri, Inverse of complemented Chi-square distribution
- cheby, Find Chebyshev coefficients
- 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
- csinh, Complex hyperbolic sine
- casinh, Complex inverse hyperbolic sine
- ccosh, Complex hyperbolic cosine
- cacosh, Complex inverse hyperbolic cosine
- ctanh, Complex hyperbolic tangent
- catanh, Complex inverse hyperbolic tangent
- cpow, Complex power function
- cmplx, Complex number arithmetic
- cabs, Complex absolute value
- csqrt, Complex square root
- const, Globally declared constants
- cosh, Hyperbolic cosine
- dawsn, Dawson's Integral
- drand, Pseudorandom number generator
- ei, Exponential Integral
- eigens, Eigenvalues and eigenvectors of a real symmetric matrix
- 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
- euclid, Rational arithmetic routines
- exp, Exponential function
- exp10, Base 10 exponential function
- exp2, Base 2 exponential function
- expn, Exponential integral En
- expx2, Exponential of squared argument
- fabs, Absolute value
- fac, Factorial function
- fdtr, F distribution
- fdtrc, Complemented F distribution
- fdtri, Inverse of complemented F distribution
- fftr, Fast Fourier transform
- floor, Floor function
- ceil, Ceil function
- frexp, Extract exponent
- ldexp, Apply exponent
- fresnl, Fresnel integral
- gamma, Gamma function

```
/*                                                      acosh.c
 *
 *      Inverse hyperbolic cosine
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, acosh();
 *
 * y = acosh( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns inverse hyperbolic cosine of argument.
 *
 * If 1 <= x < 1.5, a rational approximation
 *
 *      sqrt(z) * 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
 *    DEC       1,3         30000       4.2e-17     1.1e-17
 *    IEEE      1,3         30000       4.6e-16     8.7e-17
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * acosh domain       |x| < 1            NAN
 *
 */
```

```
/*                                                      airy.c
 *
 *      Airy function
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, ai, aip, bi, bip;
 * int airy();
 *
 * airy( x, &ai, &aip, &bi, &bip );
 *
 *
 *
 * DESCRIPTION:
 *
 * Solution of the differential equation
 *
 *      y"(x) = xy.
 *
 * The function returns the two independent solutions Ai, Bi
 * and their first derivatives Ai'(x), Bi'(x).
 *
 * Evaluation is by power series summation for small x,
 * by rational minimax approximations for large x.
 *
 *
 *
 * ACCURACY:
 * Error criterion is absolute when function <= 1, relative
 * when function > 1, except * denotes relative error criterion.
 * For large negative x, the absolute error increases as x^1.5.
 * For large positive x, the relative error increases as x^1.5.
 *
 * Arithmetic  domain   function  # trials      peak         rms
 * IEEE        -10, 0     Ai        10000       1.6e-15     2.7e-16
 * IEEE          0, 10    Ai        10000       2.3e-14*    1.8e-15*
 * IEEE        -10, 0     Ai'       10000       4.6e-15     7.6e-16
 * IEEE          0, 10    Ai'       10000       1.8e-14*    1.5e-15*
 * IEEE        -10, 10    Bi        30000       4.2e-15     5.3e-16
 * IEEE        -10, 10    Bi'       30000       4.9e-15     7.3e-16
 * DEC         -10, 0     Ai         5000       1.7e-16     2.8e-17
 * DEC           0, 10    Ai         5000       2.1e-15*    1.7e-16*
```

```
 * DEC         -10, 0     Ai'        5000       4.7e-16     7.8e-17
 * DEC           0, 10    Ai'       12000       1.8e-15*    1.5e-16*
 * DEC         -10, 10    Bi        10000       5.5e-16     6.8e-17
 * DEC         -10, 10    Bi'        7000       5.3e-16     8.7e-17
 *
 */



/*                                                      asin.c
 *
 *      Inverse circular sine
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, asin();
 *
 * y = asin( 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
 *    DEC      -1, 1        40000       2.6e-17     7.1e-18
 *    IEEE     -1, 1        10^6        1.9e-16     5.4e-17
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * asin domain        |x| > 1           NAN
 *
 */



/*                                                      acos()
 *
 *      Inverse circular cosine
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, acos();
 *
 * y = acos( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns radian angle between 0 and pi 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
 *    DEC       -1, 1       50000       3.3e-17     8.2e-18
 *    IEEE      -1, 1       10^6        2.2e-16     6.5e-17
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * asin domain        |x| > 1           NAN
 */



/*                                                      asinh.c
 *
 *      Inverse hyperbolic sine
```

```
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, asinh();
 *
 * y = asinh( 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
 *    DEC      -3,3         75000       4.6e-17     1.1e-17
 *    IEEE     -1,1         30000       3.7e-16     7.8e-17
 *    IEEE      1,3         30000       2.5e-16     6.7e-17
 *
 */



/*                                                      atan.c
 *
 *      Inverse circular tangent
 *      (arctangent)
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, atan();
 *
 * y = atan( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns radian angle between -pi/2 and +pi/2 whose tangent
 * is x.
 *
 * Range reduction is from three intervals into the interval
 * from zero to 0.66.  The approximant uses a rational
 * function of degree 4/5 of the form x + x**3 P(x)/Q(x).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    DEC       -10, 10     50000       2.4e-17     8.3e-18
 *    IEEE      -10, 10      10^6       1.8e-16     5.0e-17
 *
 */



/*                                                      atan2()
 *
 *      Quadrant correct inverse circular tangent
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, z, atan2();
 *
 * z = atan2( 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      10^6       2.5e-16     6.9e-17
 * See atan.c.
```

```
 *
 */



/*                                                      atanh.c
 *
 *      Inverse hyperbolic tangent
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, atanh();
 *
 * y = atanh( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns inverse hyperbolic tangent of argument in the range
 * MINLOG to MAXLOG.
 *
 * 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
 *    DEC       -1,1        50000       2.4e-17     6.4e-18
 *    IEEE      -1,1        30000       1.9e-16     5.2e-17
 *
 */



/*                                                      bdtr.c
 *
 *      Binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k, n;
 * double p, y, bdtr();
 *
 * y = bdtr( 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 (a,b,p), with p between 0 and 1.
 *
 *               a,b                     Relative error:
 * arithmetic  domain     # trials      peak         rms
 *  For p between 0.001 and 1:
 *    IEEE     0,100       100000      4.3e-15     2.6e-16
 * See also incbet.c.
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * bdtr domain         k < 0            0.0
 *                     n < k
 *                     x < 0, x > 1
 */



/*                                                      bdtrc()
 *
 *      Complemented binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
```

```
 * int k, n;
 * double p, y, bdtrc();
 *
 * y = bdtrc( 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:
 *
 * Tested at random points (a,b,p).
 *
 *               a,b                     Relative error:
 * arithmetic  domain     # trials      peak         rms
 *  For p between 0.001 and 1:
 *    IEEE     0,100       100000      6.7e-15     8.2e-16
 *  For p between 0 and .001:
 *    IEEE     0,100       100000      1.5e-13     2.7e-15
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * bdtrc domain      x<0, x>1, n<k       0.0
 */



/*                                                      bdtri()
 *
 *      Inverse binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k, n;
 * double p, y, bdtri();
 *
 * p = bdtr( 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:
 *
 * Tested at random points (a,b,p).
 *
 *               a,b                     Relative error:
 * arithmetic  domain     # trials      peak         rms
 *  For p between 0.001 and 1:
 *    IEEE     0,100       100000      2.3e-14     6.4e-16
 *    IEEE     0,10000     100000      6.6e-12     1.2e-13
 *  For p between 10^-6 and 0.001:
 *    IEEE     0,100       100000      2.0e-12     1.3e-14
 *    IEEE     0,10000     100000      1.5e-12     3.2e-14
 * See also incbi.c.
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * bdtri domain     k < 0, n <= k         0.0
 *                  x < 0, x > 1
 */



/*                                                      beta.c
 *
 *      Beta function
 *
 *
 *
 * SYNOPSIS:
 *
 * double a, b, y, beta();
 *
 * y = beta( a, b );
```

```
 *
 *
 *
 * DESCRIPTION:
 *
 *                     -     -
 *                    | (a) | (b)
 * beta( a, b )  =  -----------.
 *                       -
 *                      | (a+b)
 *
 * For large arguments the logarithm of the function is
 * evaluated using lgam(), then exponentiated.
 *
 *
 *
 * ACCURACY:
 *
 *                       Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    DEC        0,30        1700       7.7e-15     1.5e-15
 *    IEEE       0,30       30000       8.1e-14     1.1e-14
 *
 * ERROR MESSAGES:
 *
 *   message         condition          value returned
 * beta overflow    log(beta) > MAXLOG       0.0
 *                  a or b <0 integer        0.0
 *
 */



/*                                                      btdtr.c
 *
 *      Beta distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * double a, b, x, y, btdtr();
 *
 * y = btdtr( 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
 *
 *
 * This function is identical to the incomplete beta
 * integral function incbet(a, b, x).
 *
 * The complemented function is
 *
 * 1 - P(1-x)  =  incbet( b, a, x );
 *
 *
 * ACCURACY:
 *
 * See incbet.c.
 *
 */



/*                                                      cbrt.c
 *
 *      Cube root
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, cbrt();
 *
 * y = cbrt( 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
 *    DEC        -10,10     200000      1.8e-17     6.2e-18
 *    IEEE       0,1e308     30000      1.5e-16     5.0e-17
 *
 */



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



/*                                                      chdtr.c
 *
 *      Chi-square distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * double df, x, y, chdtr();
 *
 * y = chdtr( 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
 */



/*                                                      chdtrc()
 *
 *      Complemented Chi-square distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * double v, x, y, chdtrc();
 *
 * y = chdtrc( 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
 */



/*                                                      chdtri()
 *
 *      Inverse of complemented Chi-square distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * double df, x, y, chdtri();
 *
 * x = chdtri( 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
 *
 */



/*      cheby.c
 *
 * Program to calculate coefficients of the Chebyshev polynomial
 * expansion of a given input function.  The algorithm computes
 * the discrete Fourier cosine transform of the function evaluated
 * at unevenly spaced points.  Library routine chbevl.c uses the
 * coefficients to calculate an approximate value of the original
 * function.
 */



/*                                                      clog.c
 *
 *      Complex natural logarithm
 *
 *
 *
 * SYNOPSIS:
 *
 * void clog();
 * cmplx z, w;
 *
 * clog( &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.
 */



/*                                                      cexp()
 *
 *      Complex exponential function
 *
 *
 *
 * SYNOPSIS:
 *
 * void cexp();
 * cmplx z, w;
 *
 * cexp( &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
 *
 */



/*                                                     csin()
 *
 *      Complex circular sine
 *
 *
 *
 * SYNOPSIS:
 *
 * void csin();
 * cmplx z, w;
 *
 * csin( &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.
 *
 */



/*                                                     ccos()
 *
 *      Complex circular cosine
 *
 *
 *
 * SYNOPSIS:
 *
 * void ccos();
 * cmplx z, w;
 *
 * ccos( &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
 */



/*                                                     ctan()
 *
 *      Complex circular tangent
 *
 *
 *
 * SYNOPSIS:
 *
 * void ctan();
 * cmplx z, w;
 *
 * ctan( &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.
 */



/*							ccot()
 *
 *	Complex circular cotangent
 *
 *
 *
 * SYNOPSIS:
 *
 * void ccot();
 * cmplx z, w;
 *
 * ccot( &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.
 */



/*							casin()
 *
 *	Complex circular arc sine
 *
 *
 *
 * SYNOPSIS:
 *
 * void casin();
 * cmplx z, w;
 *
 * casin( &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.
 */



/*							cacos()
 *
 *	Complex circular arc cosine
 *
 *
 *
 * SYNOPSIS:
 *
 * void cacos();
 * cmplx z, w;
 *
 * cacos( &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
 */



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



/*							csinh
 *
 *	Complex hyperbolic sine
 *
 *
 *
 * SYNOPSIS:
 *
 * void csinh();
 * cmplx z, w;
 *
 * csinh( &z, &w );
 *
```

```
 *
 * DESCRIPTION:
 *
 * csinh z = (cexp(z) - cexp(-z))/2
 *         = sinh x * cos y  +  i cosh x * sin y .
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10,+10     30000       3.1e-16     8.2e-17
 *
 */



/*                                                      casinh
 *
 *      Complex inverse hyperbolic sine
 *
 *
 *
 * SYNOPSIS:
 *
 * void casinh();
 * cmplx z, w;
 *
 * casinh (&z, &w);
 *
 *
 *
 * DESCRIPTION:
 *
 * casinh z = -i casin iz .
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10,+10     30000       1.8e-14     2.6e-15
 *
 */



/*                                                      ccosh
 *
 *      Complex hyperbolic cosine
 *
 *
 *
 * SYNOPSIS:
 *
 * void ccosh();
 * cmplx z, w;
 *
 * ccosh (&z, &w);
 *
 *
 *
 * DESCRIPTION:
 *
 * ccosh(z) = cosh x  cos y + i sinh x sin y .
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10,+10     30000       2.9e-16     8.1e-17
 *
 */



/*                                                      cacosh
 *
 *      Complex inverse hyperbolic cosine
 *
 *
 *
 * SYNOPSIS:
 *
 * void cacosh();
 * cmplx z, w;
 *
 * cacosh (&z, &w);
 *
 *
 *
 * DESCRIPTION:
 *
 * acosh z = i acos z .
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10,+10     30000       1.6e-14     2.1e-15
 *
 */
```

```
/*                                                      ctanh
 *
 *      Complex hyperbolic tangent
 *
 *
 *
 * SYNOPSIS:
 *
 * void ctanh();
 * cmplx z, w;
 *
 * ctanh (&z, &w);
 *
 *
 *
 * DESCRIPTION:
 *
 * tanh z = (sinh 2x  +  i sin 2y) / (cosh 2x + cos 2y) .
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10,+10     30000       1.7e-14     2.4e-16
 *
 */



/*                                                      catanh
 *
 *      Complex inverse hyperbolic tangent
 *
 *
 *
 * SYNOPSIS:
 *
 * void catanh();
 * cmplx z, w;
 *
 * catanh (&z, &w);
 *
 *
 *
 * DESCRIPTION:
 *
 * Inverse tanh, equal to  -i catan (iz);
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10,+10     30000       2.3e-16     6.2e-17
 *
 */



/*                                                      cpow
 *
 *      Complex power function
 *
 *
 *
 * SYNOPSIS:
 *
 * void cpow();
 * cmplx a, z, w;
 *
 * cpow (&a, &z, &w);
 *
 *
 *
 * DESCRIPTION:
 *
 * Raises complex A to the complex Zth power.
 * Definition is per AMS55 # 4.2.8,
 * analytically equivalent to cpow(a,z) = cexp(z clog(a)).
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -10,+10     30000       9.4e-15     1.5e-15
 *
 */



/*                                                      cmplx.c
 *
 *      Complex number arithmetic
 *
 *
 *
 * SYNOPSIS:
 *
 * typedef struct {
```

```
 *      double r;     real part
 *      double i;     imaginary part
 *     }cmplx;
 *
 * cmplx *a, *b, *c;
 *
 * cadd( a, b, c );     c = b + a
 * csub( a, b, c );     c = b - a
 * cmul( a, b, c );     c = b * a
 * cdiv( a, b, c );     c = b / a
 * cneg( c );           c = -c
 * cmov( 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
 */


/*                                                      cabs()
 *
 *      Complex absolute value
 *
 *
 *
 * SYNOPSIS:
 *
 * double cabs();
 * cmplx z;
 * double a;
 *
 * a = cabs( &z );
 *
 *
 *
 * DESCRIPTION:
 *
 *
 * If z = x + iy
 *
 * then
 *
 *       a = sqrt( x**2 + y**2 ).
 *
 * Overflow and underflow are avoided by testing the magnitudes
 * of x and y before squaring.  If either is outside half of
 * the floating point full scale range, both are rescaled.
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    DEC       -30,+30     30000       3.2e-17     9.2e-18
 *    IEEE      -10,+10    100000       2.7e-16     6.9e-17
 */


/*                                                      csqrt()
 *
 *      Complex square root
 *
 *
 *
```

```
 * SYNOPSIS:
 *
 * void csqrt();
 * cmplx z, w;
 *
 * csqrt( &z, &w );
 *
 *
 *
 * DESCRIPTION:
 *
 *
 * If z = x + iy,  r = |z|, then
 *
 *                        1/2
 * Im w  =  [ (r - x)/2 ]   ,
 *
 * Re w  =  y / 2 Im w.
 *
 *
 * Note that -w is also a square root of z.  The root chosen
 * is always in the upper half plane.
 *
 * Because of the potential for cancellation error in r - x,
 * the result is sharpened by doing a Heron iteration
 * (see sqrt.c) in complex arithmetic.
 *
 *
 *
 * ACCURACY:
 *
 *                       Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    DEC       -10,+10     25000       3.2e-17     9.6e-18
 *    IEEE      -10,+10    100000       3.2e-16     7.7e-17
 *
 *                        2
 * Also tested by csqrt( z ) = z, and tested by arguments
 * close to the real axis.
 */



/*                                                      const.c
 *
 *     Globally declared constants
 *
 *
 *
 * SYNOPSIS:
 *
 * extern double nameofconstant;
 *
 *
 *
 *
 * DESCRIPTION:
 *
 * This file contains a number of mathematical constants and
 * also some needed size parameters of the computer arithmetic.
 * The values are supplied as arrays of hexadecimal integers
 * for IEEE arithmetic; arrays of octal constants for DEC
 * arithmetic; and in a normal decimal scientific notation for
 * other machines.  The particular notation used is determined
 * by a symbol (DEC, IBMPC, or UNK) defined in the include file
 * mconf.h.
 *
 * The default size parameters are as follows.
 *
 * For DEC and UNK modes:
 * MACHEP =  1.38777878078144567553E-17       2**-56
 * MAXLOG =  8.8029691931113054295988E1       log(2**127)
 * MINLOG = -8.872283911167299960540E1        log(2**-128)
 * MAXNUM =  1.701411834604692317316873e38    2**127
 *
 * For IEEE arithmetic (IBMPC):
 * MACHEP =  1.11022302462515654042E-16       2**-53
 * MAXLOG =  7.09782712893383996843E2         log(2**1024)
 * MINLOG = -7.08396418532264106224E2         log(2**-1022)
 * MAXNUM =  1.7976931348623158E308           2**1024
 *
 * The global symbols for mathematical constants are
 * PI     =  3.14159265358979323846           pi
 * PIO2   =  1.57079632679489661923           pi/2
 * PIO4   =  7.85398163397448309616E-1        pi/4
 * SQRT2  =  1.41421356237309504880           sqrt(2)
 * SQRTH  =  7.07106781186547524401E-1        sqrt(2)/2
 * LOG2E  =  1.4426950408889634073599         1/log(2)
 * SQ2OPI =  7.9788456080286535587989E-1      sqrt( 2/pi )
 * LOGE2  =  6.93147180559945309417E-1        log(2)
 * LOGSQ2 =  3.46573590279972654709E-1        log(2)/2
 * THPIO4 =  2.35619449019234492885           3*pi/4
 * TWOOPI =  6.36619772367581343075535E-1     2/pi
 *
 * These lists are subject to change.
 */
```

```
/*                                                      cosh.c
 *
 *      Hyperbolic cosine
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, cosh();
 *
 * y = cosh( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns hyperbolic cosine of argument in the range MINLOG to
 * MAXLOG.
 *
 * cosh(x)  =  ( exp(x) + exp(-x) )/2.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    DEC       +- 88       50000       4.0e-17     7.7e-18
 *    IEEE     +-MAXLOG     30000       2.6e-16     5.7e-17
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * cosh overflow    |x| > MAXLOG       MAXNUM
 *
 *
 */
```

```
/*                                                      dawsn.c
 *
 *      Dawson's Integral
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, dawsn();
 *
 * y = dawsn( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Approximates the integral
 *
 *                             x
 *                             -
 *                      2     | |        2
 *  dawsn(x)  =  exp( -x  )   |    exp( t  ) dt
 *                          | |
 *                           -
 *                           0
 *
 * Three different rational approximations are employed, for
 * the intervals 0 to 3.25; 3.25 to 6.25; and 6.25 up.
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,10        10000       6.9e-16     1.0e-16
 *    DEC       0,10         6000       7.4e-17     1.4e-17
 *
 *
 */
```

```
/*                                                      drand.c
 *
 *      Pseudorandom number generator
 *
 *
 *
 * SYNOPSIS:
 *
 * double y, drand();
 *
 * drand( &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. The period, given by them, is
 * 6953607871644.
 *
 * Versions invoked by the different arithmetic compile
 * time options DEC, IBMPC, and MIEEE, produce
 * approximately the same sequences, differing only in the
 * least significant bits of the numbers. The UNK option
 * implements the algorithm as recommended in the BYTE
 * article.  It may be used on all computers. However,
 * the low order bits of a double precision number may
 * not be adequately random, and may vary due to arithmetic
 * implementation details on different computers.
 *
 * The other compile options generate an additional random
 * integer that overwrites the low order bits of the double
 * precision number.  This reduces the period by a factor of
 * two but tends to overcome the problems mentioned.
 *
 */



/*                                                      ei.c
 *
 *      Exponential integral
 *
 *
 * SYNOPSIS:
 *
 * double x, y, ei();
 *
 * y = ei( x );
 *
 *
 *
 * DESCRIPTION:
 *
 *               x
 *                -     t
 *               | |   e
 *    Ei(x) =   -|-   ---  dt .
 *             | |     t
 *              -
 *             -inf
 *
 * Not defined for x <= 0.
 * See also expn.c.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE       0,100       50000      8.6e-16     1.3e-16
 *
 */



/*                                                      eigens.c
 *
 *      Eigenvalues and eigenvectors of a real symmetric matrix
 *
 *
 *
 * SYNOPSIS:
 *
 * int n;
 * double A[n*(n+1)/2], EV[n*n], E[n];
 * void eigens( A, EV, E, n );
 *
 *
 *
 * DESCRIPTION:
 *
 * The algorithm is due to J. vonNeumann.
 *
 * A[] is a symmetric matrix stored in lower triangular form.
 * That is, A[ row, column ] = A[ (row*row+row)/2 + column ]
 * or equivalently with row and column interchanged.  The
 * indices row and column run from 0 through n-1.
 *
 * EV[] is the output matrix of eigenvectors stored columnwise.
 * That is, the elements of each eigenvector appear in sequential
 * memory order.  The jth element of the ith eigenvector is
 * EV[ n*i+j ] = EV[i][j].
 *
 * E[] is the output matrix of eigenvalues.  The ith element
 * of E corresponds to the ith eigenvector (the ith row of EV).
 *
 * On output, the matrix A will have been diagonalized and its
 * orginal contents are destroyed.
 *
 * ACCURACY:
 *
 * The error is controlled by an internal parameter called RANGE
```

```
 * which is set to 1e-10.  After diagonalization, the
 * off-diagonal elements of A will have been reduced by
 * this factor.
 *
 * ERROR MESSAGES:
 *
 * None.
 *
 */



/*                                                      ellie.c
 *
 *      Incomplete elliptic integral of the second kind
 *
 *
 *
 * SYNOPSIS:
 *
 * double phi, m, y, ellie();
 *
 * y = ellie( 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
 *    DEC        0,2         2000       1.9e-16     3.4e-17
 *    IEEE     -10,10      150000       3.3e-15     1.4e-16
 *
 *
 */



/*                                                      ellik.c
 *
 *      Incomplete elliptic integral of the first kind
 *
 *
 *
 * SYNOPSIS:
 *
 * double phi, m, y, ellik();
 *
 * y = ellik( 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       200000      7.4e-16     1.0e-16
 *
 *
 */



/*                                                      ellpe.c
 *
 *      Complete elliptic integral of the second kind
 *
 *
 *
 * SYNOPSIS:
 *
 * double m1, y, ellpe();
 *
 * y = ellpe( 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
 *    DEC        0, 1       13000       3.1e-17     9.4e-18
 *    IEEE       0, 1       10000       2.1e-16     7.3e-17
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * ellpe domain      x<0, x>1            0.0
 *
 */



/*                                                      ellpj.c
 *
 *      Jacobian Elliptic Functions
 *
 *
 *
 * SYNOPSIS:
 *
 * double u, m, sn, cn, dn, phi;
 * int ellpj();
 *
 * ellpj( u, m, &sn, &cn, &dn, &phi );
 *
 *
 *
 * DESCRIPTION:
 *
 *
 * Evaluates the Jacobian elliptic functions sn(u|m), cn(u|m),
 * and dn(u|m) of parameter m between 0 and 1, and real
 * argument u.
 *
 * These functions are periodic, with quarter-period on the
 * real axis equal to the complete elliptic integral
 * ellpk(1.0-m).
 *
 * Relation to incomplete elliptic integral:
 * If u = ellik(phi,m), then sn(u|m) = sin(phi),
 * and cn(u|m) = cos(phi).  Phi is called the amplitude of u.
 *
 * Computation is by means of the arithmetic-geometric mean
 * algorithm, except when m is within 1e-9 of 0 or 1.  In the
 * latter case with m close to 1, the approximation applies
 * only for phi < pi/2.
 *
 * ACCURACY:
 *
```

```
 * Tested at random points with u between 0 and 10, m between
 * 0 and 1.
 *
 *            Absolute error (* = relative error):
 * arithmetic   function   # trials      peak         rms
 *    DEC       sn           1800       4.5e-16     8.7e-17
 *    IEEE      phi         10000       9.2e-16*    1.4e-16*
 *    IEEE      sn          50000       4.1e-15     4.6e-16
 *    IEEE      cn          40000       3.6e-15     4.4e-16
 *    IEEE      dn         100000       3.9e-15     1.7e-16
 *
 * Larger errors occur for m near 1.
 * Peak error observed in consistency check using addition
 * theorem for sn(u+v) was 4e-16 (absolute).  Also tested by
 * the above relation to the incomplete elliptic integral.
 * Accuracy deteriorates when u is large.
 *
 */



/*                                                     ellpk.c
 *
 *      Complete elliptic integral of the first kind
 *
 *
 *
 * SYNOPSIS:
 *
 * double m1, y, ellpk();
 *
 * y = ellpk( 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
 *    DEC        0,1        16000       3.5e-17     1.1e-17
 *    IEEE       0,1        30000       2.5e-16     6.8e-17
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * ellpk domain       x<0, x>1           0.0
 *
 */



/*                                                     euclid.c
 *
 *      Rational arithmetic routines
 *
 *
 *
 * SYNOPSIS:
 *
 *
 * typedef struct
 *      {
 *      double n;  numerator
 *      double d;  denominator
 *      }fract;
 *
 * radd( a, b, c )      c = b + a
 * rsub( a, b, c )      c = b - a
 * rmul( a, b, c )      c = b * a
 * rdiv( a, b, c )      c = b / a
 * euclid( &n, &d )     Reduce n/d to lowest terms,
 *                      return greatest common divisor.
 *
 * Arguments of the routines are pointers to the structures.
```

```
 * The double precision numbers are assumed, without checking,
 * to be integer valued.  Overflow conditions are reported.
 */



/*                                                      exp.c
 *
 *      Exponential function
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, exp();
 *
 * y = exp( 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  1 + 2x P(x**2)/( Q(x**2) - P(x**2) )
 * of degree 2/3 is used to approximate exp(f) in the basic
 * interval [-0.5, 0.5].
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    DEC       +- 88       50000       2.8e-17     7.0e-18
 *    IEEE      +- 708      40000       2.0e-16     5.6e-17
 *
 *
 * Error amplification in the exponential function can be
 * a serious matter.  The error propagation involves
 * exp( X(1+delta) ) = exp(X) ( 1 + X*delta + ... ),
 * which shows that a 1 lsb error in representing X produces
 * a relative error of X times 1 lsb in the function.
 * While the routine gives an accurate result for arguments
 * that are exactly represented by a double precision
 * computer number, the result contains amplified roundoff
 * error for large arguments not exactly represented.
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * exp underflow    x < MINLOG         0.0
 * exp overflow     x > MAXLOG         INFINITY
 *
 */



/*                                                      exp10.c
 *
 *      Base 10 exponential function
 *      (Common antilogarithm)
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, exp10();
 *
 * y = exp10( 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     -307,+307    30000       2.2e-16     5.5e-17
 * Test result from an earlier version (2.1):
 *    DEC       -38,+38     70000       3.1e-17     7.0e-18
```

```
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * exp10 underflow    x < -MAXL10        0.0
 * exp10 overflow     x > MAXL10       MAXNUM
 *
 * DEC arithmetic: MAXL10 = 38.230809449325611792.
 * IEEE arithmetic: MAXL10 = 308.2547155599167.
 *
 */



/*                                                      exp2.c
 *
 *      Base 2 exponential function
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, exp2();
 *
 * y = exp2( 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    -1022,+1024   30000       1.8e-16     5.4e-17
 *
 *
 * See exp.c for comments on error amplification.
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * exp underflow    x < -MAXL2        0.0
 * exp overflow     x > MAXL2         MAXNUM
 *
 * For DEC arithmetic, MAXL2 = 127.
 * For IEEE arithmetic, MAXL2 = 1024.
 */



/*                                                      expn.c
 *
 *              Exponential integral En
 *
 *
 *
 * SYNOPSIS:
 *
 * int n;
 * double x, y, expn();
 *
 * y = expn( n, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Evaluates the exponential integral
 *
 *                 inf.
 *                   -
 *                  | |   -xt
 *                  |    e
 *      E (x)  =    |    ----  dt.
 *       n          |      n
 *                | |     t
 *                 -
 *                  1
 *
 *
 * Both n and x must be nonnegative.
 *
 * The routine employs either a power series, a continued
 * fraction, or an asymptotic formula depending on the
```

```
 * relative values of n and x.
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    DEC       0, 30        5000       2.0e-16     4.6e-17
 *    IEEE      0, 30       10000       1.7e-15     3.6e-16
 *
 */



/*                                                      expx2.c
 *
 *      Exponential of squared argument
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, expx2();
 * int sign;
 *
 * y = expx2( 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      -26.6, 26.6    10^7       3.9e-16     8.9e-17
 *
 */



/*                                                      fabs.c
 *
 *              Absolute value
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y;
 *
 * y = fabs( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the absolute value of the argument.
 *
 */



/*                                                      fac.c
 *
 *      Factorial function
 *
 *
 *
 * SYNOPSIS:
 *
 * double y, fac();
 * int i;
 *
 * y = fac( i );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns factorial of i  =  1 * 2 * 3 * ... * i.
 * fac(0) = 1.0.
 *
 * Due to machine arithmetic bounds the largest value of
 * i accepted is 33 in DEC arithmetic or 170 in IEEE
 * arithmetic.  Greater values, or negative ones,
 * produce an error message and return MAXNUM.
 *
 *
 *
 * ACCURACY:
 *
 * For i < 34 the values are simply tabulated, and have
 * full machine accuracy.  If i > 55, fac(i) = gamma(i+1);
```

```
 * see gamma.c.
 *
 *                      Relative error:
 * arithmetic   domain      peak
 *    IEEE      0, 170    1.4e-15
 *    DEC       0, 33      1.4e-17
 *
 */



/*                                                      fdtr.c
 *
 *      F distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int df1, df2;
 * double x, y, fdtr();
 *
 * y = fdtr( df1, df2, x );
 *
 * DESCRIPTION:
 *
 * Returns the area from zero to x under the F density
 * function (also known as Snedcor's density or the
 * variance ratio density).  This is the density
 * of x = (u1/df1)/(u2/df2), where u1 and u2 are random
 * variables having Chi square distributions with df1
 * and df2 degrees of freedom, respectively.
 *
 * The incomplete beta integral is used, according to the
 * formula
 *
 *      P(x) = incbet( df1/2, df2/2, (df1*x/(df2 + df1*x) ).
 *
 *
 * The arguments a and b are greater than zero, and x is
 * nonnegative.
 *
 * ACCURACY:
 *
 * Tested at random points (a,b,x).
 *
 *               x     a,b                     Relative error:
 * arithmetic  domain  domain     # trials      peak         rms
 *    IEEE      0,1    0,100       100000      9.8e-15     1.7e-15
 *    IEEE      1,5    0,100       100000      6.5e-15     3.5e-16
 *    IEEE      0,1    1,10000     100000      2.2e-11     3.3e-12
 *    IEEE      1,5    1,10000     100000      1.1e-11     1.7e-13
 * See also incbet.c.
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * fdtr domain     a<0, b<0, x<0         0.0
 *
 */



/*                                                      fdtrc()
 *
 *      Complemented F distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int df1, df2;
 * double x, y, fdtrc();
 *
 * y = fdtrc( 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
 *
 *
 * The incomplete beta integral is used, according to the
 * formula
 *
 *      P(x) = incbet( df2/2, df1/2, (df2/(df2 + df1*x) ).
 *
 *
 * 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       100000      3.7e-14     5.9e-16
 *    IEEE      1,5    1,100       100000      8.0e-15     1.6e-15
 *    IEEE      0,1    1,10000     100000      1.8e-11     3.5e-13
 *    IEEE      1,5    1,10000     100000      2.0e-11     3.0e-12
 * See also incbet.c.
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * fdtrc domain    a<0, b<0, x<0         0.0
 *
 */



/*                                                      fdtri()
 *
 *      Inverse of complemented F distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int df1, df2;
 * double x, p, fdtri();
 *
 * x = fdtri( 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:
 *
 * 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       100000      8.3e-15     4.7e-16
 *    IEEE     1,10000     100000      2.1e-11     1.4e-13
 *  For p between 10^-6 and 10^-3:
 *    IEEE     1,100        50000      1.3e-12     8.4e-15
 *    IEEE     1,10000      50000      3.0e-12     4.8e-14
 * See also fdtrc.c.
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * fdtri domain   p <= 0 or p > 1       0.0
 *                     v < 1
 *
 */



/*                                                      fftr.c
 *
 *      FFT of Real Valued Sequence
 *
 *
 *
 * SYNOPSIS:
 *
 * double x[], sine[];
 * int m;
 *
 * fftr( x, m, sine );
 *
 *
 *
 * DESCRIPTION:
 *
 * Computes the (complex valued) discrete Fourier transform of
 * the real valued sequence x[].  The input sequence x[] contains
 * n = 2**m samples.  The program fills array sine[k] with
 * n/4 + 1 values of sin( 2 PI k / n ).
 *
 * Data format for complex valued output is real part followed
 * by imaginary part.  The output is developed in the input
 * array x[].
 *
```

```
 * The algorithm takes advantage of the fact that the FFT of an
 * n point real sequence can be obtained from an n/2 point
 * complex FFT.
 *
 * A radix 2 FFT algorithm is used.
 *
 * Execution time on an LSI-11/23 with floating point chip
 * is 1.0 sec for n = 256.
 *
 *
 *
 * REFERENCE:
 *
 * E. Oran Brigham, The Fast Fourier Transform;
 * Prentice-Hall, Inc., 1974
 *
 */



/*                                                      ceil()
 *                                                      floor()
 *                                                      frexp()
 *                                                      ldexp()
 *
 *      Floating point numeric utilities
 *
 *
 *
 * SYNOPSIS:
 *
 * double ceil(), floor(), frexp(), ldexp();
 * double x, y;
 * int expnt, n;
 *
 * y = floor(x);
 * y = ceil(x);
 * y = frexp( x, &expnt );
 * y = ldexp( x, n );
 *
 *
 *
 * DESCRIPTION:
 *
 * All four routines return a double precision floating point
 * result.
 *
 * floor() returns the largest integer less than or equal to x.
 * It truncates toward minus infinity.
 *
 * ceil() returns the smallest integer greater than or equal
 * to x.  It truncates toward plus infinity.
 *
 * frexp() 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.
 *
 * ldexp() multiplies x by 2**n.
 *
 * These functions are part of the standard C run time library
 * for many but not all C compilers.  The ones supplied are
 * written in C for either DEC or IEEE arithmetic.  They should
 * be used only if your compiler library does not already have
 * them.
 *
 * The IEEE versions assume that denormal numbers are implemented
 * in the arithmetic.  Some modifications will be required if
 * the arithmetic has abrupt rather than gradual underflow.
 */



/*                                                      fresnl.c
 *
 *      Fresnel integral
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, S, C;
 * void fresnl();
 *
 * fresnl( x, &S, &C );
 *
 *
 * DESCRIPTION:
 *
 * Evaluates the Fresnel integrals
 *
 *           x
 *           -
 *          | |
 * C(x) =   |   cos(pi/2 t**2) dt,
 *        | |
 *         -
 *          0
 *
 *           x
 *           -
```

```
 *           | |
 * S(x) =    |   sin(pi/2 t**2) dt.
 *         | |
 *          -
 *           0
 *
 *
 * The integrals are evaluated by a power series for x < 1.
 * For x >= 1 auxiliary functions f(x) and g(x) are employed
 * such that
 *
 * C(x) = 0.5 + f(x) sin( pi/2 x**2 ) - g(x) cos( pi/2 x**2 )
 * S(x) = 0.5 - f(x) cos( pi/2 x**2 ) - g(x) sin( pi/2 x**2 )
 *
 *
 *
 * ACCURACY:
 *
 *  Relative error.
 *
 * Arithmetic  function   domain     # trials      peak         rms
 *   IEEE       S(x)      0, 10       10000       2.0e-15     3.2e-16
 *   IEEE       C(x)      0, 10       10000       1.8e-15     3.3e-16
 *   DEC        S(x)      0, 10        6000       2.2e-16     3.9e-17
 *   DEC        C(x)      0, 10        5000       2.3e-16     3.9e-17
 */



/*                                                      gamma.c
 *
 *      Gamma function
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, gamma();
 * extern int sgngam;
 *
 * y = gamma( 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| <= 34 are reduced by recurrence and the function
 * approximated by a rational function of degree 6/7 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
 *    DEC      -34, 34      10000       1.3e-16     2.5e-17
 *    IEEE    -170,-33      20000       2.3e-15     3.3e-16
 *    IEEE     -33,  33     20000       9.4e-16     2.2e-16
 *    IEEE      33, 171.6   20000       2.3e-15     3.2e-16
 *
 * Error for arguments outside the test range will be larger
 * owing to error amplification by the exponential function.
 *
 */



/*                                                      lgam()
 *
 *      Natural logarithm of gamma function
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, lgam();
 * extern int sgngam;
 *
 * y = lgam( 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 13, 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 MAXLGM return MAXNUM and an error
 * message.  MAXLGM = 2.035093e36 for DEC
 * arithmetic or 2.556348e305 for IEEE arithmetic.
 *
 *
 *
 * ACCURACY:
 *
 *
 * arithmetic      domain        # trials     peak         rms
 *    DEC     0, 3                  7000     5.2e-17     1.3e-17
 *    DEC     2.718, 2.035e36       5000     3.9e-17     9.9e-18
 *    IEEE    0, 3                 28000     5.4e-16     1.1e-16
 *    IEEE    2.718, 2.556e305     40000     3.5e-16     8.3e-17
 * The error criterion was relative when the function magnitude
 * was greater than one but absolute when it was less than one.
 *
 * The following test used the relative error criterion, though
 * at certain points the relative error could be much higher than
 * indicated.
 *    IEEE    -200, -4             10000     4.8e-16     1.3e-16
 *
 */



/*                                                      gdtr.c
 *
 *      Gamma distribution function
 *
 *
 *
 * SYNOPSIS:
 *
 * double a, b, x, y, gdtr();
 *
 * y = gdtr( 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
 * gdtr domain         x < 0            0.0
 *
 */



/*                                                      gdtrc.c
 *
 *      Complemented gamma distribution function
 *
 *
 *
 * SYNOPSIS:
 *
 * double a, b, x, y, gdtrc();
 *
 * y = gdtrc( 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
 * gdtrc 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
*/



/*                                                      hyp2f1.c
 *
 *      Gauss hypergeometric function   F
 *                                     2 1
 *
 *
 * SYNOPSIS:
```

```
 *
 * double a, b, c, x, y, hyp2f1();
 *
 * y = hyp2f1( a, b, c, x );
 *
 *
 * DESCRIPTION:
 *
 *
 *  hyp2f1( a, b, c, x )  =   F ( a, b; c; x )
 *                           2 1
 *
 *           inf.
 *            -   a(a+1)...(a+k) b(b+1)...(b+k)   k+1
 *   =  1 +   >   -----------------------------  x   .
 *            -         c(c+1)...(c+k) (k+1)!
 *          k = 0
 *
 *  Cases addressed are
 *      Tests and escapes for negative integer a, b, or c
 *      Linear transformation if c - a or c - b negative integer
 *      Special case c = a or c = b
 *      Linear transformation for  x near +1
 *      Transformation for x < -0.5
 *      Psi function expansion if x > 0.5 and c - a - b integer
 *      Conditionally, a recurrence on c to make c-a-b > 0
 *
 * |x| > 1 is rejected.
 *
 * The parameters a, b, c are considered to be integer
 * valued if they are within 1.0e-14 of the nearest integer
 * (1.0e-13 for IEEE arithmetic).
 *
 * ACCURACY:
 *
 *
 *               Relative error (-1 < x < 1):
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      -1,7        230000      1.2e-11     5.2e-14
 *
 * Several special cases also tested with a, b, c in
 * the range -7 to 7.
 *
 * ERROR MESSAGES:
 *
 * A "partial loss of precision" message is printed if
 * the internally estimated relative error exceeds 1^-12.
 * A "singularity" message is printed on overflow or
 * in cases not addressed (such as x < -1).
 */



/*                                                      hyperg.c
 *
 *      Confluent hypergeometric function
 *
 *
 *
 * SYNOPSIS:
 *
 * double a, b, x, y, hyperg();
 *
 * y = hyperg( 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
 *    DEC       0,30         2000       1.2e-15     1.3e-16
 qtst1:
 21800   max =  1.4200E-14   rms =  1.0841E-15  ave = -5.3640E-17
 ltstd:
 25500   max = 1.2759e-14   rms = 3.7155e-16  ave = 1.5384e-18
```

```
 *    IEEE      0,30        30000       1.8e-14     1.1e-15
 *
 * 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.
 *
 */



/*                                                      i0.c
 *
 *      Modified Bessel function of order zero
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, i0();
 *
 * y = i0( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns modified Bessel function of order zero of the
 * argument.
 *
 * The function is defined as i0(x) = j0( ix ).
 *
 * The range is partitioned into the two intervals [0,8] and
 * (8, infinity).  Chebyshev polynomial expansions are employed
 * in each interval.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    DEC       0,30         6000       8.2e-17     1.9e-17
 *    IEEE      0,30        30000       5.8e-16     1.4e-16
 *
 */



/*                                                      i0e.c
 *
 *      Modified Bessel function of order zero,
 *      exponentially scaled
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, i0e();
 *
 * y = i0e( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns exponentially scaled modified Bessel function
 * of order zero of the argument.
 *
 * The function is defined as i0e(x) = exp(-|x|) j0( ix ).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,30        30000       5.4e-16     1.2e-16
 * See i0().
 *
 */



/*                                                      i1.c
 *
 *      Modified Bessel function of order one
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, i1();
 *
 * y = i1( x );
 *
 *
 *
```

```
 * DESCRIPTION:
 *
 * Returns modified Bessel function of order one of the
 * argument.
 *
 * The function is defined as i1(x) = -i j1( ix ).
 *
 * The range is partitioned into the two intervals [0,8] and
 * (8, infinity).  Chebyshev polynomial expansions are employed
 * in each interval.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    DEC       0, 30        3400       1.2e-16     2.3e-17
 *    IEEE      0, 30       30000       1.9e-15     2.1e-16
 *
 *
 */



/*                                                      i1e.c
 *
 *      Modified Bessel function of order one,
 *      exponentially scaled
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, i1e();
 *
 * y = i1e( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns exponentially scaled modified Bessel function
 * of order one of the argument.
 *
 * The function is defined as i1(x) = -i exp(-|x|) j1( ix ).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0, 30       30000       2.0e-15     2.0e-16
 * See i1().
 *
 */



/*                                                      igam.c
 *
 *      Incomplete gamma integral
 *
 *
 *
 * SYNOPSIS:
 *
 * double a, x, y, igam();
 *
 * y = igam( a, x );
 *
 * DESCRIPTION:
 *
 * The function is defined by
 *
 *                           x
 *                            -
 *                   1       | |  -t  a-1
 *  igam(a,x)  =   -----     |   e   t   dt.
 *                  -       | |
 *                 | (a)    -
 *                           0
 *
 *
 * In this implementation both arguments must be positive.
 * The integral is evaluated by either a power series or
 * continued fraction expansion, depending on the relative
 * values of a and x.
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,30       200000       3.6e-14     2.9e-15
 *    IEEE      0,100      300000       9.9e-14     1.5e-14
 */
```

```
/*                                                      igamc()
 *
 *      Complemented incomplete gamma integral
 *
 *
 *
 * SYNOPSIS:
 *
 * double a, x, y, igamc();
 *
 * y = igamc( 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:
 *
 * Tested at random a, x.
 *                a         x                      Relative error:
 * arithmetic   domain   domain     # trials      peak         rms
 *    IEEE     0.5,100   0,100      200000       1.9e-14     1.7e-15
 *    IEEE     0.01,0.5  0,100      200000       1.4e-13     1.6e-15
 */


/*                                                      igami()
 *
 *      Inverse of complemented imcomplete gamma integral
 *
 *
 *
 * SYNOPSIS:
 *
 * double a, x, p, igami();
 *
 * x = igami( a, p );
 *
 * DESCRIPTION:
 *
 * Given p, the function finds x such that
 *
 *  igamc( a, x ) = p.
 *
 * It is valid in the right-hand tail of the distribution, p < 0.5.
 * Starting with the approximate value
 *
 *         3
 *  x = a t
 *
 *  where
 *
 *  t = 1 - d - ndtri(p) sqrt(d)
 *
 * and
 *
 *  d = 1/9a,
 *
 * the routine performs up to 10 Newton iterations to find the
 * root of igamc(a,x) - p = 0.
 *
 * ACCURACY:
 *
 * Tested at random a, p in the intervals indicated.
 *
 *                a        p                      Relative error:
 * arithmetic   domain   domain     # trials      peak         rms
 *    IEEE     0.5,100   0,0.5       100000       1.0e-14     1.7e-15
 *    IEEE     0.01,0.5  0,0.5       100000       9.0e-14     3.4e-15
 *    IEEE    0.5,10000  0,0.5        20000       2.3e-13     3.8e-14
 */


/*                                                      incbet.c
 *
 *      Incomplete beta integral
 *
 *
 * SYNOPSIS:
 *
 * double a, b, x, y, incbet();
```

```
 *
 * y = incbet( 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 uniformly distributed random points (a,b,x) with a and b
 * in "domain" and x between 0 and 1.
 *                                        Relative error
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,5         10000       6.9e-15     4.5e-16
 *    IEEE      0,85       250000       2.2e-13     1.7e-14
 *    IEEE      0,1000      30000       5.3e-12     6.3e-13
 *    IEEE      0,10000    250000       9.3e-11     7.1e-12
 *    IEEE      0,100000    10000       8.7e-10     4.8e-11
 * Outputs smaller than the IEEE gradual underflow threshold
 * were excluded from these statistics.
 *
 * ERROR MESSAGES:
 *   message         condition      value returned
 * incbet domain      x<0, x>1          0.0
 * incbet underflow                     0.0
 */



/*                                                      incbi()
 *
 *      Inverse of imcomplete beta integral
 *
 *
 *
 * SYNOPSIS:
 *
 * double a, b, x, y, incbi();
 *
 * x = incbi( a, b, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Given y, the function finds x such that
 *
 *  incbet( a, b, x ) = y .
 *
 * The routine performs interval halving or 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   50000    5.8e-12   1.3e-13
 *    IEEE      0,1   .25,100    100000    1.8e-13   3.9e-15
 *    IEEE      0,1     0,5       50000    1.1e-12   5.5e-15
 *    VAX       0,1    .5,100     25000    3.5e-14   1.1e-15
 * With a and b constrained to half-integer or integer values:
 *    IEEE      0,1    .5,10000   50000    5.8e-12   1.1e-13
 *    IEEE      0,1    .5,100    100000    1.7e-14   7.9e-16
 * With a = .5, b constrained to half-integer or integer values:
 *    IEEE      0,1    .5,10000   10000    8.3e-11   1.0e-11
 */



/*                                                      isnan()
 *                                                      signbit()
 *                                                      isfinite()
 *
 *      Floating point numeric utilities
 *
 *
 *
 * SYNOPSIS:
```

```
 *
 * double ceil(), floor(), frexp(), ldexp();
 * int signbit(), isnan(), isfinite();
 * double x, y;
 * int expnt, n;
 *
 * y = floor(x);
 * y = ceil(x);
 * y = frexp( x, &expnt );
 * y = ldexp( x, n );
 * n = signbit(x);
 * n = isnan(x);
 * n = isfinite(x);
 *
 *
 *
 * DESCRIPTION:
 *
 * All four routines return a double precision floating point
 * result.
 *
 * floor() returns the largest integer less than or equal to x.
 * It truncates toward minus infinity.
 *
 * ceil() returns the smallest integer greater than or equal
 * to x.  It truncates toward plus infinity.
 *
 * frexp() 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.
 *
 * ldexp() multiplies x by 2**n.
 *
 * signbit(x) returns 1 if the sign bit of x is 1, else 0.
 *
 * These functions are part of the standard C run time library
 * for many but not all C compilers.  The ones supplied are
 * written in C for either DEC or IEEE arithmetic.  They should
 * be used only if your compiler library does not already have
 * them.
 *
 * The IEEE versions assume that denormal numbers are implemented
 * in the arithmetic.  Some modifications will be required if
 * the arithmetic has abrupt rather than gradual underflow.
 */



/*                                                      iv.c
 *
 *      Modified Bessel function of noninteger order
 *
 *
 *
 * SYNOPSIS:
 *
 * double v, x, y, iv();
 *
 * y = iv( v, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns modified Bessel function of order v of the
 * argument.  If x is negative, v must be integer valued.
 *
 * The function is defined as Iv(x) = Jv( ix ).  It is
 * here computed in terms of the confluent hypergeometric
 * function, according to the formula
 *
 *              v  -x
 * Iv(x) = (x/2)  e   hyperg( v+0.5, 2v+1, 2x ) / gamma(v+1)
 *
 * If v is a negative integer, then v is replaced by -v.
 *
 *
 * ACCURACY:
 *
 * Tested at random points (v, x), with v between 0 and
 * 30, x between 0 and 28.
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    DEC       0,30          2000      3.1e-15     5.4e-16
 *    IEEE      0,30         10000      1.7e-14     2.7e-15
 *
 * Accuracy is diminished if v is near a negative integer.
 *
 * See also hyperg.c.
 *
 */



/*                                                      j0.c
 *
 *      Bessel function of order zero
 *
 *
 *
```

```
 * SYNOPSIS:
 *
 * double x, y, j0();
 *
 * y = j0( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of order zero of the argument.
 *
 * The domain is divided into the intervals [0, 5] and
 * (5, infinity). In the first interval the following rational
 * approximation is used:
 *
 *
 *        2         2
 * (w - r  ) (w - r  ) P (w) / Q (w)
 *       1         2    3       8
 *
 *            2
 * where w = x  and the two r's are zeros of the function.
 *
 * In the second interval, the Hankel asymptotic expansion
 * is employed with two rational functions of degree 6/6
 * and 7/7.
 *
 *
 *
 * ACCURACY:
 *
 *                      Absolute error:
 * arithmetic   domain     # trials      peak         rms
 *    DEC       0, 30       10000       4.4e-17     6.3e-18
 *    IEEE      0, 30       60000       4.2e-16     1.1e-16
 *
 */



/*                                                      y0.c
 *
 *      Bessel function of the second kind, order zero
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, y0();
 *
 * y = y0( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of the second kind, of order
 * zero, of the argument.
 *
 * The domain is divided into the intervals [0, 5] and
 * (5, infinity). In the first interval a rational approximation
 * R(x) is employed to compute
 *   y0(x)  = R(x)  +   2 * log(x) * j0(x) / PI.
 * Thus a call to j0() is required.
 *
 * In the second interval, the Hankel asymptotic expansion
 * is employed with two rational functions of degree 6/6
 * and 7/7.
 *
 *
 *
 * ACCURACY:
 *
 *  Absolute error, when y0(x) < 1; else relative error:
 *
 * arithmetic   domain     # trials      peak         rms
 *    DEC       0, 30        9400       7.0e-17     7.9e-18
 *    IEEE      0, 30       30000       1.3e-15     1.6e-16
 *
 */



/*                                                      j1.c
 *
 *      Bessel function of order one
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, j1();
 *
 * y = j1( x );
 *
 *
 *
 * DESCRIPTION:
 *
```

```
 * Returns Bessel function of order one of the argument.
 *
 * The domain is divided into the intervals [0, 8] and
 * (8, infinity). In the first interval a 24 term Chebyshev
 * expansion is used. In the second, the asymptotic
 * trigonometric representation is employed using two
 * rational functions of degree 5/5.
 *
 *
 *
 * ACCURACY:
 *
 *                      Absolute error:
 * arithmetic   domain      # trials      peak         rms
 *    DEC       0, 30       10000       4.0e-17     1.1e-17
 *    IEEE      0, 30       30000       2.6e-16     1.1e-16
 *
 *
 */



/*                                                      y1.c
 *
 *      Bessel function of second kind of order one
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, y1();
 *
 * y = y1( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of the second kind of order one
 * of the argument.
 *
 * The domain is divided into the intervals [0, 8] and
 * (8, infinity). In the first interval a 25 term Chebyshev
 * expansion is used, and a call to j1() is required.
 * In the second, the asymptotic trigonometric representation
 * is employed using two rational functions of degree 5/5.
 *
 *
 *
 * ACCURACY:
 *
 *                      Absolute error:
 * arithmetic   domain      # trials      peak         rms
 *    DEC       0, 30       10000       8.6e-17     1.3e-17
 *    IEEE      0, 30       30000       1.0e-15     1.3e-16
 *
 * (error criterion relative when |y1| > 1).
 *
 */



/*                                                      jn.c
 *
 *      Bessel function of integer order
 *
 *
 *
 * SYNOPSIS:
 *
 * int n;
 * double x, y, jn();
 *
 * y = jn( n, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of order n, where n is a
 * (possibly negative) integer.
 *
 * The ratio of jn(x) to j0(x) is computed by backward
 * recurrence.  First the ratio jn/jn-1 is found by a
 * continued fraction expansion.  Then the recurrence
 * relating successive orders is applied until j0 or j1 is
 * reached.
 *
 * If n = 0 or 1 the routine for j0 or j1 is called
 * directly.
 *
 *
 *
 * ACCURACY:
 *
 *                      Absolute error:
 * arithmetic   range      # trials      peak         rms
 *    DEC       0, 30        5500       6.9e-17     9.3e-18
 *    IEEE      0, 30        5000       4.4e-16     7.9e-17
 *
```

```
 *
 * Not suitable for large n or x. Use jv() instead.
 *
 */



/*                                                      jv.c
 *
 *      Bessel function of noninteger order
 *
 *
 *
 * SYNOPSIS:
 *
 * double v, x, y, jv();
 *
 * y = jv( v, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of order v of the argument,
 * where v is real.  Negative x is allowed if v is an integer.
 *
 * Several expansions are included: the ascending power
 * series, the Hankel expansion, and two transitional
 * expansions for large v.  If v is not too large, it
 * is reduced by recurrence to a region of best accuracy.
 * The transitional expansions give 12D accuracy for v > 500.
 *
 *
 *
 * ACCURACY:
 * Results for integer v are indicated by *, where x and v
 * both vary from -125 to +125.  Otherwise,
 * x ranges from 0 to 125, v ranges as indicated by "domain."
 * Error criterion is absolute, except relative when |jv()| > 1.
 *
 * arithmetic  v domain  x domain    # trials      peak       rms
 *    IEEE      0,125     0,125      100000      4.6e-15    2.2e-16
 *    IEEE   -125,0       0,125       40000      5.4e-11    3.7e-13
 *    IEEE      0,500     0,500       20000      4.4e-15    4.0e-16
 * Integer v:
 *    IEEE   -125,125   -125,125      50000      3.5e-15*   1.9e-16*
 *
 */



/*                                                      k0.c
 *
 *      Modified Bessel function, third kind, order zero
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, k0();
 *
 * y = k0( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns modified Bessel function of the third kind
 * of order zero of the argument.
 *
 * The range is partitioned into the two intervals [0,8] and
 * (8, infinity).  Chebyshev polynomial expansions are employed
 * in each interval.
 *
 *
 *
 * ACCURACY:
 *
 * Tested at 2000 random points between 0 and 8.  Peak absolute
 * error (relative when K0 > 1) was 1.46e-14; rms, 4.26e-15.
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    DEC       0, 30        3100       1.3e-16     2.1e-17
 *    IEEE      0, 30       30000       1.2e-15     1.6e-16
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 *  K0 domain          x <= 0          MAXNUM
 *
 */



/*                                                      k0e()
 *
 *      Modified Bessel function, third kind, order zero,
 *      exponentially scaled
 *
 *
```

```
 *
 * SYNOPSIS:
 *
 * double x, y, k0e();
 *
 * y = k0e( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns exponentially scaled modified Bessel function
 * of the third kind of order zero of the argument.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0, 30       30000       1.4e-15     1.4e-16
 * See k0().
 *
 */



/*                                                      k1.c
 *
 *      Modified Bessel function, third kind, order one
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, k1();
 *
 * y = k1( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Computes the modified Bessel function of the third kind
 * of order one of the argument.
 *
 * The range is partitioned into the two intervals [0,2] and
 * (2, infinity).  Chebyshev polynomial expansions are employed
 * in each interval.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    DEC       0, 30        3300       8.9e-17     2.2e-17
 *    IEEE      0, 30       30000       1.2e-15     1.6e-16
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * k1 domain          x <= 0          MAXNUM
 *
 */



/*                                                      k1e.c
 *
 *      Modified Bessel function, third kind, order one,
 *      exponentially scaled
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, k1e();
 *
 * y = k1e( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns exponentially scaled modified Bessel function
 * of the third kind of order one of the argument:
 *
 *      k1e(x) = exp(x) * k1(x).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0, 30       30000       7.8e-16     1.2e-16
 * See k1().
 *
 */
```

```
/*                                                      kn.c
 *
 *      Modified Bessel function, third kind, integer order
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, kn();
 * int n;
 *
 * y = kn( n, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns modified Bessel function of the third kind
 * of order n of the argument.
 *
 * The range is partitioned into the two intervals [0,9.55] and
 * (9.55, infinity).  An ascending power series is used in the
 * low range, and an asymptotic expansion in the high range.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    DEC       0,30         3000       1.3e-9      5.8e-11
 *    IEEE      0,30        90000       1.8e-8      3.0e-10
 *
 *  Error is high only near the crossover point x = 9.55
 * between the two expansions used.
 */



/* Re Kolmogorov statistics, here is Birnbaum and Tingey's formula for the
   distribution of D+, the maximum of all positive deviations between a
   theoretical distribution function P(x) and an empirical one Sn(x)
   from n samples.

     +
    D  =         sup     [P(x) - S (x)]
     n     -inf < x < inf         n


                  [n(1-e)]
        +            -                    v-1              n-v
    Pr{D   > e} =    >    C    e (e + v/n)    (1 - e - v/n)
        n            -   n v
                    v=0

    [n(1-e)] is the largest integer not exceeding n(1-e).
    nCv is the number of combinations of n things taken v at a time.  */



/*                                                      lmdif.c
*
*     The purpose of lmdif is to minimize the sum of the squares of
*     M nonlinear functions in N variables by a modification of
*     the Levenberg-Marquardt algorithm. The user must provide a
*     subroutine that calculates the functions.  The Jacobian is
*     then calculated numerically by a forward-difference approximation.
*
*     Refer to the source code for information on the use of the routine.
*
*     This is a C language translation of the Fortran version of
*     the corresponding routine from Argonne National Laboratories
*     MINPACK subroutine suite.
*
*/



/*              Levnsn.c                */
/* Levinson-Durbin LPC
 * linear predictive coding
 *
 * | R0 R1 R2 ... RN-1 |   | A1 |       | -R1 |
 * | R1 R0 R1 ... RN-2 |   | A2 |       | -R2 |
 * | R2 R1 R0 ... RN-3 |   | A3 |   =   | -R3 |
 * |         ...       |   | ...|       | ... |
 * | RN-1 RN-2... R0   |   | AN |       | -RN |
 *
 * Ref: John Makhoul, "Linear Prediction, A Tutorial Review"
 * Proc. IEEE Vol. 63, PP 561-580 April, 1975.
 *
 * R is the input autocorrelation function.  R0 is the zero lag
 * term.  A is the output array of predictor coefficients.  Note
 * that a filter impulse response has a coefficient of 1.0 preceding
 * A1.  E is an array of mean square error for each prediction order
 * 1 to N.  REFL is an output array of the reflection coefficients.
 */
```

```
/*                                                        log.c
 *
 *      Natural logarithm
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, log();
 *
 * y = log( 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      1.44e-16    5.06e-17
 *    IEEE      +-MAXNUM    30000       1.20e-16    4.78e-17
 *    DEC       0, 10       170000      1.8e-17     6.3e-18
 *
 * In the tests over the interval [+-MAXNUM], the logarithms
 * of the random arguments were uniformly distributed over
 * [0, MAXLOG].
 *
 * ERROR MESSAGES:
 *
 * log singularity:  x = 0; returns -INFINITY
 * log domain:       x < 0; returns NAN
 */



/*                                                        log10.c
 *
 *      Common logarithm
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, log10();
 *
 * y = log10( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns logarithm to the base 10 of x.
 *
 * The argument is separated into its exponent and fractional
 * parts.  The logarithm of the fraction is approximated by
 *
 *     log(1+x) = x - 0.5 x**2 + x**3 P(x)/Q(x).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0.5, 2.0     30000      1.5e-16     5.0e-17
 *    IEEE      0, MAXNUM    30000      1.4e-16     4.8e-17
 *    DEC       1, MAXNUM    50000      2.5e-17     6.0e-18
 *
 * In the tests over the interval [1, MAXNUM], the logarithms
 * of the random arguments were uniformly distributed over
 * [0, MAXLOG].
 *
 * ERROR MESSAGES:
 *
 * log10 singularity:  x = 0; returns -INFINITY
 * log10 domain:       x < 0; returns NAN
 */



/*                                                        log2.c
 *
 *      Base 2 logarithm
```

```
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, log2();
 *
 * y = log2( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the base 2 logarithm of x.
 *
 * The argument is separated into its exponent and fractional
 * parts.  If the exponent is between -1 and +1, the base e
 * logarithm of the fraction is approximated by
 *
 *     log(1+x) = x - 0.5 x**2 + x**3 P(x)/Q(x).
 *
 * Otherwise, setting  z = 2(x-1)/x+1),
 *
 *     log(x) = z + z**3 P(z)/Q(z).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0.5, 2.0    30000       2.0e-16     5.5e-17
 *    IEEE      exp(+-700)  40000       1.3e-16     4.6e-17
 *
 * In the tests over the interval [exp(+-700)], the logarithms
 * of the random arguments were uniformly distributed.
 *
 * ERROR MESSAGES:
 *
 * log2 singularity:  x = 0; returns -INFINITY
 * log2 domain:       x < 0; returns NAN
 */


/*                                                      lrand.c
 *
 *      Pseudorandom number generator
 *
 *
 *
 * SYNOPSIS:
 *
 * long y, drand();
 *
 * drand( &y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Yields a long integer random number.
 *
 * The three-generator congruential algorithm by Brian
 * Wichmann and David Hill (BYTE magazine, March, 1987,
 * pp 127-8) is used. The period, given by them, is
 * 6953607871644.
 *
 *
 */


/*                                                      lsqrt.c
 *
 *      Integer square root
 *
 *
 *
 * SYNOPSIS:
 *
 * long x, y;
 * long lsqrt();
 *
 * y = lsqrt( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns a long integer square root of the long integer
 * argument.  The computation is by binary long division.
 *
 * The largest possible result is lsqrt(2,147,483,647)
 * = 46341.
 *
 * If x < 0, the square root of |x| is returned, and an
 * error message is printed.
 *
 *
```

```
 * ACCURACY:
 *
 * An extra, roundoff, bit is computed; hence the result
 * is the nearest integer to the actual square root.
 * NOTE: only DEC arithmetic is currently supported.
 *
 */



/*                                                      minv.c
 *
 *      Matrix inversion
 *
 *
 *
 * SYNOPSIS:
 *
 * int n, errcod;
 * double A[n*n], X[n*n];
 * double B[n];
 * int IPS[n];
 * int minv();
 *
 * errcod = minv( A, X, n, B, IPS );
 *
 *
 *
 * DESCRIPTION:
 *
 * Finds the inverse of the n by n matrix A.  The result goes
 * to X.   B and IPS are scratch pad arrays of length n.
 * The contents of matrix A are destroyed.
 *
 * The routine returns nonzero on error; error messages are printed
 * by subroutine simq().
 *
 */



/*                                                      mtransp.c
 *
 *      Matrix transpose
 *
 *
 *
 * SYNOPSIS:
 *
 * int n;
 * double A[n*n], T[n*n];
 *
 * mtransp( n, A, T );
 *
 *
 *
 * DESCRIPTION:
 *
 *
 * T[r][c] = A[c][r]
 *
 *
 * Transposes the n by n square matrix A and puts the result in T.
 * The output, T, may occupy the same storage as A.
 *
 *
 *
 */



/*                                                      nbdtr.c
 *
 *      Negative binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k, n;
 * double p, y, nbdtr();
 *
 * y = nbdtr( 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 (a,b,p), with p between 0 and 1.
 *
 *               a,b                     Relative error:
 * arithmetic  domain     # trials      peak         rms
 *    IEEE     0,100       100000      1.7e-13     8.8e-15
 * See also incbet.c.
 *
 */



/*                                                      nbdtr.c
 *
 *      Complemented negative binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k, n;
 * double p, y, nbdtrc();
 *
 * y = nbdtrc( 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:
 *
 * Tested at random points (a,b,p), with p between 0 and 1.
 *
 *               a,b                     Relative error:
 * arithmetic  domain     # trials      peak         rms
 *    IEEE     0,100       100000      1.7e-13     8.8e-15
 * See also incbet.c.
 */



/*                                                      nbdtr.c
 *
 *      Functional inverse of negative binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k, n;
 * double p, y, nbdtri();
 *
 * p = nbdtri( 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       100000      1.5e-14     8.5e-16
 * See also incbi.c.
 */



/*                                                      ndtr.c
 *
 *      Normal distribution function
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, ndtr();
 *
```

```
 * y = ndtr( 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       1.3e-15     2.2e-16
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition         value returned
 * erfc underflow    x > 37.519379347       0.0
 *
 */



/*                                                     ndtr.c
 *
 *      Error function
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, erf();
 *
 * y = erf( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * The integral is
 *
 *                           x
 *                            -
 *                 2         | |          2
 *   erf(x)  =  --------     |    exp( - t  ) dt.
 *              sqrt(pi)   | |
 *                          -
 *                           0
 *
 * The magnitude of x is limited to 9.231948545 for DEC
 * arithmetic; 1 or -1 is returned outside this range.
 *
 * For 0 <= |x| < 1, erf(x) = x * P4(x**2)/Q5(x**2); otherwise
 * erf(x) = 1 - erfc(x).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    DEC       0,1         14000       4.7e-17     1.5e-17
 *    IEEE      0,1         30000       3.7e-16     1.0e-16
 *
 */



/*                                                     ndtr.c
 *
 *      Complementary error function
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, erfc();
 *
 * y = erfc( 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 expx2.c is used to suppress error amplification
 * in computing exp(-x^2).
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,26.6417   30000       1.3e-15     2.2e-16
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition              value returned
 * erfc underflow    x > 9.231948545 (DEC)       0.0
 *
 *
 */



/*							ndtri.c
 *
 *	Inverse of Normal distribution function
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, ndtri();
 *
 * x = ndtri( y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the argument, x, for which the area under the
 * Gaussian probability density function (integrated from
 * minus infinity to x) is equal to y.
 *
 *
 * For small arguments 0 < y < exp(-2), the program computes
 * z = sqrt( -2.0 * log(y) );  then the approximation is
 * x = z - log(z)/z  - (1/z) P(1/z) / Q(1/z).
 * There are two rational functions P/Q, one for 0 < y < exp(-32)
 * and the other for y up to exp(-2).  For larger arguments,
 * w = y - 0.5, and  x/sqrt(2pi) = w + w**3 R(w**2)/S(w**2)).
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain        # trials      peak         rms
 *    DEC      0.125, 1         5500       9.5e-17     2.1e-17
 *    DEC      6e-39, 0.135     3500       5.7e-17     1.3e-17
 *    IEEE     0.125, 1        20000       7.2e-16     1.3e-16
 *    IEEE     3e-308, 0.135   50000       4.6e-16     9.8e-17
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition    value returned
 * ndtri domain       x <= 0        -MAXNUM
 * ndtri domain       x >= 1         MAXNUM
 *
 */



/*							pdtr.c
 *
 *	Poisson distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k;
 * double m, y, pdtr();
 *
 * y = pdtr( 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().
 *
 */



/*                                                      pdtrc()
 *
 *      Complemented poisson distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k;
 * double m, y, pdtrc();
 *
 * y = pdtrc( 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.
 *
 */



/*                                                      pdtri()
 *
 *      Inverse Poisson distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int k;
 * double m, y, pdtr();
 *
 * m = pdtri( 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
 *
 */



/*                                                      planck.c
 *
 *      Integral of Planck's black body radiation formula
 *
 *
 *
 * SYNOPSIS:
 *
 * double lambda, T, y, plancki();
 *
 * y = plancki( lambda, T );
 *
 *
 *
 * DESCRIPTION:
 *
 *  Evaluates the definite integral, from wavelength 0 to lambda,
 *  of Planck's radiation formula
 *                      -5
 *            c1  lambda
 *     E =  ------------------
 *            c2/(lambda T)
 *           e             - 1
 *
 * Physical constants c1 and c2 (see below) are built in
 * to the function program.  They are scaled to provide a result
 * in watts per square meter.  Argument T represents temperature in degrees
 * Kelvin; lambda is wavelength in meters.
 *
 * The integral is expressed in closed form, in terms of polylogarithms
 * (see polylog.c).
 *
 * The total area under the curve is
 *      (-1/8) (42 zeta(4) - 12 pi^2 zeta(2) + pi^4 ) c1 (T/c2)^4
 *       = (pi^4 / 15)  c1 (T/c2)^4
 *       =  sigma T^4
 *
 *
 * CONSTANTS:
 *
 * First radiation constant c1 = 2 pi h c^2 = 3.741 771 53 (17) e-16 W m2
 * Second radiation constant c2 = h c / k  = 0.014 387 770 (13) m K
 * Stefan-Boltzmann constant sigma = 5.670 373 (21) e-8 W m^-2 K^-4
 * Wien wavelength displacement law constant  wien = 2.8977721 (26) e-3 m K
 * These are NIST values as of 2010.
 *
 *
 * ACCURACY:
 *
 * The left tail of the function experiences some relative error
 * amplification in computing the dominant term exp(-c2/(lambda T)).
 * For the right-hand tail see planckc, below.
 *
 *                      Relative error.
 *   The domain refers to lambda T / c2.
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0.1, 10      50000      7.1e-15     5.4e-16
 *
 */



/*                                                      polevl.c
 *                                                      p1evl.c
 *
 *      Evaluate polynomial
 *
 *
 *
 * SYNOPSIS:
 *
 * int N;
 * double x, y, coef[N+1], polevl[];
 *
 * y = polevl( x, coef, N );
 *
 *
 *
 * DESCRIPTION:
 *
 * Evaluates polynomial of degree N:
 *
 *                     2          N
 * y  =  C  + C x + C x  +...+ C x
```

```
 *        0    1     2          N
 *
 * Coefficients are stored in reverse order:
 *
 * coef[0] = C  , ..., coef[N] = C  .
 *            N                   0
 *
 *  The function p1evl() assumes that coef[N] = 1.0 and is
 * omitted from the array.  Its calling arguments are
 * otherwise the same as polevl().
 *
 *
 * SPEED:
 *
 * In the interest of speed, there are no checks for out
 * of bounds arithmetic.  This routine is used by most of
 * the functions in the library.  Depending on available
 * equipment features, the user may wish to rewrite the
 * program in microcode or assembly language.
 *
 */



/*                                                      polmisc.c
 * Square root, sine, cosine, and arctangent of polynomial.
 * See polyn.c for data structures and discussion.
 */



/*                                                      polrt.c
 *
 *      Find roots of a polynomial
 *
 *
 *
 * SYNOPSIS:
 *
 * typedef struct
 *      {
 *      double r;
 *      double i;
 *      }cmplx;
 *
 * double xcof[], cof[];
 * int m;
 * cmplx root[];
 *
 * polrt( xcof, cof, m, root )
 *
 *
 *
 * DESCRIPTION:
 *
 * Iterative determination of the roots of a polynomial of
 * degree m whose coefficient vector is xcof[].  The
 * coefficients are arranged in ascending order; i.e., the
 * coefficient of x**m is xcof[m].
 *
 * The array cof[] is working storage the same size as xcof[].
 * root[] is the output array containing the complex roots.
 *
 *
 * ACCURACY:
 *
 * Termination depends on evaluation of the polynomial at
 * the trial values of the roots.  The values of multiple roots
 * or of roots that are nearly equal may have poor relative
 * accuracy after the first root in the neighborhood has been
 * found.
 *
 */



/*                                                      polylog.c
 *
 *      Polylogarithms
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, polylog();
 * int n;
 *
 * y = polylog( n, x );
 *
 *
 * The polylogarithm of order n is defined by the series
 *
 *
 *              inf   k
 *               -   x
 *  Li (x)  =    >   ---  .
 *    n          -     n
 *              k=1   k
 *
 *
```

```
 *  For x = 1,
 *
 *               inf
 *                -    1
 *   Li (1)  =    >   ---   =  Riemann zeta function (n)  .
 *     n          -     n
 *               k=1   k
 *
 *
 *  When n = 2, the function is the dilogarithm, related to Spence's integral:
 *
 *                 x                      1-x
 *                 -                        -
 *                | |  -ln(1-t)           | |  ln t
 *   Li (x)  =    |    -------- dt    =    |    ------ dt    =   spence(1-x) .
 *     2        | |       t             | |    1 - t
 *               -                        -
 *                0                        1
 *
 *
 *  See also the program cpolylog.c for the complex polylogarithm,
 *  whose definition is extended to x > 1.
 *
 *  References:
 *
 *  Lewin, L., _Polylogarithms and Associated Functions_,
 *  North Holland, 1981.
 *
 *  Lewin, L., ed., _Structural Properties of Polylogarithms_,
 *  American Mathematical Society, 1991.
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain   n   # trials      peak         rms
 *    IEEE      0, 1     2     50000      6.2e-16     8.0e-17
 *    IEEE      0, 1     3    100000      2.5e-16     6.6e-17
 *    IEEE      0, 1     4     30000      1.7e-16     4.9e-17
 *    IEEE      0, 1     5     30000      5.1e-16     7.8e-17
 *
 */



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

```
 * Notes:
 * poldiv() is an integer routine; poleva() is double.
 * Any of the arguments a, b, c may refer to the same array.
 *
 */



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



/*                                                      pow.c
 *
 *      Power function
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, z, pow();
 *
 * z = pow( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Computes x raised to the yth power.  Analytically,
 *
 *      x**y  =  exp( y log(x) ).
 *
 * Following Cody and Waite, this program uses a lookup table
 * of 2**-i/16 and pseudo extended precision arithmetic to
 * obtain an extra three bits of accuracy in both the logarithm
 * and the exponential.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
```

```
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     -26,26       30000      4.2e-16      7.7e-17
 *    DEC      -26,26       60000      4.8e-17      9.1e-18
 * 1/26 < x < 26, with log(x) uniformly distributed.
 * -26 < y < 26, y uniformly distributed.
 *    IEEE     0,8700       30000      1.5e-14      2.1e-15
 * 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
 *
 */



/*                                                      powi.c
 *
 *      Real raised to integer power
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, powi();
 * int n;
 *
 * y = powi( 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
 *    DEC       .04,26     -26,26    100000       2.7e-16     4.3e-17
 *    IEEE      .04,26     -26,26     50000       2.0e-15     3.8e-16
 *    IEEE        1,2    -1022,1023   50000       8.6e-14     1.6e-14
 *
 * Returns MAXNUM on overflow, zero on underflow.
 *
 */



/*                                                      psi.c
 *
 *      Psi (digamma) function
 *
 *
 * SYNOPSIS:
 *
 * double x, y, psi();
 *
 * y = psi( x );
 *
 *
 * DESCRIPTION:
 *
 *              d      -
 *   psi(x)  =  -- ln | (x)
 *              dx
 *
 * is the logarithmic derivative of the gamma function.
 * For integer x,
 *                   n-1
 *                    -
 * psi(n) = -EUL  +   >  1/k.
 *                    -
 *                   k=1
 *
 * This formula is used for 0 < n <= 10.  If x is negative, it
 * is transformed to a positive argument by the reflection
 * formula  psi(1-x) = psi(x) + pi cot(pi x).
 * For general positive x, the argument is made greater than 10
 * using the recurrence  psi(x+1) = psi(x) + 1/x.
 * Then the following asymptotic expansion is applied:
 *
 *                           inf.   B
 *                            -      2k
 * psi(x) = log(x) - 1/2x -   >   -------
 *                            -        2k
 *                           k=1   2k x
 *
 * where the B2k are Bernoulli numbers.
```

```
 *
 * ACCURACY:
 *    Relative error (except absolute when |psi| < 1):
 * arithmetic   domain     # trials      peak         rms
 *    DEC       0,30         2500       1.7e-16     2.0e-17
 *    IEEE      0,30        30000       1.3e-15     1.4e-16
 *    IEEE      -30,0       40000       1.5e-15     2.2e-16
 *
 * ERROR MESSAGES:
 *     message         condition      value returned
 * psi singularity    x integer <=0      MAXNUM
 */



/*							revers.c
 *
 *	Reversion of power series
 *
 *
 *
 * SYNOPSIS:
 *
 * extern int MAXPOL;
 * int n;
 * double x[n+1], y[n+1];
 *
 * polini(n);
 * revers( y, x, n );
 *
 *  Note, polini() initializes the polynomial arithmetic subroutines;
 *  see polyn.c.
 *
 *
 * DESCRIPTION:
 *
 * If
 *
 *          inf
 *           -       i
 *  y(x)  =  >   a  x
 *           -    i
 *          i=1
 *
 * then
 *
 *          inf
 *           -       j
 *  x(y)  =  >   A  y    ,
 *           -    j
 *          j=1
 *
 * where
 *                   1
 *         A    =   ---
 *          1        a
 *                    1
 *
 * etc.  The coefficients of x(y) are found by expanding
 *
 *          inf      inf
 *           -        -      i
 *  x(y)  =  >   A    >  a  x
 *           -    j   -   i
 *          j=1      i=1
 *
 *  and setting each coefficient of x , higher than the first,
 *  to zero.
 *
 *
 *
 * RESTRICTIONS:
 *
 *  y[0] must be zero, and y[1] must be nonzero.
 *
 */



/*						rgamma.c
 *
 *	Reciprocal gamma function
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, rgamma();
 *
 * y = rgamma( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns one divided by the gamma function of the argument.
 *
 * The function is approximated by a Chebyshev expansion in
 * the interval [0,1].  Range reduction is by recurrence
 * for arguments between -34.034 and +34.84425627277176174.
```

```
 * 1/MAXNUM is returned for positive arguments outside this
 * range.  For arguments less than -34.034 the cosecant
 * reflection formula is applied; lograrithms are employed
 * to avoid unnecessary overflow.
 *
 * The reciprocal gamma function has no singularities,
 * but overflow and underflow may occur for large arguments.
 * These conditions return either MAXNUM or 1/MAXNUM with
 * appropriate sign.
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    DEC      -30,+30       4000       1.2e-16     1.8e-17
 *    IEEE     -30,+30      30000       1.1e-15     2.0e-16
 * For arguments less than -34.034 the peak error is on the
 * order of 5e-15 (DEC), excepting overflow or underflow.
 */



/*                                                      round.c
 *
 *      Round double to nearest or even integer valued double
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, round();
 *
 * y = round(x);
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the nearest integer to x as a double precision
 * floating point result.  If x ends in 0.5 exactly, the
 * nearest even integer is chosen.
 *
 *
 *
 * ACCURACY:
 *
 * If x is greater than 1/(2*MACHEP), its closest machine
 * representation is already an integer, so rounding does
 * not change it.
 */



/*                                                      shichi.c
 *
 *      Hyperbolic sine and cosine integrals
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, Chi, Shi, shichi();
 *
 * shichi( x, &Chi, &Shi );
 *
 *
 * DESCRIPTION:
 *
 * Approximates the integrals
 *
 *                            x
 *                            -
 *                           | |   cosh t - 1
 *   Chi(x) = eul + ln x +   |    -----------  dt,
 *                         | |          t
 *                          -
 *                          0
 *
 *               x
 *               -
 *              | |  sinh t
 *   Shi(x) =   |    ------  dt
 *            | |       t
 *             -
 *             0
 *
 * where eul = 0.57721566490153286061 is Euler's constant.
 * The integrals are evaluated by power series for x < 8
 * and by Chebyshev expansions for x between 8 and 88.
 * For large x, both functions approach exp(x)/2x.
 * Arguments greater than 88 in magnitude return MAXNUM.
 *
 *
 * ACCURACY:
 *
 * Test interval 0 to 88.
 *                      Relative error:
 * arithmetic   function  # trials      peak         rms
 *    DEC          Shi       3000       9.1e-17
 *    IEEE         Shi      30000       6.9e-16     1.6e-16
```

```
 *        Absolute error, except relative when |Chi| > 1:
 *    DEC          Chi       2500       9.3e-17
 *    IEEE         Chi      30000       8.4e-16     1.4e-16
 */



/*                                                      sici.c
 *
 *      Sine and cosine integrals
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, Ci, Si, sici();
 *
 * sici( x, &Si, &Ci );
 *
 *
 * DESCRIPTION:
 *
 * Evaluates the integrals
 *
 *                          x
 *                          -
 *                         |  cos t - 1
 *   Ci(x) = eul + ln x +  |  --------- dt,
 *                         |      t
 *                        -
 *                         0
 *             x
 *             -
 *            |  sin t
 *   Si(x) =  |  ----- dt
 *            |    t
 *           -
 *            0
 *
 * where eul = 0.57721566490153286061 is Euler's constant.
 * The integrals are approximated by rational functions.
 * For x > 8 auxiliary functions f(x) and g(x) are employed
 * such that
 *
 * Ci(x) = f(x) sin(x) - g(x) cos(x)
 * Si(x) = pi/2 - f(x) cos(x) - g(x) sin(x)
 *
 *
 * ACCURACY:
 *    Test interval = [0,50].
 * Absolute error, except relative when > 1:
 * arithmetic   function   # trials      peak         rms
 *    IEEE        Si        30000       4.4e-16     7.3e-17
 *    IEEE        Ci        30000       6.9e-16     5.1e-17
 *    DEC         Si         5000       4.4e-17     9.0e-18
 *    DEC         Ci         5300       7.9e-17     5.2e-18
 */



/*                                                      simpsn.c        */
/* simpsn.c
 * Numerical integration of function tabulated
 * at equally spaced arguments
 */



/*                                                      simq.c
 *
 *      Solution of simultaneous linear equations AX = B
 *      by Gaussian elimination with partial pivoting
 *
 *
 *
 * SYNOPSIS:
 *
 * double A[n*n], B[n], X[n];
 * int n, flag;
 * int IPS[];
 * int simq();
 *
 * ercode = simq( A, B, X, n, flag, IPS );
 *
 *
 *
 * DESCRIPTION:
 *
 * B, X, IPS are vectors of length n.
 * A is an n x n matrix (i.e., a vector of length n*n),
 * stored row-wise: that is, A(i,j) = A[ij],
 * where ij = i*n + j, which is the transpose of the normal
 * column-wise storage.
 *
 * The contents of matrix A are destroyed.
 *
 * Set flag=0 to solve.
 * Set flag=-1 to do a new back substitution for different B vector
 * using the same A matrix previously reduced when flag=0.
 *
```

```
 * The routine returns nonzero on error; messages are printed.
 *
 *
 * ACCURACY:
 *
 * Depends on the conditioning (range of eigenvalues) of matrix A.
 *
 *
 * REFERENCE:
 *
 * Computer Solution of Linear Algebraic Systems,
 * by George E. Forsythe and Cleve B. Moler; Prentice-Hall, 1967.
 *
 */
```

```
/*                                                      sin.c
 *
 *      Circular sine
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, sin();
 *
 * y = sin( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Range reduction is into intervals of pi/4.  The reduction
 * error is nearly eliminated by contriving an extended precision
 * modular arithmetic.
 *
 * Two polynomial approximating functions are employed.
 * Between 0 and pi/4 the sine is approximated by
 *      x  +  x**3 P(x**2).
 * Between pi/4 and pi/2 the cosine is represented as
 *      1  -  x**2 Q(x**2).
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain      # trials      peak         rms
 *    DEC       0, 10       150000       3.0e-17     7.8e-18
 *    IEEE -1.07e9,+1.07e9  130000       2.1e-16     5.4e-17
 *
 * ERROR MESSAGES:
 *
 *   message           condition        value returned
 * sin total loss   x > 1.073741824e9      0.0
 *
 * Partial loss of accuracy begins to occur at x = 2**30
 * = 1.074e9.  The loss is not gradual, but jumps suddenly to
 * about 1 part in 10e7.  Results may be meaningless for
 * x > 2**49 = 5.6e14.  The routine as implemented flags a
 * TLOSS error for x > 2**30 and returns 0.0.
 */
```

```
/*                                                      cos.c
 *
 *      Circular cosine
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, cos();
 *
 * y = cos( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Range reduction is into intervals of pi/4.  The reduction
 * error is nearly eliminated by contriving an extended precision
 * modular arithmetic.
 *
 * Two polynomial approximating functions are employed.
 * Between 0 and pi/4 the cosine is approximated by
 *      1  -  x**2 Q(x**2).
 * Between pi/4 and pi/2 the sine is represented as
 *      x  +  x**3 P(x**2).
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain      # trials      peak         rms
 *    IEEE -1.07e9,+1.07e9  130000       2.1e-16     5.4e-17
 *    DEC        0,+1.07e9   17000       3.0e-17     7.2e-18
 */
```

```
/*                                                      sincos.c
 *
 *      Circular sine and cosine of argument in degrees
 *      Table lookup and interpolation algorithm
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, sine, cosine, flg, sincos();
 *
 * sincos( x, &sine, &cosine, flg );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns both the sine and the cosine of the argument x.
 * Several different compile time options and minimax
 * approximations are supplied to permit tailoring the
 * tradeoff between computation speed and accuracy.
 *
 * Since range reduction is time consuming, the reduction
 * of x modulo 360 degrees is also made optional.
 *
 * sin(i) is internally tabulated for 0 <= i <= 90 degrees.
 * Approximation polynomials, ranging from linear interpolation
 * to cubics in (x-i)**2, compute the sine and cosine
 * of the residual x-i which is between -0.5 and +0.5 degree.
 * In the case of the high accuracy options, the residual
 * and the tabulated values are combined using the trigonometry
 * formulas for sin(A+B) and cos(A+B).
 *
 * Compile time options are supplied for 5, 11, or 17 decimal
 * relative accuracy (ACC5, ACC11, ACC17 respectively).
 * A subroutine flag argument "flg" chooses betwen this
 * accuracy and table lookup only (peak absolute error
 * = 0.0087).
 *
 * If the argument flg = 1, then the tabulated value is
 * returned for the nearest whole number of degrees. The
 * approximation polynomials are not computed.  At
 * x = 0.5 deg, the absolute error is then sin(0.5) = 0.0087.
 *
 * An intermediate speed and precision can be obtained using
 * the compile time option LINTERP and flg = 1.  This yields
 * a linear interpolation using a slope estimated from the sine
 * or cosine at the nearest integer argument.  The peak absolute
 * error with this option is 3.8e-5.  Relative error at small
 * angles is about 1e-5.
 *
 * If flg = 0, then the approximation polynomials are computed
 * and applied.
 *
 *
 *
 * SPEED:
 *
 * Relative speed comparisons follow for 6MHz IBM AT clone
 * and Microsoft C version 4.0.  These figures include
 * software overhead of do loop and function calls.
 * Since system hardware and software vary widely, the
 * numbers should be taken as representative only.
 *
 *                      flg=0   flg=0   flg=1   flg=1
 *                      ACC11   ACC5    LINTERP Lookup only
 * In-line 8087 (/FPi)
 * sin(), cos()         1.0     1.0     1.0     1.0
 *
 * In-line 8087 (/FPi)
 * sincos()             1.1     1.4     1.9     3.0
 *
 * Software (/FPa)
 * sin(), cos()         0.19    0.19    0.19    0.19
 *
 * Software (/FPa)
 * sincos()             0.39    0.50    0.73    1.7
 *
 *
 *
 * ACCURACY:
 *
 * The accurate approximations are designed with a relative error
 * criterion.  The absolute error is greatest at x = 0.5 degree.
 * It decreases from a local maximum at i+0.5 degrees to full
 * machine precision at each integer i degrees.  With the
 * ACC5 option, the relative error of 6.3e-6 is equivalent to
 * an absolute angular error of 0.01 arc second in the argument
 * at x = i+0.5 degrees.  For small angles < 0.5 deg, the ACC5
 * accuracy is 6.3e-6 (.00063%) of reading; i.e., the absolute
 * error decreases in proportion to the argument.  This is true
 * for both the sine and cosine approximations, since the latter
 * is for the function 1 - cos(x).
 *
 * If absolute error is of most concern, use the compile time
 * option ABSERR to obtain an absolute error of 2.7e-8 for ACC5
 * precision.  This is about half the absolute error of the
 * relative precision option.  In this case the relative error
```

```
 * for small angles will increase to 9.5e-6 -- a reasonable
 * tradeoff.
 */



/*                                                      sindg.c
 *
 *      Circular sine of angle in degrees
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, sindg();
 *
 * y = sindg( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Range reduction is into intervals of 45 degrees.
 *
 * Two polynomial approximating functions are employed.
 * Between 0 and pi/4 the sine is approximated by
 *      x  +  x**3 P(x**2).
 * Between pi/4 and pi/2 the cosine is represented as
 *      1  -  x**2 P(x**2).
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain      # trials      peak         rms
 *    DEC       +-1000        3100      3.3e-17      9.0e-18
 *    IEEE      +-1000       30000      2.3e-16      5.6e-17
 *
 * ERROR MESSAGES:
 *
 *   message           condition        value returned
 * sindg total loss   x > 8.0e14 (DEC)      0.0
 *                    x > 1.0e14 (IEEE)
 *
 */



/*                                                      cosdg.c
 *
 *      Circular cosine of angle in degrees
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, cosdg();
 *
 * y = cosdg( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Range reduction is into intervals of 45 degrees.
 *
 * Two polynomial approximating functions are employed.
 * Between 0 and pi/4 the cosine is approximated by
 *      1  -  x**2 P(x**2).
 * Between pi/4 and pi/2 the sine is represented as
 *      x  +  x**3 P(x**2).
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain      # trials      peak         rms
 *    DEC      +-1000         3400       3.5e-17     9.1e-18
 *    IEEE     +-1000        30000       2.1e-16     5.7e-17
 *  See also sin().
 *
 */



/*                                                      sinh.c
 *
 *      Hyperbolic sine
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, sinh();
 *
 * y = sinh( x );
 *
 *
 *
```

```
 * DESCRIPTION:
 *
 * Returns hyperbolic sine of argument in the range MINLOG to
 * MAXLOG.
 *
 * 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
 *    DEC      +- 88        50000       4.0e-17     7.7e-18
 *    IEEE     +-MAXLOG     30000       2.6e-16     5.7e-17
 *
 */



/*                                                      spence.c
 *
 *      Dilogarithm
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, spence();
 *
 * y = spence( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Computes the integral
 *
 *                    x
 *                    -
 *                   | | log t
 * spence(x)  =  -   |   ----- dt
 *                 | |   t - 1
 *                  -
 *                  1
 *
 * for x >= 0.  A rational approximation gives the integral in
 * the interval (0.5, 1.5).  Transformation formulas for 1/x
 * and 1-x are employed outside the basic expansion range.
 *
 *
 *
 * ACCURACY:
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      0,4         30000       3.9e-15     5.4e-16
 *    DEC       0,4          3000       2.5e-16     4.5e-17
 *
 *
 */



/*                                                      sqrt.c
 *
 *      Square root
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, sqrt();
 *
 * y = sqrt( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the square root of x.
 *
 * Range reduction involves isolating the power of two of the
 * argument and using a polynomial approximation to obtain
 * a rough value for the square root.  Then Heron's iteration
 * is used three times to converge to an accurate value.
 *
 *
 *
 * ACCURACY:
 *
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    DEC       0, 10       60000       2.1e-17     7.9e-18
 *    IEEE      0,1.7e308   30000       1.7e-16     6.3e-17
 *
 *
```

```
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * sqrt domain        x < 0            0.0
 *
 */



/*							stdtr.c
 *
 *	Student's t distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * double t, stdtr();
 * short k;
 *
 * y = stdtr( 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 < -2, 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 <= 25.  The "domain" refers to t.
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE     -100,-2      50000       5.9e-15     1.4e-15
 *    IEEE     -2,100      500000       2.7e-15     4.9e-17
 */



/*							stdtri.c
 *
 *	Functional inverse of Student's t distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * double p, t, stdtri();
 * int k;
 *
 * t = stdtri( k, p );
 *
 *
 * DESCRIPTION:
 *
 * Given probability p, finds the argument t such that stdtr(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    .001,.999     25000       5.7e-15     8.0e-16
 *    IEEE    10^-6,.001    25000       2.0e-12     2.9e-14
 */



/*							struve.c
 *
 *      Struve function
 *
 *
 *
 * SYNOPSIS:
```

```
 *
 * double v, x, y, struve();
 *
 * y = struve( v, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Computes the Struve function Hv(x) of order v, argument x.
 * Negative x is rejected unless v is an integer.
 *
 * This module also contains the hypergeometric functions 1F2
 * and 3F0 and a routine for the Bessel function Yv(x) with
 * noninteger v.
 *
 *
 *
 * ACCURACY:
 *
 * Not accurately characterized, but spot checked against tables.
 *
 */



/*                                                      tan.c
 *
 *      Circular tangent
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, tan();
 *
 * y = tan( 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
 *    DEC      +-1.07e9      44000      4.1e-17     1.0e-17
 *    IEEE     +-1.07e9      30000      2.9e-16     8.1e-17
 *
 * ERROR MESSAGES:
 *
 *   message         condition          value returned
 * tan total loss   x > 1.073741824e9     0.0
 *
 */



/*                                                      cot.c
 *
 *      Circular cotangent
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, cot();
 *
 * y = cot( 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      2.9e-16     8.2e-17
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition          value returned
```

```
 * cot total loss   x > 1.073741824e9       0.0
 * cot singularity  x = 0                  INFINITY
 *
 */



/*                                                      tandg.c
 *
 *      Circular tangent of argument in degrees
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, tandg();
 *
 * y = tandg( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the circular tangent of the argument x in degrees.
 *
 * 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
 *    DEC      0,10          8000      3.4e-17      1.2e-17
 *    IEEE     0,10         30000      3.2e-16      8.4e-17
 *
 * ERROR MESSAGES:
 *
 *   message         condition          value returned
 * tandg total loss   x > 8.0e14 (DEC)      0.0
 *                    x > 1.0e14 (IEEE)
 * tandg singularity  x = 180 k  +  90     MAXNUM
 */



/*                                                      cotdg.c
 *
 *      Circular cotangent of argument in degrees
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, cotdg();
 *
 * y = cotdg( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the circular cotangent of the argument x in degrees.
 *
 * 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].
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition          value returned
 * cotdg total loss   x > 8.0e14 (DEC)      0.0
 *                    x > 1.0e14 (IEEE)
 * cotdg singularity  x = 180 k            MAXNUM
 */



/*                                                      tanh.c
 *
 *      Hyperbolic tangent
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, tanh();
 *
 * y = tanh( x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns hyperbolic tangent of argument in the range MINLOG to
 * MAXLOG.
 *
```

```
 * 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
 *    DEC       -2,2        50000       3.3e-17     6.4e-18
 *    IEEE      -2,2        30000       2.5e-16     5.8e-17
 *
 */
```

```
/*                                                      unity.c
 *
 * Relative error approximations for function arguments near
 * unity.
 *
 *    log1p(x) = log(1+x)
 *    expm1(x) = exp(x) - 1
 *    cosm1(x) = cos(x) - 1
 *
 */
```

```
/*                                                      yn.c
 *
 *      Bessel function of second kind of integer order
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, yn();
 * int n;
 *
 * y = yn( n, x );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of order n, where n is a
 * (possibly negative) integer.
 *
 * The function is evaluated by forward recurrence on
 * n, starting with values computed by the routines
 * y0() and y1().
 *
 * If n = 0 or 1 the routine for y0 or y1 is called
 * directly.
 *
 *
 *
 * ACCURACY:
 *
 *
 *                      Absolute error, except relative
 *                      when y > 1:
 * arithmetic   domain     # trials      peak         rms
 *    DEC       0, 30        2200       2.9e-16     5.3e-17
 *    IEEE      0, 30       30000       3.4e-15     4.3e-16
 *
 *
 * ERROR MESSAGES:
 *
 *   message         condition      value returned
 * yn singularity   x = 0              MAXNUM
 * yn overflow                         MAXNUM
 *
 * Spot checked against tables for x, n between 0 and 100.
 *
 */
```

```
/*                                                      zeta.c
 *
 *      Riemann zeta function of two arguments
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, q, y, zeta();
 *
 * y = zeta( x, q );
 *
 *
 *
 * DESCRIPTION:
 *
 *
 *
```

```
 *                 inf.
 *                  -        -x
 *   zeta(x,q)  =   >   (k+q)
 *                  -
 *                 k=0
 *
 * where x > 1 and q is not a negative integer or zero.
 * The Euler-Maclaurin summation formula is used to obtain
 * the expansion
 *
 *                n
 *                -       -x
 * zeta(x,q)  =   >  (k+q)
 *                -
 *               k=1
 *
 *           1-x                 inf.  B   x(x+1)...(x+2j)
 *      (n+q)           1         -     2j
 *  +  ---------  -  -------  +   >    --------------------
 *        x-1              x      -                   x+2j+1
 *                   2(n+q)      j=1       (2j)! (n+q)
 *
 * where the B2j are Bernoulli numbers.  Note that (see zetac.c)
 * zeta(x,1) = zetac(x) + 1.
 *
 *
 *
 * ACCURACY:
 *
 *
 *
 * REFERENCE:
 *
 * Gradshteyn, I. S., and I. M. Ryzhik, Tables of Integrals,
 * Series, and Products, p. 1073; Academic Press, 1980.
 *
 */



/*                                                      zetac.c
 *
 *      Riemann zeta function
 *
 *
 *
 * SYNOPSIS:
 *
 * double x, y, zetac();
 *
 * y = zetac( x );
 *
 *
 *
 * DESCRIPTION:
 *
 *
 *
 *                inf.
 *                 -    -x
 *   zetac(x)  =   >   k   ,   x > 1,
 *                 -
 *                k=2
 *
 * is related to the Riemann zeta function by
 *
 *      Riemann zeta(x) = zetac(x) + 1.
 *
 * Extension of the function definition for x < 1 is implemented.
 * Zero is returned for x > log2(MAXNUM).
 *
 * An overflow error may occur for large negative x, due to the
 * gamma function in the reflection formula.
 *
 * ACCURACY:
 *
 * Tabulated values have full machine accuracy.
 *
 *                      Relative error:
 * arithmetic   domain     # trials      peak         rms
 *    IEEE      1,50        10000       9.8e-16     1.3e-16
 *    DEC       1,50         2000       1.1e-16     1.9e-17
 *
 *
 */
```


Last update: 5 October 2014