Cephes Mathematical Library

Documentation for single.zip.
Documentation for double.zip.
Documentation for ldouble.zip.
Documentation for 128bit.shar.gz.
Documentation for qlib.zip.

Extended Precision Special Functions Suite Documentation

These are high precision a priori check routines used mainly to design and test lower precision function programs. For standard precision codes, see the archives and descriptions listed above.
Select function name for additional information.

- qacosh.c, hyperbolic arccosine
- qairy.c, Airy functions
- qasin.c, circular arcsine
- qacos.c, circular arccosine
- qasinh.c, hyperbolic arcsine
- qatanh.c, hyperbolic arctangent
- qatn.c, circular arctangent
- qatn2.c, quadrant correct arctangent
- qbeta.c, beta function
- qcbrt.c, cube root
- qcgamma.c, complex gamma function
- qclgam.c, log of complex gamma function
- qchyp1f1.c, complex confluent hypergeometric function
- qcmplx.c, complex arithmetic
- qcos.c, circular cosine
- qcosh.c, hyperbolic cosine
- qcpolylog.c, complex polylogarithms
- qdawsn.c, Dawson's integral
- qei.c, exponential integral
- qellie.c, incomplete elliptic integral of the second kind
- qellik.c, incomplete elliptic integral of the first kind
- qellpe.c, complete elliptic integral of the second kind
- qellpj.c, Jacobian elliptic ntegral
- qellpk.c, complete elliptic integral of the first kind
- qerf.c, error function
- qerfc.c, complementary error function
- qeuclid.c, rational arithmetic
- qexp.c, exponential function
- qexp10.c, antilogarithm
- qexp2.c, base 2 exponential function
- qexpn.c, exponential integral
- qfloor.c, floor, round
- qflt.c, extended precision floating point routines
- qflta.c, extended precision floating point utilities
- qfresnl.c, Fresnel integrals
- qlgam.c, log of gamma function
- qgamma.c, gamma function
- qhyp2f1.c, Gauss hypergeometric function 2F1
- qhyp.c, Confluent hypergeometric function 1F1
- qigam.c, incomplete gamma integral
- qigami.c, inverse of incomplete gamma integral
- qin.c, modified Bessel function I of noninteger order
- qincb.c, incomplete beta integral
- qincbi.c, inverse of incomplete beta integral
- qine.c, modified Bessel function I of noninteger order, exponentially scaled
- qjn.c, Bessel function noninteger order
- qkn.c, modified Bessel function of the third kind, integer order
- qkne.c, modified Bessel function of the third kind, integer order, exponentially scaled
- qlog.c, natural logarithm
- qlog1.c, relative error logarithm
- qlog10.c, common logarithm
- qndtr.c, normal distribution function
- qndtri.c, inverse of normal distribution function
- qpolylog.c, polylogarithms
- qpolyr.c, arithmetic on polynomials with rational coefficients
- qpow.c, power function
- qprob.c, various probability integrals
- qbdtr, binomial distribution
- qbdtrc, complemented binomial distribution
- qbdtri, inverse of binomial distribution
- qchdtr, chi-square distribution
- qchdtc, complemented chi-square distribution
- qchdti, inverse of chi-square distribution
- qfdtr, F distribution
- qfdtrc, complemented F distribution

```
/*                                                      qacosh.c
 *
 *      Inverse hyperbolic cosine
 *
 *
 *
 * SYNOPSIS:
 *
 * int qacosh( x, y )
 * QELT *x, *y;
 *
 * qacosh( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * acosh(x)  =  log( x + sqrt( (x-1)(x+1) ).
 *
 */
```

```
/*                                                      qairy.c
 *
 *      Airy functions
 *
 *
 *
 * SYNOPSIS:
 *
 * int qairy( x, ai, aip, bi, bip );
 * QELT *x, *ai, *aip, *bi, *bip;
 *
 * qairy( 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 asymptotic expansion for large x.
 *
 *
 * ACCURACY:
 *
 * The asymptotic expansion is truncated at less than full working precision.
 *
 */
```

```
/*                                                      qasin.c
 *
 *      Inverse circular sine
 *
 *
 *
 * SYNOPSIS:
 *
```

```
 * int qasin( x, y );
 * QELT *x, *y;
 *
 * qasin( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns radian angle between -pi/2 and +pi/2 whose sine is x.
 *
 *    asin(x) = arctan (x / sqrt(1 - x^2))
 *
 * If |x| > 0.5 it is transformed by the identity
 *
 *    asin(x) = pi/2 - 2 asin( sqrt( (1-x)/2 ) ).
 *
 */



/*                                                      qacos
 *
 *      Inverse circular cosine
 *
 *
 *
 * SYNOPSIS:
 *
 * int qacos( x, y );
 * QELT x[], y[];
 *
 * qacos( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns radian angle between 0 and pi whose cosine
 * is x.
 *
 * acos(x) = pi/2 - asin(x)
 *
 */



/*                                                      qasinh.c
 *
 *      Inverse hyperbolic sine
 *
 *
 *
 * SYNOPSIS:
 *
 * int qasinh( x, y );
 * QELT *x, *y;
 *
 * qasinh( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns inverse hyperbolic sine of argument.
 *
 *     asinh(x) = log( x + sqrt(1 + x*x) ).
 *
 * For very large x, asinh(x) = log x  +  log 2.
 *
 */



/*                                                      qatanh.c
 *
 *      Inverse hyperbolic tangent
 *
 *
 *
 * SYNOPSIS:
 *
 * int qatanh( x, y );
 * QELT x[], y[];
 *
 * qatanh( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns inverse hyperbolic tangent of argument.
 *
 *        atanh(x) = 0.5 * log( (1+x)/(1-x) ).
 *
 * For very small x, the first few terms of the Taylor series
 * are summed.
 *
 */
```

```
/*							qatn
 *
 *	Inverse circular tangent
 *	(arctangent)
 *
 *
 *
 * SYNOPSIS:
 *
 * int qatn( x, y );
 * QELT *x, *y;
 *
 * qatn( x, y );
 *
 *
 *
 * 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 pi/8.
 *
 *                    2     2     2
 *              x    x   4 x   9 x
 * arctan(x) =  ---  ---  ----  ----  ...
 *             1 -  3 -  5 -   7 -
 *
 */



/*							qatn2
 *
 *	Quadrant correct inverse circular tangent
 *
 *
 *
 * SYNOPSIS:
 *
 * int qatn2( y, x, z );
 * QELT *x, *y, *z;
 *
 * qatn2( y, x, z );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns radian angle -PI < z < PI whose tangent is y/x.
 *
 */



/*							qbeta.c
 *
 *	Beta function
 *
 *
 *
 * SYNOPSIS:
 *
 * int qbeta( a, b, y );
 * QELT *a, *b, *y;
 *
 * qbeta( a, b, y );
 *
 *
 *
 * DESCRIPTION:
 *
 *                   -     -
 *                  | (a) | (b)
 * beta( a, b )  =  -----------.
 *                     -
 *                    | (a+b)
 *
 */



/*							qcbrt.c
 *
 *	Cube root
 *
 *
 *
 * SYNOPSIS:
 *
 * int qcbrt( x, y );
 * QELT *x, *y;
 *
 * qcbrt( x, y );
 *
 *
 *
```

```
 * DESCRIPTION:
 *
 * Returns the cube root of the argument, which may be negative.
 *
 */



/*                                                      qcgamma
 *
 *      Complex gamma function
 *
 *
 *
 * SYNOPSIS:
 *
 * int qcgamma( x, y );
 * qcmplx *x, *y;
 *
 * qcgamma( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns complex-valued gamma function of the complex argument.
 *
 * gamma(x) = exp (log(gamma(x)))
 *
 */



/*                                                      qclgam
 *
 *      Natural logarithm of complex gamma function
 *
 *
 *
 * SYNOPSIS:
 *
 * int qclgam( x, y );
 * qcmplx *x, *y;
 *
 * qclgam( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the base e (2.718...) logarithm of the complex gamma
 * function of the argument.
 *
 * The logarithm of the gamma function is approximated by the
 * logarithmic version of Stirling's asymptotic formula.
 * Arguments of real part less than +32 are increased by recurrence.
 * The cosecant reflection formula is employed for arguments
 * having real part less than -34.
 *
 */



/*                                                          qchyp1f1.c
 *
 *      confluent hypergeometric function
 *
 *                          1           2
 *                       a x    a(a+1) x
 *   F ( a,b;x )  =  1 + ---- + --------- + ...
 *  1 1                  b 1!   b(b+1) 2!
 *
 *
 * Series summation terminates at 70 bits accuracy.
 *
 */



/*                              qcmplx.c
 * Q type complex number arithmetic
 *
 * The syntax of arguments in
 *
 * cfunc( a, b, c )
 *
 * is
 * c = b + a
 * c = b - a
 * c = b * a
 * c = b / a.
 */



/*                                                      qcos.c
 *
 *      Circular cosine
 *
 *
 *
```

```
 * SYNOPSIS:
 *
 * int qcos( x, y );
 * QELT *x, *y;
 *
 * qcos( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * cos(x) = sin(pi/2 - x)
 *
 */



/*                                                        qcosh.c
 *
 *      Hyperbolic cosine
 *
 *
 *
 * SYNOPSIS:
 *
 * int qcosh(x, y);
 * QELT *x, *y;
 *
 * qcosh(x, y);
 *
 *
 *
 * DESCRIPTION:
 *
 * cosh(x)  =  ( exp(x) + exp(-x) )/2.
 *
 */



/*
                                                          qcpolylog.c
   Complex polylogarithms.


               inf   k
                -   x
   Li (x)  =    >   ---
     n          -     n
               k=1   k


                 x
                  -
                 | |  -ln(1-t)
    Li (x)  =    |    -------- dt
      2        | |       t
                -
                 0


                1-x
                  -
                 | |  ln t
           =     |    ------ dt   =   spence(1-x)
               | |    1 - t
                -
                 1


                       2       3
                      x       x
            =  x  +  ---  +  ---  +  ...
                      4       9


  d                 1
  --   Li (x)  =   ---  Li   (x)
  dx     n          x     n-1

  */



/*                                                        qdawsn.c
 *
 *      Dawson's Integral
 *
 *
 *
 * SYNOPSIS:
 *
 * int qdawsn( x, y );
 * QELT *x, *y;
 *
 * qdawsn( x, y );
 *
 *
 *
 * DESCRIPTION:
```

```
 *
 * Approximates the integral
 *
 *                              x
 *                              -
 *                       2     | |        2
 *  dawsn(x)  =  exp( -x  )    |    exp( t  ) dt
 *                           | |
 *                            -
 *                            0
 *
 *
 *
 * ACCURACY:
 *
 * Series expansions are truncated at NBITS/2.
 *
 */



/*                                                      qei.c
 *
 *      Exponential integral
 *
 *
 * SYNOPSIS:
 *
 * QELT *x, *y;
 *
 * qei( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 *               x
 *                -     t
 *               | |   e
 *    Ei(x) =   -|-   ---  dt .
 *             | |     t
 *              -
 *             -inf
 *
 * Not defined for x <= 0.
 * See also qexpn.c.
 *
 * ACCURACY:
 *
 * Series truncated at NBITS/2.
 *
 */



/*                                                      qellie.c
 *
 *      Incomplete elliptic integral of the second kind
 *
 *
 *
 * SYNOPSIS:
 *
 * int qellie( phi, m, y );
 * QELT *phi, *m, *y;
 *
 * qellie( phi, m, y );
 *
 *
 *
 * 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:
 *
 * Sequence terminates at NBITS/2.
 *
 */



/*                                                      qellik.c
 *
```

```
 *      Incomplete elliptic integral of the first kind
 *
 *
 *
 * SYNOPSIS:
 *
 * int qellik( phi, m, y );
 * QELT *phi, *m, *y;
 *
 * qellik( phi, m, y );
 *
 *
 *
 * 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:
 *
 * Sequence terminates at NBITS/2.
 *
 */



/*                                                      qellpe.c
 *
 *      Complete elliptic integral of the second kind
 *
 *
 *
 * SYNOPSIS:
 *
 * int qellpe(x, y);
 * QELT *x, *y;
 *
 * qellpe(x, y);
 *
 *
 *
 * DESCRIPTION:
 *
 * Approximates the integral
 *
 *
 *            pi/2
 *             -
 *            | |                 2
 * E(m)  =    |    sqrt( 1 - m sin t ) dt
 *          | |
 *           -
 *            0
 *
 * Where m = 1 - m1, using the arithmetic-geometric mean method.
 *
 *
 * ACCURACY:
 *
 * Method terminates at NBITS/2.
 *
 */



/*                                                      qellpj.c
 *
 *      Jacobian Elliptic Functions
 *
 *
 *
 * SYNOPSIS:
 *
 * int qellpj( u, m, sn, cn, dn, ph );
 * QELT *u, *m;
 * QELT *sn, *cn, *dn, *ph;
 *
 * qellpj( u, m, sn, cn, dn, ph );
 *
 *
 *
 * 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:
 *
 * Truncated at 70 bits.
 *
 */



/*                                                      qellpk.c
 *
 *      Complete elliptic integral of the first kind
 *
 *
 *
 * SYNOPSIS:
 *
 * int qellpk(x, y);
 * QELT *x, *y;
 *
 * qellpk(x, y);
 *
 *
 *
 * DESCRIPTION:
 *
 * Approximates the integral
 *
 *
 *
 *            pi/2
 *             -
 *            | |
 *            |           dt
 * K(m)  =    |    ------------------
 *            |                   2
 *          | |    sqrt( 1 - m sin t )
 *           -
 *            0
 *
 * where m = 1 - m1, using the arithmetic-geometric mean method.
 *
 * 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:
 *
 * Truncated at NBITS/2.
 *
 */



/*                                                      qerf.c
 *
 *      Error function
 *
 *
 *
 * SYNOPSIS:
 *
 * int qerf( x, y );
 * QELT *x, *y;
 *
 * qerf( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * The integral is
 *
 *                           x
 *                            -
 *                 2         | |          2
 *   erf(x)  =  --------     |    exp( - t  ) dt.
 *              sqrt(pi)   | |
 *                          -
 *                           0
 *
```

```
 *
 */



/*                                                      qerfc.c
 *
 *      Complementary error function
 *
 *
 *
 * SYNOPSIS:
 *
 * int qerfc( x, y );
 * QELT *x, *y;
 *
 * qerfc( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 *
 *  1 - erf(x) =
 *
 *                           inf.
 *                             -
 *                  2         | |          2
 *   erfc(x)  =  --------     |    exp( - t  ) dt
 *               sqrt(pi)   | |
 *                           -
 *                            x
 *
 */



/*                                                      qeuclid.c
 *
 * Rational arithmetic routines
 *
 * 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 g.c.d.
 *
 * Note: arguments are assumed,
 * without checking,
 * to be integer valued.
 */



/*                                                      qexp.c
 *
 *      Exponential function check routine
 *
 *
 *
 * SYNOPSIS:
 *
 * int qexp( x, y );
 * QELT *x, *y;
 *
 * qexp( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns e (2.71828...) raised to the x power.
 *
 */



/*                                                      exp10.c
 *
 *      Base 10 exponential function
 *      (Common antilogarithm)
 *
 *
 *
 * SYNOPSIS:
 *
 * int qexp10( x, y );
 * QELT *x, *y;
 *
 * qexp10( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns 10 raised to the x power.
 *
 *   x      x ln 10
 * 10   =  e
 *
 */
```

```
/*                                                      qexp2.c
 *
 *      Check routine for base 2 exponential function
 *
 *
 *
 * SYNOPSIS:
 *
 * int qexp2( x, y );
 * QELT *x, *y;
 *
 * qexp2( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns 2 raised to the x power.
 *
 *        x      ln 2  x      x ln 2
 * y  =  2  = ( e     )   =  e
 *
 */
```

```
/*                                                      qexpn.c
 *
 *              Exponential integral En
 *
 *
 *
 * SYNOPSIS:
 *
 * int qexpn( n, x, y );
 * int n;
 * QELT *x, *y;
 *
 * qexpn( n, x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Evaluates the exponential integral
 *
 *                 inf.
 *                   -
 *                  | |   -xt
 *                  |    e
 *      E (x)  =    |    ----  dt.
 *       n          |      n
 *                | |     t
 *                 -
 *                  1
 *
 *
 * Both n and x must be nonnegative.
 *
 *
 * ACCURACY:
 *
 * Series expansions are truncated at less than full working precision.
 *
 */
```

```
/*                                                      qfloor.c
 * qfloor - largest integer not greater than x
 * qround - nearest integer to x
 */
```

```
/*                                                      qflt.c
 *                      QFLOAT
 *
 *      Extended precision floating point routines
 *
 *      asctoq( string, q )     ascii string to q type
 *      dtoq( &d, q )           DEC double precision to q type
 *      etoq( &d, q )           IEEE double precision to q type
 *      e24toq( &d, q )         IEEE single precision to q type
 *      e113toq( &d, q )        128-bit long double precision to q type
 *      ltoq( &l, q )           long integer to q type
 *      qabs(q)                 absolute value
 *      qadd( a, b, c )         c = b + a
 *      qclear(q)               q = 0
 *      qcmp( a, b )            compare a to b
 *      qdiv( a, b, c )         c = b / a
 *      qifrac( x, &l, frac )   x to integer part l and q type fraction
 *      qfrexp( x, l, y )       find exponent l and fraction y between .5 and 1
 *      qldexp( x, l, y )       multiply x by 2^l
 *      qinfin( x )             set x to infinity, leaving its sign alone
 *      qmov( a, b )            b = a
 *      qmul( a, b, c )         c = b * a
 *      qmuli( a, b, c )        c = b * a, a has only 16 significant bits
```

```
 *      qisneg(q)               returns sign of q
 *      qneg(q)                 q = -q
 *      qnrmlz(q)               adjust exponent and mantissa
 *      qsub( a, b, c )         c = b - a
 *      qtoasc( a, s, n )       q to ASCII string, n digits after decimal
 *      qtod( q, &d )           convert q type to DEC double precision
 *      qtoe( q, &d )           convert q type to IEEE double precision
 *      qtoe24( q, &d )         convert q type to IEEE single precision
 *      qtoe113( q, &d )        convert q type to 128-bit long double precision
 *
 * Data structure of the number (a "word" is 16 bits)
 *
 *      sign word               (0 for positive, -1 for negative)
 *      exponent                (EXPONE for 1.0)
 *      high guard word         (always zero after normalization)
 *      N-1 mantissa words      (most significant word first,
 *                               most significant bit is set)
 *
 * Numbers are stored in C language as arrays.  All routines
 * use pointers to the arrays as arguments.
 *
 * The result is always normalized after each arithmetic operation.
 * All arithmetic results are chopped. No rounding is performed except
 * on conversion to double precision.
 */



/*              qflta.c
 * Utilities for extended precision arithmetic, called by qflt.c.
 * These should all be written in machine language for speed.
 *
 * addm( x, y )         add significand of x to that of y
 * shdn1( x )           shift significand of x down 1 bit
 * shdn8( x )           shift significand of x down 8 bits
 * shdn16( x )          shift significand of x down 16 bits
 * shup1( x )           shift significand of x up 1 bit
 * shup8( x )           shift significand of x up 8 bits
 * shup16( x )          shift significand of x up 16 bits
 * divm( a, b )         divide significand of a into b
 * mulm( a, b )         multiply significands, result in b
 * mdnorm( x )          normalize and round off
 *
 * Copyright (c) 1984 - 1988 by Stephen L. Moshier.  All rights reserved.
 */



/*                                                      qfresnl
 *
 *      Fresnel integral
 *
 *
 *
 * SYNOPSIS:
 *
 * int qfresnl( x, s, c );
 * QELT *x, *s, *c;
 *
 * qfresnl( 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 large x 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 )
 *
 * Routine qfresfg computes f and g.
 *
 *
 * ACCURACY:
 *
 * Series expansions are truncated at less than full working precision.
 */
```

```
/*                                                      qlgam
 *
 *      Natural logarithm of gamma function
 *
 *
 *
 * SYNOPSIS:
 *
 * int qlgam( x, y );
 * QELT *x, *y;
 *
 * qlgam( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the base e (2.718...) logarithm of the absolute
 * value of the gamma function of the argument.
 *
 */



/*                                                      qgamma
 *
 *      Gamma function
 *
 *
 *
 * SYNOPSIS:
 *
 * int qgamma( x, y );
 * QELT *x, *y;
 *
 * qgamma( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns gamma function of the argument.
 *
 * qgamma(x) = exp(qlgam(x))
 *
 */



/*                                                      hyp2f1.c
 *
 *      Gauss hypergeometric function   F
 *                                     2 1
 *
 *
 * SYNOPSIS:
 *
 * int qhy2f1( a, b, c, x, y );
 * QELT *a, *b, *c, *x, *y;
 *
 * qhy2f1( a, b, c, x, y );
 *
 *
 * 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
 *
 *
 * ACCURACY:
 *
 * Expansions are set to terminate at less than full working precision.
 *
 */



/*                                                      qhyp.c
 *
 *      Confluent hypergeometric function
 *
 *
 *
 * SYNOPSIS:
 *
 * int qhyp( a, b, x, y );
 * QELT *a, *b, *x, *y;
 *
 * qhyp( a, b, x, y );
 *
 *
 *
 * 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!
 *
 *
 * ACCURACY:
 *
 * Series expansion is truncated at less than full working precision.
 *
 */



/*                                                      qigam.c
 *      Check routine for incomplete gamma integral
 *
 *
 *
 * SYNOPSIS:
 *
 * For the left tail:
 * int qigam( a, x, y );
 * QELT *a, *x, *y;
 * qigam( a, x, y );
 *
 * For the right tail:
 * int qigamc( a, x, y );
 * QELT *a, *x, *y;
 * qigamc( a, x, y );
 *
 *
 * 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:
 *
 * Expansions terminate at less than full working precision.
 *
 */



/*                                                      qigami()
 *
 *      Inverse of complemented imcomplete gamma integral
 *
 *
 *
 * SYNOPSIS:
 *
 * int qigami( a, p, x );
 * QELT *a, *p, *x;
 *
 * qigami( a, p, x );
 *
 * DESCRIPTION:
 *
 * The program refines an initial estimate generated by the
 * double precision routine igami to find the root of
 *
 *  igamc(a,x) - p = 0.
 *
 *
 * ACCURACY:
 *
 * Set to do just one Newton-Raphson iteration.
 *
 */



/*                                                      qin.c
 *
 *      Modified Bessel function I of noninteger order
 *
 *
 * SYNOPSIS:
 *
 * int qin( v, x, y );
 * QELT *v, *x, *y;
```

```
 *
 * qin( v, x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns modified Bessel function of order v of the
 * argument.
 *
 * The power series is
 *
 *               inf      2   k
 *             v  -     (z /4)
 * I (z) = (z/2)   >  --------------
 *  v              -       -
 *               k=0  k! | (v+k+1)
 *
 *
 * For large x,
 *                                  2          2       2
 *            exp(z)            u - 1     (u - 1 )(u - 3 )
 * I (z)  =  ------------ { 1 - -------- + ---------------- + ...}
 *  v        sqrt(2 pi z)              1              2
 *                             1! (8z)        2! (8z)
 *
 * asymptotically, where
 *
 *            2
 *     u = 4 v .
 *
 *
 * x <= 0 is not supported.
 *
 * Series expansion is truncated at less than full working precision.
 *
 */



/*                                                     qincb.c
 *
 *      Incomplete beta integral
 *
 *
 * SYNOPSIS:
 *
 * int qincb( a, b, x, y );
 * QELT *a, *b, *x, *y;
 *
 * qincb( a, b, x, y );
 *
 *
 * DESCRIPTION:
 *
 * Returns incomplete beta integral of the arguments, evaluated
 * from zero to x.
 *
 *                   x
 *     -             -
 *    | (a+b)       | |  a-1     b-1
 *  -----------     |   t   (1-t)   dt.
 *   -     -      | |
 *  | (a) | (b)   -
 *                  0
 *
 *
 * ACCURACY:
 *
 * Series expansions terminate at less than full working precision.
 *
 */



/*                                                     qincbi()
 *
 *      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 up to 10 Newton iterations to find the
 * root of incbet(a,b,x) - y = 0.
 *
 */
```

```
/*                                                      qine.c
 *
 *      Modified Bessel function I of noninteger order
 *      Exponentially scaled
 *
 *
 * SYNOPSIS:
 *
 * int qine( v, x, y );
 * QELT *v, *x, *y;
 *
 * qine( v, x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns modified Bessel function of order v of the
 * argument.
 *
 * The power series is
 *
 *               inf      2   k
 *             v  -     (z /4)
 * I (z) = (z/2)   >  -------------
 *  v              -       -
 *               k=0  k! | (v+k+1)
 *
 *
 * For large x,
 *                                    2          2       2
 *            exp(z)            u - 1     (u - 1 )(u - 3 )
 * I (z)  =  ------------ { 1 - -------- + ---------------- + ...}
 *  v        sqrt(2 pi z)              1              2
 *                             1! (8z)        2! (8z)
 *
 * asymptotically, where
 *
 *            2
 *     u = 4 v .
 *
 *
 * The routine returns
 *
 *    sqrt(x) exp(-x) I (x)
 *                     v
 *
 * x <= 0 is not supported.
 *
 * Series expansion is truncated at less than full working precision.
 *
 */



/*                                                      qjn.c
 *
 *      Bessel function of noninteger order
 *
 *
 *
 * SYNOPSIS:
 *
 * int qjn( v, x, y );
 * QELT *v, *x, *y;
 *
 * qjn( v, x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of order v of the argument,
 * where v is real.  Negative x is allowed if v is an integer.
 *
 * Two expansions are used: the ascending power series and the
 * Hankel expansion for large v.  If v is not too large, it
 * is reduced by recurrence to a region of better accuracy.
 *
 */



/*                                                      kn.c
 *
 *      Modified Bessel function, third kind, integer order
 *
 *
 *
 * SYNOPSIS:
 *
 * int qkn( n, x, y );
 * int n;
 * QELT *x, *y;
 *
 * qkn( n, x, y );
 *
 *
```

```
 *
 * 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:
 *
 * Series expansions are set to terminate at less than full
 * working precision.
 *
 */



/*   qkne.c
 *
 *  exp(x) sqrt(x) Kn(x)
 */



/*                                                      qlog.c
 *
 *      Natural logarithm
 *
 *
 *
 * SYNOPSIS:
 *
 * int qlog( x, y );
 * QELT *x, *y;
 *
 * qlog( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the base e (2.718...) logarithm of x.
 *
 * After reducing the argument into the interval [1/sqrt(2), sqrt(2)],
 * the logarithm is calculated by
 *
 *       x-1
 * w  =  ---
 *       x+1
 *                     3     5
 *                    w     w
 * ln(x) / 2  =  w + --- + --- + ...
 *                    3     5
 */



/*                                                      qlog1.c
 *
 *      Relative error logarithm
 *
 *
 *
 * SYNOPSIS:
 *
 * int qlog1( x, y );
 * QELT *x, *y;
 *
 * qlog1( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns the base e (2.718...) logarithm of 1 + x.
 *
 * For small x, this continued fraction is used:
 *
 *     1+z
 * w = ---
 *     1-z
 *
 *               2   2   2
 *         2z   z  4z  9z
 * ln(w) = --- --- --- --- ...
 *         1 - 3 - 5 - 7 -
 *
 * after setting z = x/(x+2).
 *
 */



/*                                                      qlog10.c
 *
 *      Common logarithm
 *
 *
 *
```

```
 * SYNOPSIS:
 *
 * int qlog10( x, y );
 * QELT *x, *y;
 *
 * qlog10( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns base 10, or common, logarithm of x.
 *
 * log  (x) = log  (e) log (x)
 *    10         10       e
 *
 */



/*                                                      qndtr.c
 *
 *      Normal distribution function
 *
 *
 *
 * SYNOPSIS:
 *
 * int qndtr( x, y );
 * QELT *x, *y;
 *
 * qndtr( x, y );
 *
 *
 *
 * 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).
 *
 */



/*                                                      qndtri.c
 *
 *      Inverse of Normal distribution function
 *
 *
 *
 * SYNOPSIS:
 *
 * int qndtri(y, x);
 * QELT *y, *x;
 *
 * qndtri(y, x);
 *
 *
 *
 * 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.
 *
 * The routine refines a trial solution computed by the double
 * precision function ndtri.
 *
 */



/*                                                      qplanck.c
 *  Integral of Planck's radiation formula.
 *
 *                                      1
 *                               ------------------
 *                                5
 *                               t  (exp(1/bw) - 1)
 *
 * Set
 *   b = T/c2
 *   u = exp(1/bw)
 *
 *  In terms of polylogarithms Li_n(u), the integral is
 *
 *                  (           Li (u)      Li (u)                  )
```

```
 *     1         4 (              3           2          log(1-u)  )
 *    ----  -  6 b  ( Li (u)  -  ------  +  --------  +  ---------- )
 *      4          (   4          bw             2            3  )
 *    4 w          (                        2 (bw)       6 (bw)   )
 *
 *   Since u > 1, the Li_n are complex valued.  This is not
 * the best way to calculate the result, which is real, but it
 * is adopted as a the priori formula against which other formulas
 * can be verified.
 */



/*                                                      qpolylog.c
 *
   Polylogarithms.


             inf   k
              -   x
   Li (x)  =   >   ---
     n        -     n
             k=1   k


               x
                -
               | |  -ln(1-t)
    Li (x)  =   |    -------- dt
      2       | |       t
               -
               0


              1-x
                -
               | |  ln t
           =    |    ------ dt   =   spence(1-x)
              | |    1 - t
               -
               1


                       2       3
                      x       x
            =  x  +  ---  +  ---  +  ...
                      4       9


  d                  1
  --   Li (x)  =   ---  Li   (x)
  dx     n          x     n-1

  Series expansions are set to terminate at less than full
  working precision.

  */



/*                                                      qpolyr.c
 *
 * 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   .
 *
 *
 *
 * 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.
 *
 */



/*                                              qpow
 *
 *      Power function check routine
 *
 *
 *
 * SYNOPSIS:
 *
 * int qpow( x, y, z );
 * QELT *x, *y, *z;
 *
 * qpow( x, y, z );
 *
 *
 *
 * DESCRIPTION:
 *
 * Computes x raised to the yth power.
 *
 *       y
 *      x  =  exp( y log(x) ).
 *
 */



/*  qprob.c  */
/* various probability integrals
 * computed via incomplete beta and gamma integrals
 */



/*                                                      qbdtr
 *
 *      Binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int qbdtr( k, n, p, y );
 * int k, n;
 * QELT *p, *y;
 *
 * qbdtr( k, n, p, y );
 *
 * DESCRIPTION:
 *
 * Returns (in y) 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.
 *
 */



/*                                                      qbdtrc
 *
 *      Complemented binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int qbdtrc( k, n, p, y );
```

```
 * int k, n;
 * QELT *p, *y;
 *
 * y = qbdtrc( k, n, p, y );
 *
 * 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.
 *
 */



/*                                                      qbdtri
 *
 *      Inverse binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int qbdtri( k, n, y, p );
 * int k, n;
 * QELT *p, *y;
 *
 * qbdtri( k, n, y, p );
 *
 * 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 ).
 *
 */



/*                                                      qchdtr
 *
 *      Chi-square distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int qchdtr( df, x, y );
 * QELT *df, *x, *y;
 *
 * qchdtr( df, x, y );
 *
 *
 *
 * 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.
 *
 */
```

```
/*                                                      qchdtc
 *
 *      Complemented Chi-square distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int qchdtc( df, x, y );
 * QELT df[], x[], y[];
 *
 * qchdtc( df, x, y );
 *
 *
 *
 * 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.
 *
 */



/*                                                      qchdti
 *
 *      Inverse of complemented Chi-square distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int qchdti( df, y, x );
 * QELT *df, *x, *y;
 *
 * qchdti( df, y, x );
 *
 *
 *
 *
 * 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
 *
 */



/*                                                      qfdtr
 *
 *      F distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int qfdtr( ia, ib, x, y );
 * int ia, ib;
 * QELT *x, *y;
 *
```

```
 * qfdtr( ia, ib, x, y );
 *
 * 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.
 *
 */



/*                                                      qfdtrc
 *
 *      Complemented F distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int qfdtrc( ia, ib, x, y );
 * int ia, ib;
 * QELT x[], y[];
 *
 * qfdtrc( ia, ib, x, y );
 *
 * 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) ).
 *
 */



/*                                                      qfdtri
 *
 *      Inverse of complemented F distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int qfdtri( ia, ib, y, x );
 * int ia, ib;
 * QELT x[], y[];
 *
 * qfdtri( ia, ib, y, x );
 *
 * 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)).
 *
 */
```

```
/*                                                      qgdtr
 *
 *      Gamma distribution function
 *
 *
 *
 * SYNOPSIS:
 *
 * int qgdtr( a, b, x, y );
 * QELT *a, *b, *x, *y;
 *
 * qgdtr( a, b, x, y );
 *
 *
 *
 * 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 ).
 *
 */



/*                                                      qgdtrc
 *
 *      Complemented gamma distribution function
 *
 *
 *
 * SYNOPSIS:
 *
 * int qgdtrc( a, b, x, y );
 * QELT *a, *b, *x, *y;
 *
 * qgdtrc( a, b, x, y );
 *
 *
 *
 * 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 ).
 *
 */



/*                                                      qnbdtr
 *
 *      Negative binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int qnbdtr( k, n, p, y );
 * int k, n;
 * QELT *p, *y;
 *
 * qnbdtr( k, n, p, y );
 *
 * 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.
 *
 */



/*                                                      qnbdtc
 *
 *      Complemented negative binomial distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int qnbdtc( k, n, p, y );
 * int k, n;
 * QELT *p, *y;
 *
 * qnbdtc( k, n, p, y );
 *
 * 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.
 *
 */



/*                                                      qpdtr
 *
 *      Poisson distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int qpdtr( k, m, y );
 * int k;
 * QELT *m, *y;
 *
 * qpdtr( k, m, y );
 *
 *
 *
 * 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.
 *
 */



/*                                                      qpdtrc
 *
 *      Complemented poisson distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int qpdtrc( k, m, y );
 * int k;
 * QELT *m, *y;
```

```
 *
 * qpdtrc( k, m, y );
 *
 *
 *
 * 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.
 *
 */



/*                                                      qpdtri
 *
 *      Inverse Poisson distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int qpdtri( k, y, m );
 * int k;
 * QELT *m, *y;
 *
 * qpdtri( k, y, m );
 *
 *
 *
 *
 * 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 ).
 *
 */



/*                                                      qpsi.c
 *      Psi (digamma) function check routine
 *
 *
 * SYNOPSIS:
 *
 * int qpsi( x, y );
 * QELT *x, *y;
 *
 * qpsi( x, y );
 *
 *
 * DESCRIPTION:
 *
 *              d      -
 *   psi(x)  =  -- ln | (x)
 *              dx
 *
 * is the logarithmic derivative of the gamma function.
 * For general positive x, the argument is made greater than 16
 * 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.
 *
 * psi(-x)  =  psi(x+1) + pi/tan(pi(x+1))
 */



/*                                                      qrand.c
 *
 *      Pseudorandom number generator
 *
 *
```

```
 *
 * SYNOPSIS:
 *
 * int qrand( q );
 * QELT q[NQ];
 *
 * qrand( q );
 *
 *
 *
 * DESCRIPTION:
 *
 * Yields a random number 1.0 <= q < 2.0.
 *
 * A three-generator congruential algorithm adapted from Brian
 * Wichmann and David Hill (BYTE magazine, March, 1987,
 * pp 127-8) is used to generate random 16-bit integers.
 * These are copied into the significand area to produce
 * a pseudorandom bit pattern.
 */



/*                                                      qshici.c
 *
 *      Hyperbolic sine and cosine integrals
 *
 *
 *
 * SYNOPSIS:
 *
 * int qshici( x, si, ci );
 * QELT *x, *si, *ci;
 *
 * qshici( x, si, ci );
 *
 *
 * DESCRIPTION:
 *
 *
 *                             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 power series are
 *
 *           inf        2n+1
 *            -        z
 * Shi(z)  =  >  --------------
 *            -  (2n+1) (2n+1)!
 *           n=0
 *
 *                              inf      2n
 *                               -      z
 * Chi(z)  =  eul +  ln(z)  +   >  -----------
 *                               -    2n (2n)!
 *                              n=1
 *
 * Asymptotically,
 *
 *
 *     -x                1   2!   3!
 * 2x e   Shi(x)  =  1 + - + -- + -- + ...
 *                       x    2    3
 *                           x    x
 *
 * ACCURACY:
 *
 * Series expansions are set to terminate at less than full
 * working precision.
 *
 */



/*                                                      qsici.c
 *      Sine and cosine integrals
 *
 *
 *
 * SYNOPSIS:
 *
 * int qsici( x, si, ci );
 * QELT *x, *si, *ci;
 *
 * qsici( 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 power series are
 *
 *          inf      n  2n+1
 *           -   (-1)  z
 * Si(z)  =  >  --------------
 *           -  (2n+1) (2n+1)!
 *          n=0
 *
 *                            inf      n  2n
 *                             -   (-1)  z
 * Ci(z)  =  eul +  ln(z)  +   >  -----------
 *                             -    2n (2n)!
 *                            n=1
 *
 * ACCURACY:
 *
 * Series expansions are set to terminate at less than full
 * working precision.
 *
 */



/*                                                      qsimq.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.
 *
 */



/*                                                      qsin.c
 *      Circular sine check routine
 *
 *
 *
```

```
 * SYNOPSIS:
 *
 * int qsin( x, y );
 * QELT *x, *y;
 *
 * qsin( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Range reduction is into intervals of pi/2.
 * Then
 *
 *               3    5    7
 *              z    z    z
 * sin(z) = z - -- + -- - -- + ...
 *              3!   5!   7!
 *
 */



/*                                                      qsindg.c
 *
 * sin, cos, tan in degrees
 */



/*                                                      qsinh.c
 *
 *      Hyperbolic sine check routine
 *
 *
 *
 * SYNOPSIS:
 *
 * int qsinh( x, y );
 * QELT *x, *y;
 *
 * qsinh( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * The range is partitioned into two segments.  If |x| <= 1/4,
 *
 *               3    5    7
 *              x    x    x
 * sinh(x) = x + -- + -- + -- + ...
 *              3!   5!   7!
 *
 * Otherwise the calculation is sinh(x) = ( exp(x) - exp(-x) )/2.
 *
 */



/*                                                      qspenc.c
 *
 *      Dilogarithm
 *
 *
 *
 * SYNOPSIS:
 *
 * int qspenc( x, y );
 * QELT *x, *y;
 *
 * qspenc( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Computes the integral
 *
 *                    x
 *                    -
 *                   | | log t
 * spence(x)  =  -   |   ----- dt
 *                 | |   t - 1
 *                  -
 *                  1
 *
 * for x >= 0.  A power series 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.
 *
 *
 *
 */



/*                                                      qsqrt.c
 *      Square root check routine
 *
 *
```

```
 *
 * SYNOPSIS:
 *
 * int qsqrt( x, y );
 * QELT *x, *y;
 *
 * qsqrt( x, y );
 *
 *
 *
 * 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 to converge to an accurate value.
 *
 */



/*      qsqrta.c                */
/* Square root check routine, done by long division. */
/* Copyright (C) 1984-1988 by Stephen L. Moshier. */



/*                                                      qstdtr.c
 *
 *      Student's t distribution
 *
 *
 *
 * SYNOPSIS:
 *
 * int qstudt( k, t, y );
 * int k;
 * QELT *t, *y;
 *
 * qstudt( k, t, y );
 *
 *
 * 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:
 *
 */



/*                                                      qtan.c
 *      Circular tangent check routine
 *
 *
 *
 * SYNOPSIS:
 *
 * int qtan( x, y );
 * QELT *x, *y;
 *
 * qtan( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Domain of approximation is reduced by the transformation
 *
 * x -> x - pi floor((x + pi/2)/pi)
 *
 *
```

```
 * then tan(x) is the continued fraction
 *
 *                  2   2   2
 *             x   x   x   x
 * tan(x)  =  --- --- --- --- ...
 *            1 - 3 - 5 - 7 -
 *
 */



/*                                                      qcot
 *
 *      Circular cotangent check routine
 *
 *
 *
 * SYNOPSIS:
 *
 * int qcot( x, y );
 * QELT *x, *y;
 *
 * qcot( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * cot (x) = 1 / tan (x).
 *
 */



/*                                                      qtanh.c
 *      Hyperbolic tangent check routine
 *
 *
 *
 *
 * SYNOPSIS:
 *
 * int qtanh( x, y );
 * QELT *x, *y;
 *
 * qtanh( x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * For x >= 1 the program uses the definition
 *
 *             exp(x) - exp(-x)
 * tanh(x)  =  ----------------
 *             exp(x) + exp(-x)
 *
 *
 * For x < 1 the method is a continued fraction
 *
 *                   2   2   2
 *              x   x   x   x
 * tanh(x)  =  --- --- --- --- ...
 *              1+  3+  5+  7+
 *
 */



/*                                                      qyn.c
 *
 *      Real bessel function of second kind and general order.
 *
 *
 *
 * SYNOPSIS:
 *
 * int qyn( v, x, y );
 * QELT *v, *x, *y;
 *
 * qyn( v, x, y );
 *
 *
 *
 * DESCRIPTION:
 *
 * Returns Bessel function of order v.
 * If v is not an integer, the result is
 *
 *    Y (z) = ( cos(pi v) * J (x) - J  (x) )/sin(pi v)
 *     v                     v       -v
 *
 * Hankel's expansion is used for large x:
 *
 * Y (z) = sqrt(2/(pi z)) (P sin w + Q cos w)
 *  v
 *
 * w = z - (.5 v + .25) pi
 *
 *
```

```
 *         (u-1)(u-9)   (u-1)(u-9)(u-25)(u-49)
 * P = 1 - ---------- + ---------------------- - ...
 *                 2                    4
 *          2! (8z)           4! (8z)
 *
 *
 *      (u-1)    (u-1)(u-9)(u-25)
 * Q =  -----  - ---------------- + ...
 *       8z                 3
 *                  3! (8z)
 *
 *         2
 *  u = 4 v
 *
 * (AMS55 #9.2.6).
 *
 *
 *                 -n   n-1
 *           -(z/2)      -  (n-k-1)!   2    k
 * Y (z)  =  -------     >  -------- (z / 4)   +  (2/pi) ln (z/2) J (z)
 *  n           pi       -     k!                                  n
 *                      k=0
 *
 *
 *                  n     inf                             2    k
 *             (z/2)       -                          (- z / 4)
 *           - ------  -   >  (psi(k+1) + psi(n+k+1)) ----------
 *               pi        -                           k!(n+k)!
 *                        k=0
 *
 *  (AMS55 #9.1.11).
 *
 * ACCURACY:
 *
 * Series expansions are set to terminate at less than full working
 * precision.
 *
 */



/*                                                      qzetac.c
 *
 *      Riemann zeta function
 *
 *
 *
 * SYNOPSIS:
 *
 * int qzetac( x, y );
 * QELT *x, *y;
 *
 * qzetac( x, y );
 *
 *
 *
 * 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.
 *
 *
 * ACCURACY:
 *
 * Series summation terminates at NBITS/2.
 *
 */
```

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

Last update: 31 July 2000