```
/*
2016-12-14, Stig Rosenlund.
C functions for several mathematical-statistical distribution functions (CDF:s),
based on the beta distribution (incomplete beta function), and their inverses.
A carefully designed program for log(Gamma(x)) is included. The functions are
designed for maximal execution speed while giving double accuracy to the
returns. I believe that my programs add value in addition to existing programs,
such as, for the t-distribution, one by Ross Ihaka (R Development Core Team) and
the Medical College of Wisconsin, 1994-2000, released under Gnu.

If you find my programs useful, then it would nice if you mention my name while
using them or modifications of them.

About me. Born 1942 in Sweden where I still live. Ph. D. in mathematical
statistics. I have published many articles in peer-reviewed scientific journals
in this area, the first one in 1972 and the latest one in 2014. I have been
programming for 43 years, of which in C (among other languages) for 25 years.
The functions below, which I have decided to give away for free, are part of
the source code for my language Rapp for actuarial mathematics, mathematical
statistics and mathematics in general, in that order. I have developed Rapp for
33 years, the first eight in other languages than C. Access Rapp by googling
free actuarial language. You get a hit in November 2016. If Google changes its
search criteria, then google Stig Rosenlund and bypass the artist with the same
name. The Rapp manual tells what kind of functions. Look in particular at Proc
Mbasic.

If you want access to all source code for Rapp, then please help me make Rapp
open source. It is now stuck in a review process for open source, and I find
all the legal details in it confusing. I am a mathematician and programmer, not
a lawyer. My aim with having the Rapp code as open source is to have a maximal
permissive license, so that everyone can freely use it, while at the same time
preventing commercial software companies from appropriating it. My only
requirement would be that my name is mentioned when the code is used. Contact
me by e-mail stig.ingvar.rosenlund@gmail.com.

Explanations
------------
Some functions with a 0 last in the name are included below, but not listed in
the synopsis following immediately here. They are first versions that have been
superseded by better versions. For example tdist0().

betadist(a,b,x,&logc,0)
  Returns beta function of x with parameters a, b. logC is set to the logarithm
  of a constant for the beta function. If logC is not needed, set the 4:th
  argument to 0. If value 1 is set for the 5:th argument, 1 - (function value)
  is returned. If x is close to 1, then betadist(a,b,x,0,1) will have better
  precision than 1 - betadist(a,b,x,0,0). Only the values 0 and 1 are valid for
  the 5:th argument complflg. It is a flag for the complement, ie
    (return value with complflg = 1) = 1 - (return value with complflg = 0).
  It is an argument also for several other functions.

betadistinv(a,b,p,0)
  Returns inverse beta function of p with parameters a, b. If value 1 is set for
  the 4:th argument, 1 - (function value) is returned.

betadistinv_ap(a,b,p)
  Auxiliary function for betadistinv(). Here _ap means approximation, ie the
  function gives an approximation of betadistinv() that is a start value for
  the exact solution. If maximal execution speed was not important, the
  function could be replaced by the value 0.5.

bindist(n,p,k,0)
  Returns binomial distribution function of k with probability p for success.
  Ie, if X is bin(n,p), then P(X <= k) = bindist(n,p,k).
  Value 1 for the 4:th argument means that 1 - (function value) is returned.
```

```
bindistinv(n,p,p0)
  Returns inverse binomial distribution with parameters as above, defined as
  min{k: P(X <= k) >= p0}.

fdist(x,nu1,nu2,0)
  Returns F distribution function of x with nu1 and nu2 degrees of freedom.
  If value 1 is set for the 4:th argument, 1 - (function value) is returned.

fdistinv(p,nu1,nu2)
  Returns inverse F distribution function of p with nu1 and nu2 degrees of
  freedom.

fdistinv_ap(p,nu1,nu2)
  Auxiliary function for betadistinv_ap().

loggam(x)
  Returns natural logarithm of Gamma(x).

ndistinv_ap(p)
  Auxiliary function for betadistinv_ap(), fdistinv_ap() and tdistinv_ap().

tdist(x,nu,0)
  Returns t distribution function of x with nu degrees of freedom. If value 1
  is set for the 3:th argument, 1 - (function value) is returned.

tdistinv(p,nu)
  Returns inverse t distribution function of p with nu degrees of freedom.

tdistinv_ap(p,nu)
  Auxiliary function for betadistinv().

Remark. C manuals usually strongly recommend that double constants be written
with a decimal point, eg 1.0 and not 1. I have however found that this has no
effect whatsoever on the speed of the compiled and linked code, at least if you
use Microsoft's free compiler and linker. And for Windows I have found that these
are the best, better than the ones of Intel C and other products, free or for
purchase. If you work in other environments, you might want to put in decimal
points, though. (Of course I write eg 1.0/2 if 0.5 is intended and not 0. And
eg 123456789012.0 if an integer value is intended as a double and too large for
the type int. If an __int64 is intended, I write (__int64)123456789012.)
*/
#include <ctype.h>
#include <math.h>
#include <signal.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#define MAXIT 1000000
#define PI       3.14159265358979323846
#define PI2      6.28318530717958647692
#define SQRPI    1.772453850905516
#define SQR2PI   2.50662827463100050242
#define LOGSQ2PI 0.91893853320467273
#define absb(x) ((x) < 0.0 ? -(x) : (x))
double betadist(double,double,double,double *,unsigned char) ;
double betadistinv(double,double,double,unsigned char) ;
double betadistinv_ap(double,double,double) ;
double bindist(int,double,int,unsigned char) ;
int bindistinv(int,double,double) ;
double fdist(double,int,int,unsigned char) ;
double fdistinv(double,int,int) ;
double fdistinv_ap(double,int,int) ;
double loggam(double) ;
double ndistinv_ap(double) ;
double tdist0(double,int,unsigned char) ;
double tdist(double,int,unsigned char) ;
```

```
double tdistinv0(double,int) ;    // A first variant I made.
double tdistinv(double,int) ;     // A second and mostly slightly better variant.
double tdistinv_ap0(double,int) ; // A first variant.
double tdistinv_ap(double,int) ;  // A second and better variant.

void main(int argc, char *argv[]) {
  int k,k0,k1,n,nu,nu1,nu2;
  double a,b,f,p,p0,x,y1,y2;

  if (argc < 4) exit(0);
  a = atof(argv[1]);
  b = atof(argv[2]);
  p = atof(argv[3]);
  x = betadistinv(a,b,p,0);
  printf("Inversebeta(%1.15f,%1.15f,%1.15f) = %1.15f\n",a,b,p,x);
  y1 = betadist(a,b,x,0,0);
  printf("       Beta(%1.15f,%1.15f,%1.15f) = %1.15f\n",a,b,x,y1);

  if (argc > 4) {
    nu = atoi(argv[4]);
    y1 = tdist(a,nu,0);
    printf("   tdist(%1.15f,%d) = %1.15f\n",a,nu,y1);
    y2 = tdist(a,nu,1);
    printf(" 1-tdist(%1.15f,%d) = %1.15f\n",a,nu,y2);
    y2 = tdistinv(y1,nu);
    printf("tdistinv(%1.15f,%d) = %1.15f\n",y1,nu,y2);
  }

  f = a;
  nu1 = (int)b;
  nu2 = (int)p; // To make sense p has to be > 1 resulting in nonsense above.
  y1 = fdist(f,nu1,nu2,0);
  printf("  fdist(%1.15f,%d,%d) = %1.15f\n",f,nu1,nu2,y1);
  y2 = fdist(f,nu1,nu2,1);
  printf("1-fdist(%1.15f,%d,%d) = %1.15f\n",f,nu1,nu2,y2);
  y2 = fdistinv(y1,nu1,nu2);
  printf("fdistinv(%1.15f,%d,%d) = %1.15f\n",y1,nu1,nu2,y2);

  n = (int)a;
  k = (int)b;
  y1 = bindist(n,p,k,0);
  printf("  bindist(%d,%1.15f,%d) = %1.15lE\n",n,p,k,y1);
  printf("1-bindist(%d,%1.15f,%d) = %1.15lE\n",n,p,k,bindist(n,p,k,1));

  p0 = y1 - 5e-16;
  k0 = bindistinv(n,p,p0);
  printf("bindistinv(%d,%1.15f,%1.15lE) = %d =? %d\n",n,p,p0,k0,k);
  p0 = y1 + 5e-16;
  k0 = bindistinv(n,p,p0);
  if (k >= n) k1 = n; else k1 = k + 1;
  printf("bindistinv(%d,%1.15f,%1.15lE) = %d =? %d\n",n,p,p0,k0,k1);

  // Free allocations if the function is not used any more, as a matter of order.
  betadist(0,0,-1,0,0);
}

double betadist(double a0, double b0, double x_in,
    double *plogC, unsigned char complflg) {
  /* complflg = 1 means that 1 - result shall be returned.
  Beta function adapted from "Numerical Recipes in C", §6.4, p. 226 ff.
  One adaption is that only one function is used instead of the two
  functions betai() and betacf() of Numerical Recipes. This makes it
  possible to have the relative error the same, which is not the case
  when Numerical Recipes uses 1 - betacf(b,a,1-x)/b. Another adaption is
  that two sets of parameters for (a,b) and (b,a) are stored statically,
  and the fastest one is used by determining the parameter symmetryuse.
```

```
The static storing of the coefficients coeffev[] for even steps and
coeffod[] for odd steps is beneficial, provided the function is used
more than a few times with (a,b) or (b,a) the same as the (a,b) of the
previous call to the function. And this is the case for the inversion
algorithm. The larger time used when this is not so is neglibible. */
typedef struct {
  int lastinxs,noalls;
  double *coeffevs,*coeffods,Ds,as,bs,qabByqaps,qams,qaps;
}
ABPARMS;
static ABPARMS *abparms1,*abparms2,*abparms;
static unsigned char funcused=0;
unsigned char epsilonchanged,symmetryuse,whichmatches,which2use;
int m,m2;
static double *coeffev,*coeffod,fpmin=1e-300,qab,logC;
// epsilon cannot be static since it is sometimes changed.
double bt,btbya,c,d,d1,d2,delta,epsilon=5e-16,h,x;
#define a        (abparms->as)
#define b        (abparms->bs)
#define qap      (abparms->qaps)
#define qam      (abparms->qams)
#define qabByqap (abparms->qabByqaps)
#define noall    (abparms->noalls)
#define lastinx  (abparms->lastinxs)

if (x_in == -1) { // Argument -1 is signal to free allocations.
  if (funcused) {
    funcused = 0;
    free(abparms1->coeffevs);
    free(abparms1->coeffods);
    free(abparms2->coeffevs);
    free(abparms2->coeffods);
    free(abparms1);
    free(abparms2);
  }
  return 0;
}
// 1 - I_x(a,b) = I_{1-x}(b,a)
if (complflg) return betadist(b0,a0,1-x_in,plogC,0);
if (x_in <= 0 || a0 <= 0 || b0 <= 0) return 0;
if (x_in >= 1) return 1;
if (!funcused) {
  funcused = 1;
  abparms = 0;
  abparms1 = malloc(sizeof(ABPARMS));
  abparms2 = malloc(sizeof(ABPARMS));
  abparms1->coeffevs = malloc(sizeof(double));
  abparms1->coeffods = malloc(sizeof(double));
  abparms2->coeffevs = malloc(sizeof(double));
  abparms2->coeffods = malloc(sizeof(double));
  abparms1->noalls = abparms2->noalls = 1;
  abparms1->as = abparms2->as = abparms1->bs = abparms2->bs = 0;
}
if      (a0 == abparms1->as && b0 == abparms1->bs) whichmatches = 1;
else if (a0 == abparms2->as && b0 == abparms2->bs) whichmatches = 2;
else {
  whichmatches = 0;
  abparms = 0;
  qab = a0 + b0;
  logC = loggam(qab) - loggam(a0) - loggam(b0);

  abparms1->as = a0;
  abparms1->bs = b0;
  abparms1->qaps = a0 + 1;
  abparms1->qams = a0 - 1;
  abparms1->qabByqaps = qab/abparms1->qaps;
```

```
  abparms1->Ds = abparms1->qaps/(qab + 2); // (a0+1)/(a0+ba0+2)
  abparms1->lastinxs = 0;

  abparms2->as = b0;
  abparms2->bs = a0;
  abparms2->qaps = b0 + 1;
  abparms2->qams = b0 - 1;
  abparms2->qabByqaps = qab/abparms2->qaps;
  abparms2->Ds = abparms2->qaps/(qab + 2); // (b0+1)/(a0+ba0+2)
  abparms2->lastinxs = 0;
}
if (plogC != NULL) *plogC = logC;
if (whichmatches == 0 || whichmatches == 1) {
  if (x_in < abparms1->Ds) { symmetryuse = 0; which2use = 1; }
  else                     { symmetryuse = 1; which2use = 2; }
}
else if (whichmatches == 2) {
  if (x_in < abparms2->Ds) { symmetryuse = 0; which2use = 2; }
  else                     { symmetryuse = 1; which2use = 1; }
}
// The following statements' purpose is to save CPU by only setting
// coeffev and coeffod if they were not set right the previous call.
// The saving is of course marginal.
if (which2use == 1) {
  if (abparms != abparms1) {
    abparms = abparms1;
    coeffev = abparms1->coeffevs - 1;
    coeffod = abparms1->coeffods - 1;
  }
}
else {
  if (abparms != abparms2) {
    abparms = abparms2;
    coeffev = abparms2->coeffevs - 1;
    coeffod = abparms2->coeffods - 1;
  }
}
if (symmetryuse == 0) x = x_in; else x = 1 - x_in;
bt = exp(logC + a*log(x) + b*log(1-x));
epsilonchanged = 0;
btbya = bt/a;
/*
Below delta = h(j)/h(j-1) and delta - 1 = [h(j) - h(j-1)]/h(j-1).
If symmetryuse = 0, then |delta-1| < epsilon ==> |h(j)-h(j-1)| < epsilon*h(j-1),
so that that relative error epsilon is obtained since return = h(j)*bt/a.
If symmetryuse = 1, then return = 1-h(j)*bt/a. Put btbya = bt/a. We want then
  |[1-btbya*h(j)]-[1-btbya*h(j-1)]| < epsilon*|1-btbya*h(j-1)| ie
   btbya*|h(j)-h(j-1)| < epsilon*|1-btbya*h(j-1)|, ie
  |h(j)-h(j-1)|/h(j-1) <  epsilon*|1/[btbya*h(j-1)]-1| ie
  |delta-1| < epsilon*|1/[btbya*h(j-1)]-1|.
At the first occasion when |delta - 1| < epsilon, epsilon is recomputed to
epsilon*|1/[btbya*h(j-1)]-1|. One might as well use h(j) instead of h(j-1),
ie epsilon = epsilon*|1/(btbya*h)-1|. If then |delta-1| >= epsilon the execution
continues. A flag epsilonchanged is set so that recomputation is only made once.
*/
c = 1; // First step of Lentz's method.
d = 1 - qabByqap*x; // d = 1 - qab*x/qap
if (absb(d) < fpmin) d=fpmin;
d = 1/d;
h=d;
for (m=1; m<=lastinx; m++) {
  m2 = 2*m;
  d1 = coeffev[m]*x;
  d = 1 + d1*d; // One step (the even one) of the recurrence.
  if (absb(d) < fpmin) d=fpmin;
  c = 1 + d1/c;
```

```
    if (absb(c) < fpmin) c=fpmin;
    d = 1/d;
    h *= d*c;
    d1 = coeffod[m]*x;
    d = 1 + d1*d; // Next step of the recurrence (the odd one).
    if (absb(d) < fpmin) d=fpmin;
    c = 1 + d1/c;
    if (absb(c) < fpmin) c=fpmin;
    d = 1/d;
    delta = d*c;
    h *= delta;
    d2 = delta - 1;
    if (d2 < 0) d2 = -d2;
    if (d2 < epsilon) {
      if (epsilonchanged == 0 && symmetryuse == 1) {
        d1 = 1/(btbya*h) - 1;
        if (d1 < 0) d1 = -d1;
        if (d1 > 0) epsilon *= d1;
        if (d2 < epsilon) goto ENDCOMPUTE;
        epsilonchanged = 1;
        continue;
      }
      goto ENDCOMPUTE;
    }
  }
  for ( ; m<=MAXIT; m++) {
    if (m > noall) {
      noall += 20;
      if (which2use == 1) {
        abparms1->coeffevs = realloc(abparms1->coeffevs,noall*sizeof(double));
        abparms1->coeffods = realloc(abparms1->coeffods,noall*sizeof(double));
      }
      else {
        abparms2->coeffevs = realloc(abparms2->coeffevs,noall*sizeof(double));
        abparms2->coeffods = realloc(abparms2->coeffods,noall*sizeof(double));
      }
      coeffev = abparms->coeffevs - 1;
      coeffod = abparms->coeffods - 1;
    }
    m2=2*m;
    coeffev[m] = m*(b-m)/((qam+m2)*(a+m2));
    d1 = coeffev[m]*x;
    d = 1 + d1*d;
    if (absb(d) < fpmin) d=fpmin;
    c = 1 + d1/c;
    if (absb(c) < fpmin) c=fpmin;
    d = 1/d;
    h *= d*c;
    coeffod[m] = -(a+m)*(qab+m)/((a+m2)*(qap+m2));
    d1 = coeffod[m]*x;
    d = 1 + d1*d;
    if (absb(d) < fpmin) d=fpmin;
    c = 1 + d1/c;
    if (absb(c) < fpmin) c=fpmin;
    d = 1/d;
    delta = d*c;
    h *= delta;
    d2 = delta - 1;
    if (d2 < 0) d2 = -d2;
    if (d2 < epsilon) {
      lastinx = m;
      if (epsilonchanged == 0 && symmetryuse == 1) {
        d1 = 1/(btbya*h) - 1;
        if (d1 < 0) d1 = -d1;
        if (d1 > 0) epsilon *= d1;
        if (d2 < epsilon) goto ENDCOMPUTE;
```

```
        epsilonchanged = 1;
        continue;
      }
      goto ENDCOMPUTE;
    }
  }
  printf("Maxno iterations exceeded in betadist().\n");
  exit(0);
  ENDCOMPUTE:
  if (symmetryuse == 0) return btbya*h;
  return 1 - btbya*h;
}

#undef a
#undef b
#undef qap
#undef qam
#undef qabByqap
#undef noall
#undef lastinx

double betadistinv(double a, double b, double p, unsigned char complflg) {
  // Beta function inverse. complflg = 1 means that 1 - result is returned.
  static int ninvqua=100;
  int j;
  static double epsilon0=5e-16;
  double Cinv,am1,bm1,d1,diffappr,diffupperlower,epsilon,h1,h2,
    lower=0,p0,p0old,step,twomab,upper=1,y0,y1;
  if (complflg) return betadistinv(b,a,1-p,0);
  if (p <= 0 || a <= 0 || b <= 0) return 0;
  if (p >= 1) return 1;
  twomab = 2 - a - b;
  am1 = a - 1;
  bm1 = b - 1;
  y0 = betadistinv_ap(a,b,p); if (y0 <= 0) y0 = 1e-300; // 2016-12-04 correction.
  p0 = betadist(a,b,y0,&Cinv,0);
  if (p0 == p) return y0;
  Cinv = 1/exp(Cinv); // 1/exp(loggam(a+b) - loggam(a) - loggam(b))
  if (p0 > p) upper = y0;
  else        lower = y0;
  diffappr = diffupperlower = upper - lower;
  p0old = p0;
  // Inverse quadratic search.
  for (j=1; j<=ninvqua; j++) {
    d1 = 1 - y0;
    h1 = exp(am1*log(y0) + bm1*log(d1)); // y0^{a-1}*(1-y0)^{b-1}
    h2 = h1*(twomab*y0+am1)/(y0*d1);
    h1 = 1/h1;
    h2 = -h2*h1*h1*h1;
    d1 = Cinv*(p - p0);
    y1 = y0 + d1*(h1 + 0.5*h2*d1);
    if (isnan(y1) || y1 <= 0 || y1 >= 1) { epsilon = y0*epsilon0; break; }
    d1 = y1 - y0; if (d1 < 0) d1 = -d1; // New difference approximation.
    // If d1 > 0.5*diffappr the procedure is now worse
    // than bisection or is even beginning to diverge.
    if (d1 > 0.5*diffappr) { epsilon = y0*epsilon0; break; }
    diffappr = d1;
    p0 = betadist(a,b,y1,0,0);
    // If this condition holds we have deficient accuracy, so no use to continue.
    if ((p0 >= p0old && y1 <= y0) || (p0 <= p0old && y1 >= y0)) return y0;
    y0 = y1;
    if (p0 == p) return y0;
    if (p0 > p) { if (upper > y0) upper = y0; }
    else        { if (lower < y0) lower = y0; }
    diffupperlower = upper - lower;
    epsilon = y0*epsilon0; // Recompute epsilon.
```

```
    if (diffupperlower < epsilon) return y0;
    if (diffappr < epsilon) break;
    p0old = p0;
  }
  if (diffappr > diffupperlower) diffappr = diffupperlower;
  /* Now we have a solution estimate y0 by inverse quadratic search and an
  estimate diffappr of the difference between it and the true solution. If
  diffappr is small, often the estimate will be very close to the true solution,
  so a small step in stepping down or up will likely give new better limits
  lower and upper quickly. A little more than half diffappr seems optimal.
  Inverse quadratic search can fail, and probably will if a good initial
  approximation p0 is not found. Therefore the following steps are needed to
  secure a solution within epsilon. */
  step = 0.50001*diffappr;
  if (step < epsilon) step = epsilon;
  p0old = p0;
  if (lower != y0) { // Here y0 = upper.
    for (d1=y0-step,j=1; d1>lower; d1-=step,j++) {
      p0 = betadist(a,b,d1,0,0);
      // if p0 >= p0old we have deficient accuracy, so no use to continue.
      if (p0 >= p0old) return d1+step;
      if (p0 == p) return d1;
      if (p0 < p) { lower = d1; break; }
      upper = d1;
      if (j == 4) break;
      p0old = p0;
    }
  }
  else if (upper != y0) { // Here y0 = lower.
    for (d1=y0+step,j=1; d1<upper; d1+=step,j++) {
      p0 = betadist(a,b,d1,0,0);
      if (p0 <= p0old) return d1-step; // Deficient accuracy.
      if (p0 == p) return d1;
      if (p0 > p) { upper = d1; break; }
      lower = d1;
      if (j == 4) break;
      p0old = p0;
    }
  }
  if (upper - lower < epsilon) return 0.5*(lower+upper);
  // Bisection, ie interval halvings, as final step if needed.
  h2 = 1e300;
  for (;;) {
    d1 = 0.5*(lower+upper);
    p0 = betadist(a,b,d1,0,0);
    if (p0 == p) return d1;
    if (p0 > p) upper = d1; else lower = d1;
    h1 = upper - lower;
    if (h1 < epsilon || h1 >= h2) break; // If h1 gets stuck at values > epsilon.
    h2 = h1;
  }
  return 0.5*(lower+upper);
}

double betadistinv_ap(double a, double b, double p) {
  // Method 0. Abramowitz & Stegun (26.7.5), page 949, t-distribution inverse.
  // Method 1. Abramowitz & Stegun (26.5.22) page 945. Approximation of beta
  // function inverse. Cannot obviously be used for a or b <= 0.5.
  // Method 2. Approximation derived from the Abramowitz & Stegun (26.6.14),
  // page 947, approximation of the F-distribution function.
  unsigned char method;
  static double Fivebysix=0.8333333333333333,
    c0=2.515517,c1=0.802853,c2=0.010328,d1=1.432788,d2=0.189269,d3=0.001308;
  double h,lambda,r1,r2,y,w;
  if (p <= 0) return 0;
  if (p >= 1) return 1;
```

```
  if (p > 0.5) return 1 - betadistinv_ap(b,a,1-p);
  method = 1;
  if ((r1=2*a) == floor(r1) && (r2=2*b) == floor(r2)) {
    // If one of a and b is 0.5 and the other one is a multiple of 0.5,
    // use Abramowitz & Stegun (26.7.5), page 949, t-distribution inverse,
    // mapping beta function parameters to t-distribution parameters.
    if (b == 0.5) {
      h = tdistinv_ap(1-0.5*p,r1);
      h = r1/(h*h+r1);
      if (isnan(h)) h = 0;
      return h;
    }
    if (a == 0.5) {
      h = tdistinv_ap(0.5*(1-p),r2);
      h = h*h;
      h = h/(h+r2);
      if (isnan(h)) h = 0;
      return h;
    }
    // It is possible to use Abramowitz & Stegun (26.6.14). These criteria
    // for approximative optimality have been established by experimentation.
    if (a == 1) {
      if (b == 1) { if (p >= 0.26) method = 2; }
      else        { if (p >= 0.17) method = 2; }
    }
    else if (a == 1.5) {
      if      (b == 1.0) { if (p >= 0.10) method = 2; }
      else if (b == 1.5) { if (p >= 0.35) method = 2; else method = 1; }
      else               { if (p >= 0.22) method = 2; }
    }
    else if (a == 2.0) {
      if (b == 1) { if (p >= 0.05) method = 2; }
      else        { if (p >= 0.23) method = 2; }
    }
    else if (a == 2.5) { if (b == 1) { if (p >=   0.03) method = 2; } }
    else if (a == 3.0) { if (b == 1) { if (p >=   0.02) method = 2; } }
    else if (a <= 5.5) { if (b == 1) { if (p >=  0.001) method = 2; } }
    else if (a <= 6.5) { if (b == 1) { if (p >= 0.0001) method = 2; } }
    else { if (b == 1) method = 2; }
  }
  if (method == 1) {
    if (a < 0.8 || b < 0.8) h = 0.5;
    else {
      y = ndistinv_ap(1-p);
      r1 = 1/(2*a-1);
      r2 = 1/(2*b-1);
      h = 2/(r1+r2);
      lambda = (y*y-3)/6;
      w = y*sqrt(h+lambda)/h -(r2-r1)*(lambda+Fivebysix-2/(3*h));
      h = a/(a+b*exp(2*w));
      if (isnan(h) || h <= 0 || h >= 1) h = 0.5;
    }
    return h;
  }
  h = fdistinv_ap(1-p,r2,r1);
  return r1/(r1+r2*h);
}

double bindist(int n, double p, int k, unsigned char complflg) {
  // Binomial distribution of X for n trials with probability
  // p for success. P(X <= k) = bindist(n,p,k).
  // complflg = 1 means that 1 - result shall be returned.
//if (n <= 0 || k <= 0 || p < 0 || p > 1) return 0; 2017-05-08 correction.
  if (n <= 0 || p < 0 || p > 1) return 0; // Bad arguments.
  if (k < 0) { if (complflg) return 1; return 0; }
  if (p == 0) return complflg;
```

```
  if (p == 1 || k >= n) return 1 - complflg;
  return betadist(n-k,k+1,1-p,0,complflg);
}

int bindistinv(int n, double p, double p0) {
  /* Inverse of binomial distribution with parameters as above.
     Returns min{k: P(X <= k) >= p0}.
     As n grows the distribution of (X - E[X])/sqrt(Var[X]) converges
     to the normal distribution Phi() with mean 0 and variance 1.
     So an approximation of E[X] + sqrt(Var[X])*Phi^{-1}(p0) is taken as
     start value from which we step up or down until the solution is found.
     Now E[X] = n*p and Var[X] = n*p*(1-p). We take ndistinv_ap(p0) as an
     approximation of Phi^{-1}(p0). */
  int k;
  double q,p1;
  if (n <= 0 || p0 <= 0 || p <= 0) return 0;
  if (p0 >= 1 || p >= 1) return n;
  q = 1 - p;
  p1 = n*p;
  k = (int)(p1 + sqrt(p1*q)*ndistinv_ap(p0));
  if (k <= 0) k = 0; else if (k > n) k = n;
  p1 = betadist(n-k,k+1,q,0,0);
  if (p1 == p0) return k;
  if (p1 < p0) {
    for (k++; k<=n; k++) if (betadist(n-k,k+1,q,0,0) >= p0) return k;
    return n;
  }
  for (k--; k>=0; k--) {
    p1 = betadist(n-k,k+1,q,0,0);
    if (p1 == p0) return k;
    if (p1 < p0) break;
  }
  return k + 1;
}

double fdist(double f, int nu1, int nu2, unsigned char complflg) {
  // F-distribution with nu1 and nu2 degrees of freedom.
  // complflg = 1 means that 1 - result shall be returned.
  if (nu1 <= 0 || nu2 <= 0) return 0; // Bad arguments.
  if (f <= 0) return complflg;
  return betadist(0.5*nu2,0.5*nu1,nu2/(nu2+nu1*f),0,1-complflg);
}

double fdistinv(double p, int nu1, int nu2) {
  // F-distribution inverse with nu1 and nu2 degrees of freedom.
  double d1;
  if (nu1 <= 0 || nu2 <= 0 || p <= 0) return 0; // Bad arguments.
  d1 = betadistinv(0.5*nu1,0.5*nu2,p,0);
  if (d1 >= 1) return 1e30;
  return d1*nu2/(nu1*(1-d1));
}

double fdistinv_ap(double p, int nu1, int nu2) {
  int c1,c2;
  double d1,d2,d3,d4,h,w;
  if (nu1 <= 0 || nu2 <= 0 || p <= 0) return 0; // Bad arguments.
  if (p >= 1) return 1;
  if (p > 0.5) return 1/fdistinv_ap(1-p,nu2,nu1);
  w = ndistinv_ap(p);
  c1 = 2*nu1 - 1;
  c2 = 2*nu2 - 1;
  d1 = c2 - w*w;
  d2 = c1 + d1;
  d3 = sqrt((double)c1*c2);
  d4 = w*sqrt(d2);
  h = (d3 + d4)/d1;
```

```
  h = h*h;
  h *= (double)nu2/nu1;
  if (isnan(h) || h < 0) return 0;
  if (h >= 1) return 1;
  return h;
}

double loggam(double x) {
  // Returns the natural logarithm of Gamma(x). The function is quite different
  // from gammln() in "Numerical Recipes in C" based on Lanczo's results. I have
  // tried the latter and found that it lacks precision. Maybe some newer version
  // in C++ is better? See Abramowitz o Stegun page 257, 6.1.40 and 6.1.41. I have
  // deduced ten coefficients for the sum in 6.1.40, while 6.1.41 has four,
  // thereby gaining needed precision.
  // loggam_for_multiples_of_onehalf[i] = loggam(0.5*i) is computed initially,
  // with some redundancy (for simplicity), since for i = 0, 2, 4 it is 0. The
  // purpose is to speed up the return of loggam(0.5*i) for moderate values of i.
  static unsigned char first=1,multiples_of_onehalf_can_be_used=0;
  int i;
  double d01,d02,d03,d04,d05,d07,d09,d11,d13,d15,d17,d19;
  static double loggam_for_multiples_of_onehalf[1001],logPI,
    c01 =  1.0/12.0,
    c02 = -1.0/360.0,
    c03 =  1.0/1260.0,
    c04 = -1.0/1680.0,
    c05 =  1.0/1188.0,
    c06 = -691.0/360360.0,
    c07 =  1.0/156.0,
    c08 = -3617.0/122400.0,
    c09 =  43867.0/244188.0,
    c10 = -174611.0/125400.0;
  if (first) { // Compute only once.
    first = 0;
    logPI = log(PI);
    for (i=1; i<=1000; i++) loggam_for_multiples_of_onehalf[i] = loggam(0.5*i);
    multiples_of_onehalf_can_be_used = 1;
  }
  if (x == -1 || x == 0 || x == 1 || x == 2) return 0;
  // Reflection formula. Returns log(|PI/(sin(PI*x)*Gamma(1-x))|
  // = log(PI/(|sin(PI*x)|*Gamma(1-x))
  if (x < 0) {
    if (x == floor(x)) return 0;
    // Reduction with the right multiple of 2*Pi to avoid numerical errors in sin().
    d01 = -x;
    i = ((int)d01)%2;
    d01 = -sin(PI*(i+d01-floor(d01))); // sin(PI*x)
    if (d01 == 0) return 0;
    if (d01 < 0) d01 = -d01;
    return logPI - log(d01) - loggam(1-x);
  }
  if (multiples_of_onehalf_can_be_used && 2*x == floor(2*x) && (i=2*x) <= 1000)
    return loggam_for_multiples_of_onehalf[i];
  if (x >= 5) {
    d01 = 1/x;
    d02 = d01*d01; // 1/x^2
    d03 = d02*d01; // 1/x^3
    d04 = d02*d02; // 1/x^4
    d05 = d03*d02; // 1/x^5
    d07 = d03*d04; // 1/x^7
    d09 = d05*d04; // 1/x^9
    d11 = d07*d04; // 1/x^11
    d13 = d09*d04; // 1/x^13
    d15 = d11*d04; // 1/x^15
    d17 = d13*d04; // 1/x^17
    d19 = d15*d04; // 1/x^19
    d01 = ((((((((c10*d19 + c09*d17) + c08*d15) + c07*d13) + c06*d11)
```

```
          + c05*d09) + c04*d07) + c03*d05) + c02*d03) + c01*d01;
    return ((LOGSQ2PI + d01) - x) + (x-0.5)*log(x);
  }
  // 2016-04-15. Taylor expansion around 1 och 2.
  if (0.99999765 <= x && x <= 1.00007966) {
    double psi0=-0.5772156649015329,psi1=1.644934066848226,psi2=-2.404113806319189;
    d01 = x - 1; d02 = d01*d01; d03 = d02*d01;
    return psi0*d01 + psi1*d02/2 + psi2*d03/6;
  }
  if (1.99999955 <= x && x <= 2.00035646) {
    double psi0=0.4227843350984671,psi1=0.6449340668482264,psi2=-0.4041138063191886;
    d01 = x - 2; d02 = d01*d01; d03 = d02*d01;
    return psi0*d01 + psi1*d02/2 + psi2*d03/6;
  }
  { // If x < 5 the computation is referred to x >= 5 via gamma(x+1) = x*gamma(x)
    // several times. See Abramowitz o Stegun page 256, 6.1.15.
    double x2=x+5;
    d01 = 1/x2;
    d02 = d01*d01;
    d03 = d02*d01;
    d04 = d02*d02;
    d05 = d03*d02;
    d07 = d03*d04;
    d09 = d05*d04;
    d11 = d07*d04;
    d13 = d09*d04;
    d15 = d11*d04;
    d17 = d13*d04;
    d19 = d15*d04;
    d01 = (((((((c10*d19 + c09*d17) + c08*d15) + c07*d13) + c06*d11)
          + c05*d09) + c04*d07) + c03*d05) + c02*d03) + c01*d01;
    return ((LOGSQ2PI + d01) - x2) + (x2-0.5)*log(x2)
      - log(x*(x+1)*(x+2)*(x+3)*(x+4));
  }
}

double ndistinv_ap(double p) {
  // Algorithm by Boris Moro and Peter J. Acklam, latest version 2004-05-04.
  // Approximates normal distribution function inverse with error < 1.15E-9.
  double q,r,x;
  static double
    a1 = -3.969683028665376e+1,
    a2 =  2.209460984245205e+2,
    a3 = -2.759285104469687e+2,
    a4 =  1.383577518672690e+2,
    a5 = -3.066479806614716e+1,
    a6 =  2.506628277459239,

    b1 = -5.447609879822406e+1,
    b2 =  1.615858368580409e+2,
    b3 = -1.556989798598866e+2,
    b4 =  6.680131188771972e+1,
    b5 = -1.328068155288572e+1,

    c1 = -7.784894002430293e-3,
    c2 = -3.223964580411365e-1,
    c3 = -2.400758277161838,
    c4 = -2.549732539343734,
    c5 =  4.374664141464968,
    c6 =  2.938163982698783,

    d1 =  7.784695709041462e-3,
    d2 =  3.224671290700398e-1,
    d3 =  2.445134137142996,
    d4 =  3.754408661907416,
```

```
    p_low  = 0.02425,
    p_high = 0.97575;

  // Return = argument value if not 0 < p < 1.
  if (p <= 0 || p >= 1) return p;
  if (p == 0.5) return 0;
  if (p < p_low) {
    q = sqrt(-2.0*log(p));
    x = ((((c1*q+c2)*q+c3)*q+c4)*q+c5)*q+c6) /
        ((((d1*q+d2)*q+d3)*q+d4)*q+1);
  }
  else if (p <= p_high) {
    q = p - 0.5;
    r = q*q;
    x = ((((a1*r+a2)*r+a3)*r+a4)*r+a5)*r+a6)*q /
        ((((b1*r+b2)*r+b3)*r+b4)*r+b5)*r+1);
  }
  else {
    q = sqrt(-2.0*log(1-p));
    x = -((((c1*q+c2)*q+c3)*q+c4)*q+c5)*q+c6) /
         ((((d1*q+d2)*q+d3)*q+d4)*q+1);
  }
  return x;
  // Acklam's program has a final step with a call to a function
  // returning the normal distribution function, but the precision
  // increase from that is unnecessary in this context.
}

double tdist0(double t, int nu, unsigned char complflg) {
  // First version of Student's t-distribution with nu degrees of freedom.
  // complflg = 1 means that 1 - result shall be returned. Based on Abramowitz
  // & Stegun (26.7.3) and (26.7.4), page 948. A(t|nu) = 2*tdist(t,nu) - 1 and
  // tdist(t,nu) = (1 + A(t|nu))/2. It is fast but not very accurate, due to
  // rounding errors in the trigonometric functions.
  unsigned char negflg=0;
  int i1,n1;
  double d01,s01,t1,theta,sinth,costh,costh2;
  if (nu <= 0) return 0; // Bad argument.
  if (t == 0) return 0.5;
  if (t < 0) { negflg = 1; t1 = -t; } else  t1 = t;
  theta = atan(t1/sqrt(nu));
  sinth = sin(theta);
  costh = cos(theta);
  costh2 = costh*costh;
  n1 = nu - 2;
  if (nu%2 == 1) {
    s01 = 0;
    if (nu > 1) {
      for (i1=1,d01=costh; ; i1+=2) {
        s01 += d01;
        if (i1 == n1) break;
        d01 *= costh2*(i1+1)/(i1+2);
      }
    }
    s01 = (2/PI)*(theta + sinth*s01);
  }
  else {
    s01 = 1;
    if (nu > 2) {
      for (i1=2,d01=1; ; i1+=2) {
        d01 *= costh2*(i1-1)/i1;
        s01 += d01;
        if (i1 == n1) break;
      }
    }
    s01 = sinth*s01;
```

```
  }
  if ((!complflg && !negflg) || (complflg && negflg)) return (s01+1)/2;
  return (1-s01)/2;
}

double tdist(double t, int nu, unsigned char complflg) {
  // Same arguments and return as above. Slower but more accurate.
  double t2=t*t;
  if (nu <= 0) return 0; // Bad argument.
  if (t == 0) return 0.5;
  if ((complflg && t > 0) || (t < 0 && complflg == 0))
     return 0.5*betadist(0.5*nu,0.5,nu/(nu+t2),0,0);
  return 0.5 + 0.5*betadist(0.5,0.5*nu,t2/(nu+t2),0,0);
}

double tdistinv0(double p, int nu) {
  // First version of inverse of Student's t-distribution with nu degrees
  // of freedom. Short and fast formulas are available for nu = 1, 2, 4.
  double t;
  if (nu <= 0) return 0;
  if (p == 0.5) return 0;
  if (p <= 0) return -1e30;
  if (p >= 1) return  1e30;
  if (nu == 1) return tan(PI*(p-0.5));
  if (nu == 2) return (2*p-1)/sqrt(2*p*(1-p));
  if (p > 0.5) t = 1 - p; else t = p;
  if (nu == 4) {
    t = 2*sqrt(t*(1-t));
    t = (4/t)*cos(acos(t)/3);
    t = sqrt(t-4);
  }
  else {
    t = betadistinv(0.5*nu,0.5,2*t,0);
    t = sqrt(nu*(1-t)/t);
  }
  if (p < 0.5) return -t;
  return t;
}

double tdistinv(double p, int nu) {
  // Final version of inverse of Student's t-distribution with nu degrees
  // of freedom. Short and fast formulas are available for nu = 1, 2, 4.
  int j;
  static int ninvqua=100;
  static double epsilon0=5e-16;
  double C1inv,d1,dfinv,dfplus1by2,diffappr,diffupperlower,epsilon,h1,h2,
    lower,p0,p0old,px,step,upper=0,y0,y1;
  if (nu <= 0) return 0; // Bad argument.
  if (p == 0.5) return 0;
  if (p <= 0) return -1e30;
  if (p >= 1) return  1e30;
  if (nu == 1) return tan(PI*(p-0.5));
  if (nu == 2) return (2*p-1)/sqrt(2*p*(1-p));
  if (nu == 4) {
    y0 = 2*sqrt(p*(1-p));
    y0 = (4/y0)*cos(acos(y0)/3);
    y0 = -sqrt(y0-4);
    if (p > 0.5) return -y0;
    return y0;
  }
  if (p > 0.5) px = 1 - p; else px = p;
  lower = (2*px-1)/sqrt(2*p*(1-p));
  y0 = tdistinv_ap(px,nu);
  p0 = tdist(y0,nu,0);
  if (p0 == px) goto THEEND2;
  if (p0  > px) upper = y0;
```

```
else          lower = y0;
diffappr = diffupperlower = upper - lower;
dfinv = 1.0/nu;
d1 = 0.5*nu;
dfplus1by2 = 0.5+d1;
C1inv = exp(-loggam(dfplus1by2)+loggam(d1))*SQRPI*sqrt(nu);
epsilon = -y0*epsilon0; // Set limit for relative error.
p0old = p0;
// Inverse quadratic search.
for (j=1; j<=ninvqua; j++) {
  d1 = 1 + y0*y0*dfinv;
  h1 = C1inv*pow(d1,dfplus1by2);
  h2 = C1inv*C1inv*(nu+1)*dfinv*y0*pow(d1,nu);
  d1 = px - p0;
  y1 = y0 + h1*d1 + h2*d1*d1*0.5;
  if (isnan(y1) || y1 >= 0) break;
  d1 = y1 - y0; if (d1 < 0) d1 = -d1; // New difference approximation.
  // If d1 > 0.5*diffappr the procedure is now worse
  // than bisection or is even beginning to diverge.
  if (d1 > 0.5*diffappr) break;
  diffappr = d1;
  p0 = tdist(y1,nu,0);
  // If this condition holds we have deficient accuracy, so no use to continue.
  if ((p0 >= p0old && y1 <= y0) || (p0 <= p0old && y1 >= y0)) goto THEEND2;
  y0 = y1;
  if (p0 == px) goto THEEND2;
  if (p0 >  px) { if (upper > y0) upper = y0; }
  else          { if (lower < y0) lower = y0; }
  diffupperlower = upper - lower;
  if (diffupperlower < epsilon) goto THEEND2;
  if (diffappr < epsilon) break;
  p0old = p0;
}
if (diffappr > diffupperlower) diffappr = diffupperlower;
/* Now we have a solution estimate y0 by inverse quadratic search and an
estimate diffappr of the difference between it and the true solution. If
diffappr is small, often the estimate will be very close to the true solution,
so a small step in stepping down or up will likely give new better limits
lower and upper quickly. A little more than half diffappr seems optimal.
Inverse quadratic search can fail, and probably will if a good initial
approximation p0 is not found. Therefore the following steps are needed to
secure a solution within epsilon. */
step = 0.50001*diffappr;
if (step < epsilon) step = epsilon;
p0old = p0;
if (lower != y0) { // Here y0 = upper.
  for (d1=y0-step,j=1; d1>lower; d1-=step,j++) {
    p0 = tdist(d1,nu,0);
    // if p0 >= p0old we have deficient accuracy, so no use to continue.
    if (p0 >= p0old) { y0 = d1+step; goto THEEND2; }
    if (p0 == px) { y0 = d1; goto THEEND2; }
    if (p0 <  px) { lower = d1; break; }
    upper = d1;
    if (j == 4) break;
    p0old = p0;
  }
}
else if (upper != y0) { // Here y0 = lower.
  for (d1=y0+step,j=1; d1<upper; d1+=step,j++) {
    p0 = tdist(d1,nu,0);
    if (p0 <= p0old) { y0 = d1-step; goto THEEND2; } // Deficient accuracy.
    if (p0 == px) { y0 = d1; goto THEEND2; }
    if (p0 >  px) { upper = d1; break; }
    lower = d1;
    if (j == 4) break;
    p0old = p0;
```

```
    }
  }
  if (upper - lower < epsilon) goto THEEND1;
  // Bisection, ie interval halvings, as final step if needed.
  h2 = 1e300;
  for (;;) {
    d1 = 0.5*(lower+upper);
    p0 = tdist(d1,nu,0);
    if (p0 == px) { y0 = d1; goto THEEND2; }
    if (p0 > px) upper = d1; else lower = d1;
    h1 = upper - lower;
    if (h1 < epsilon || h1 >= h2) break; // If h1 gets stuck at values > epsilon.
    h2 = h1;
  }
  THEEND1: y0 = 0.5*(lower+upper);
  THEEND2: if (p > 0.5) return -y0; return y0;
}

double tdistinv_ap0(double p, int nu) {
  // Abramowitz & Stegun (26.7.5) page 949. Approximates t
  // distribution function inverse with nu degrees of freedom.
  double g1,g2,g3,g4,invdf,invdf2,t,t2,t3,t5,t7,t9;
  if (nu <= 0) return 0; // Bad argument.
  if (p == 0.5) return 0;
  invdf = 1.0/nu;
  invdf2 = invdf*invdf;
  if (p <= 0) return -1e30;
  if (p >= 1) return  1e30;
  if (nu == 1) return tan(PI*(p-0.5));
  if (nu == 2) return (2*p-1)/sqrt(2*p*(1-p));
  // Make argument of Abramowitz & Stegun (26.7.5) belong to (0,0.5).
  if (p < 0.5) t = p; else t = 1 - p;
  if (nu == 4) {
    t = 2*sqrt(p*(1-p));
    t = (4/t)*cos(acos(t)/3);
    t = -sqrt(t-4);
  }
  else {
    t = ndistinv_ap(t);
    t2 = t*t;
    t3 = t*t2;
    t5 = t3*t2;
    t7 = t5*t2;
    t9 = t7*t2;
    g1 = (t3+t)/4;
    g2 = (5*t5+16*t3+3*t)/96;
    g3 = (3*t7+19*t5+17*t3-15*t)/384;
    g4 = (79*t9+776*t7+1482*t5-1920*t3-945*t)/92160;
    t = t + invdf*g1 + invdf2*g2 + invdf*invdf2*g3 + invdf2*invdf2*g4;
  }
  if (p > 0.5) return -t;
  return t;
}
```

```
double tdistinv_ap(double p, int nu) {
  /*
   Approximation of t-distribution inverse with high accuracy and high speed
   2016-11, Stig Rosenlund

   Abramowitz & Stegun (26.7.5) page 949 plus correction in the form, for p < 0.5, -a[-log(p)]^q/nu^5.
   Approximates the t-distribution function inverse with nu degrees of freedom. The improvement from
   the correction is mostly better than a factor 0.01, ie if the error of the Abramowitz & Stegun
   approximation is err, then the error with the correction is mostly < 0.01*err. Often it is better
   than a factor 0.001. High speed is contingent on the C compiler/linker being able to generate code
   that quickly computes pow(,). It is my experience that this is so for the C compiler/linker that I
   use. More work has been spent on small values of p, since these are more important in applications.
   Possibly the arrays aarr[][] and qarr[][] can be replaced by combinations of trigonometric
   functions or polynomials or something like that without sacrificing accuracy.

   Degrees of freedom are grouped to give an index dinx. For nu = 1, 2, 4 there are exact expressions.
     nu    3  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30-39
     dinx  0  1  2  3  4  5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25   26

     nu  40-49   50-59   60-69   70-79   80-89   90-99  100-109  110-119  120-129  130-139  140-149  150-159
     dinx  27      28      29      30      31      32      33       34       35       36       37       38

     nu  160-169  170-179  180-189  190-199  200-209  210-219  220-229  230-239  240-249  >= 250
     dinx   39       40       41       42       43       44       45       46       47    no correction

   p in (0,0.5) is grouped to give an index pinx.
     p     <= 1e-10  <= 1e-9  <= 1e-8  <= 1e-7  <= 1e-6  <= 1e-5  <= 1e-4  <= 1e-3  <= 0.005  <= 0.01  <= 0.05
     pinx     0         1        2        3        4        5        6        7        8         9        10

     p     <= 0.10  <= 0.15  <= 0.20  <= 0.30  <= 0.40  <= 0.43  <= 0.46  <= 0.47  <= 0.48  <= 0.49  <= 0.495  > 0.495
     pinx    11       12       13       14       15       16       17       18       19       20       21     no correction

   Optimal coefficients were obtained by searching through grids of (a,q).
  */
  int dinx,pinx; // lastpinx[dinx] is the highest value of pinx for which the correction gives a noticeable improvement.
  static int lastpinx[]={21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,11,10,10,9,8,8,8,8,7,7,7,7,7,7,6,6,6,5,5,4,4};
  double g1,g2,g3,g4,invdf,invdf2,invdf3,invdf4,px,t,t2,t3,t5,t7,t9;
  static double aarr[48][22]={ // Numbers 0, 1, 2, ... , 47 within comment are values of dinx.
  // pinx, p:  0  <= 1e-10      1  <= 1e-9       2  <= 1e-8       3  <= 1e-7       4  <= 1e-6       5  <= 1e-5       6  <= 1e-4       7  <= 1e-3       8  <= 0.005      9  <= 0.01      10  <= 0.05
  //          11  <= 0.10      12  <= 0.15      13  <= 0.20      14  <= 0.30      15  <= 0.40      16  <= 0.43      17  <= 0.46      18  <= 0.47      19  <= 0.48      20  <= 0.49      21  <= 0.495
  /* 0*/{3.44633507694952e-7,3.00207124782614e-5,3.67389931571773e-4,1.52371813128548e-5,5.41388835759558e-5,7.73267086151340e-5,1.27984647510805e-4,1.32407929733969e-4,1.30405430369068e-4,1.16859947890005e-4,1.35994385853547e-4,
       1.49593876848649e-3,1.06939522818976e-2,1.67710758857320e-2,1.62334837222117e-2,1.38399961353467e-2,1.56011858146225e-2,2.29519308020922e-2,5.65382813830470e-2,1.70646853268079e-1,3.16368465694046e+0,9.89733200163356e+2},
  /* 1*/{4.74348562957583e-5,9.42854786559174e-5,9.76893991067984e-5,1.17083762402537e-4,1.53163161640682e-4,2.22742666253035e-4,2.78794384120881e-4,1.99555677549622e-4,1.56217935387785e-4,1.26544131127568e-4,1.41274837989413e-4,
       1.80106932065624e-3,1.31015996962378e-2,1.94609004387973e-2,1.82089107411889e-2,1.53757358437162e-2,1.73552061166346e-2,2.55707416763239e-2,6.18372253888134e-2,1.85637577714072e-1,3.53262719058138e+0,9.95035298217340e+2},
  /* 2*/{9.77492904035774e-5,1.02257054104318e-4,1.99191837489881e-4,1.69006171759032e-4,2.02561705963477e-4,1.98499258609462e-4,2.29465029088444e-4,2.07648312660267e-4,1.62053860587160e-4,1.27813288803522e-4,1.40105524995747e-4,
       1.84215683377456e-3,1.36513634281283e-2,2.00916726195107e-2,1.86153124008497e-2,1.56785394005058e-2,1.77060246292599e-2,2.60996607613180e-2,6.10414335901408e-2,1.94506070749631e-1,3.60967077214437e+0,1.01701293430356e+3},
  /* 3*/{1.20353946707277e-4,1.48888955116670e-4,1.83288992229604e-4,2.17191896196719e-4,2.39009745710713e-4,2.53692715013085e-4,2.50546160754135e-4,2.18971912906964e-4,1.66721351383596e-4,1.28351413685357e-4,1.38498883761350e-4,
       1.85588485036211e-3,1.39999775178117e-2,2.05288923011372e-2,1.88619838943023e-2,1.58605458132832e-2,1.79208204803245e-2,2.64276660112091e-2,6.39969655432333e-2,1.96915613447944e-1,3.65621935234422e+0,1.02129085023693e+3},
  /* 4*/{1.61000912612948e-4,1.94430429922932e-4,2.23992534245373e-4,2.85327480438251e-4,2.75181591415383e-4,2.80972017460694e-4,2.67175706912021e-4,2.27699784171728e-4,1.68871161700709e-4,1.28543880764471e-4,1.36864932829651e-4,
       1.85581787194047e-3,1.42161915922897e-2,2.08658563686191e-2,1.90126923058109e-2,1.59755332613841e-2,1.80621170515396e-2,2.66479374324953e-2,6.45465147296420e-2,1.98646105184525e-1,3.68618773685176e+0,1.03763506721341e+3},
  /* 5*/{2.04018340684331e-4,2.97564011918166e-4,3.64513084360628e-4,2.88202797663710e-4,3.01611562621595e-4,3.01144074126883e-4,2.91911202293549e-4,2.26634371230696e-4,1.71013149184370e-4,1.28563916930670e-4,1.35353550086791e-4,
       1.84868561401395e-3,1.43331620379301e-2,2.11627358638051e-2,1.90994884807695e-2,1.60520627729266e-2,1.81639540889156e-2,2.68122159455019e-2,6.49480363675274e-2,1.99814274927438e-1,3.70656885096482e+0,1.03434568254675e+3},
  /* 6*/{2.43381353158323e-4,2.72747234259826e-4,3.48918372511169e-4,3.64523617093060e-4,3.25348368929458e-4,3.17987108254763e-4,2.91443425520243e-4,2.78619647842826e-4,1.72417881092530e-4,1.28503570441338e-4,1.34028095487344e-4,
       1.83791740790312e-3,1.43642699624873e-2,2.14524982057676e-2,1.91383030351430e-2,1.61064186839282e-2,1.82471202048747e-2,2.69526043802963e-2,6.52022861377853e-2,2.00732015405344e-1,3.72120378053136e+0,1.04551163939119e+3},
  /* 7*/{2.78906753610076e-4,3.08590692347301e-4,3.28634553565199e-4,3.37093362078950e-4,3.44820882723810e-4,3.31847135142421e-4,2.99997654034800e-4,2.47303543391266e-4,2.06622549959427e-4,1.28407292332469e-4,1.32891018881873e-4,
       1.82532300099548e-3,1.43112984410658e-2,2.17875039913937e-2,1.91363841263587e-2,1.61486437513514e-2,1.83249136437788e-2,2.70917700052642e-2,6.55872764082964e-2,2.01542314569212e-1,3.73266823399072e+0,1.03839374423307e+3},
  /* 8*/{3.16430031026272e-4,3.40303164215610e-4,3.54150681982384e-4,3.63764511944302e-4,3.65835196948254e-4,3.43748538124267e-4,3.09890432015609e-4,3.20547773471464e-4,1.74350678685382e-4,1.28299810175898e-4,1.31944801889952e-4,
       1.81171994630610e-3,1.41681987569976e-2,2.21937253845129e-2,1.90958205279596e-2,1.61855541274391e-2,1.84086954325238e-2,2.72481147692111e-2,6.59050316480473e-2,2.02356106844738e-1,3.74201103503624e+0,1.03818462020790e+3},
  /* 9*/{3.51240689614779e-4,3.62837454127222e-4,3.75694401903772e-4,3.82243495135125e-4,3.80325351601553e-4,3.54339975106035e-4,3.11790067687466e-4,2.50113233193137e-4,1.75656332236530e-4,1.28195451608219e-4,1.31018506582307e-4,
       1.79727810557243e-3,1.39235595768462e-2,2.27177823271480e-2,1.90158536209215e-2,1.62221106829410e-2,1.85075390273875e-2,2.74384744297532e-2,6.63015100404183e-2,2.03303004245765e-1,3.75098407861224e+0,1.03659801280962e+3},
  /*10*/{3.69376638140992e-4,3.87149076662365e-4,3.96387833268972e-4,3.98636513031159e-4,4.81068788156442e-4,3.62404131197951e-4,3.18583010201322e-4,2.52585752421572e-4,1.76106014922158e-4,1.27916794960052e-4,1.30238173099532e-4,
       1.78210557454941e-3,1.35627822986942e-2,2.34069913324665e-2,1.88935057541844e-2,1.62624967278706e-2,1.86303433454826e-2,2.76757507539119e-2,6.68264352346798e-2,2.04403819013960e-1,3.76118935337969e+0,1.04188640810085e+3},
  /*11*/{3.93474249519955e-4,5.78304768792754e-4,4.16677544136698e-4,4.12869598154663e-4,4.77745078179333e-4,3.70134178214159e-4,3.22966914380818e-4,2.54753696472254e-4,1.76199072685550e-4,1.28038378097402e-4,1.29320142334069e-4,
       1.76594107899394e-3,1.30696824403599e-2,2.43234515332741e-2,1.87241379971912e-2,1.63103565412786e-2,1.87862589601877e-2,2.79917172878375e-2,6.74668949800311e-2,2.05786626730403e-1,3.77221831753044e+0,1.02901215487798e+3},
  /*12*/{5.04920488343171e-4,5.68760645848741e-4,4.30790492920686e-4,4.26529812649657e-4,4.08241869694256e-4,3.77197666034202e-4,3.26917207669419e-4,2.56653831805626e-4,1.76403256107547e-4,1.27976356242267e-4,1.28554462423724e-4,
       1.74826308691296e-3,1.24296259339952e-2,2.55497434823096e-2,1.85013285901124e-2,1.63693002306856e-2,1.89851036720072e-2,2.83944376465210e-2,6.82925455467532e-2,2.07516786515038e-1,3.78578126543682e+0,1.02267326753668e+3},
  /*13*/{4.90881386297646e-4,5.62062021287158e-4,4.45691807656907e-4,4.72986779368196e-4,4.01476813556111e-4,3.83011326942083e-4,3.30741468222724e-4,2.58311982185181e-4,1.76815978078844e-4,1.27950756068816e-4,1.28054621021080e-4,
       1.72876645575036e-3,1.16331998332610e-2,2.71995441605640e-2,1.82176313590629e-2,1.64429395095002e-2,1.92377641121567e-2,2.89135042470325e-2,6.93683969671768e-2,2.09789892350249e-1,3.80192046179374e+0,1.02233867813342e+3},
  /*14*/{4.56103161604920e-4,5.13707986573907e-4,4.52740184032730e-4,4.47482489309145e-4,3.98850556629570e-4,3.97002824581698e-4,3.33695871343109e-4,2.59822004914501e-4,1.76948091259306e-4,1.27956231601717e-4,1.27429971866572e-4,
       1.70698602197330e-3,1.06801037559844e-2,2.94338196756512e-2,1.78384166760876e-2,1.65350908573704e-2,1.95564045769575e-2,2.95789921110543e-2,7.07297080370526e-2,2.12750953726575e-1,3.82206004438185e+0,1.00379874539701e+3},
  /*15*/{4.70362816636546e-4,5.32803339395409e-4,4.77647266306744e-4,4.51903720097926e-4,4.42499389145884e-4,3.93747609177122e-4,3.36445479916887e-4,2.61131894601901e-4,1.77166222934777e-4,1.27995639935773e-4,1.27034801772080e-4,
       1.68226195442123e-3,9.49008533981530e-3,3.24926331782688e-2,1.73179884679374e-2,1.66495060822486e-2,1.99545970319882e-2,3.04274549721586e-2,7.23052989088839e-2,2.15662911756125e-1,3.84986495360583e+0,9.89775308632496e+2},
  /*16*/{4.53372956401104e-4,5.05525224791380e-4,4.79920265691021e-4,4.43637529207621e-4,4.41429873473300e-4,3.95867575989284e-4,3.39039917882771e-4,2.62336754716375e-4,1.76753548550458e-4,1.28074321581570e-4,1.26554734631102e-4,
       1.65412469836483e-3,8.10683960809295e-3,3.67430163657064e-2,1.66731996888223e-2,1.67898566865688e-2,2.04478563067686e-2,3.15039608505435e-2,7.48487566235096e-2,2.21536085655827e-1,3.88638828199717e+0,9.72269234947992e+2},
  /*17*/{4.37321487197236e-4,5.12918763475494e-4,4.91464669638562e-4,4.40579380224473e-4,5.67482335159668e-4,4.00677470689838e-4,3.41205131335207e-4,2.63476881059525e-4,1.76931559645361e-4,1.28197374442589e-4,1.26057609379515e-4,
       1.62223729749596e-3,6.63165744249108e-3,4.27774630702877e-2,1.59095067279662e-2,1.69602268545381e-2,2.10537331024733e-2,3.28684932969248e-2,7.78769774753416e-2,2.27827817035192e-1,3.93304957439624e+0,1.06715830589240e+3},
  /*18*/{4.46153941173522e-4,5.33920333476978e-4,5.00839983727485e-4,4.37961001459101e-4,5.63264948858245e-4,4.04807070993364e-4,3.43421846852597e-4,2.64490399956006e-4,1.76434071632225e-4,1.28370974972882e-4,1.25624937727871e-4,
       1.58580039507626e-3,5.15902500963203e-3,4.35311485344446e-2,1.50345461675206e-2,1.71636726118173e-2,2.17920852390673e-2,3.45919157451714e-2,8.17420580298347e-2,2.36353206865261e-1,3.98190013919609e+0,1.04723079299561e+3},
  /*19*/{5.25962389352656e-4,5.29981807837644e-4,5.03917958913896e-4,6.28947475069093e-4,4.53812489714579e-4,5.74959005082552e-4,3.45240740567587e-4,2.65410947151876e-4,1.76016902500538e-4,1.28603741284397e-4,1.25308568670583e-4,
       1.54464100091952e-3,4.00458502473928e-3,4.32877231162078e-2,1.40560490682777e-2,1.74038875904340e-2,2.26851993411352e-2,3.67724926857254e-2,8.68744838756356e-2,2.47141536865708e-1,4.06736173259584e+0,1.01218291863986e+3},
  /*20*/{5.39088907899672e-4,5.30694093449409e-4,5.20512085509119e-4,5.83404963593876e-4,4.73548167554257e-4,5.74750602630800e-4,3.47190651215826e-4,2.66303415143916e-4,1.75733855247423e-4,1.28897334031528e-4,1.24935463397838e-4,
       1.49850332190438e-3,3.03834189257739e-3,4.30116744816806e-2,1.29886819015835e-2,1.76833265009085e-2,2.37589801404355e-2,3.95413411197596e-2,9.36282353665601e-2,2.61818754654510e-1,4.17283456903525e+0,1.08896348739973e+3},
  /*21*/{5.47817491431665e-4,5.40646428067415e-4,5.24478390258751e-4,5.80812446558380e-4,4.61533579616477e-4,5.69527594670992e-4,3.48882387631807e-4,2.67150304535465e-4,1.75175701964772e-4,1.29035653347340e-4,1.24644744504062e-4,
       1.44638231585203e-3,2.19948367502363e-3,4.26990395611649e-2,1.18507520207279e-2,1.80034283462214e-2,2.50425310402714e-2,4.30682891873924e-2,1.02914850950724e-1,2.82412582496688e-1,4.31704421953656e+0,1.02774213971935e+3},
  /*22*/{5.48890121443586e-4,5.52285600795030e-4,5.29887639317008e-4,5.78159962121910e-4,4.70563790642400e-4,5.69226194234811e-4,3.50349269223834e-4,2.67965106127432e-4,1.74389537673754e-4,1.29052639260608e-4,1.24428849180572e-4,
       1.38858504953756e-3,1.51342392003420e-3,4.23303364329974e-2,1.06645355094625e-2,1.83647822680793e-2,2.65668170818000e-2,4.75968704077710e-2,1.15689226215335e-1,3.12235903352722e-1,4.52208478403791e+0,1.08154683477860e+3},
  /*23*/{5.66671522096945e-4,5.52470244391968e-4,5.31823248224988e-4,5.75870040852945e-4,4.68727126818648e-4,5.69463230350887e-4,3.51731125771223e-4,2.68749609406511e-4,1.74312656837993e-4,1.29896027166374e-4,1.22566596320074e-4,
       1.32504703474613e-3,9.83542748043309e-4,4.18904195682249e-2,9.45619156339423e-3,1.87660566975616e-2,2.83698863016280e-2,5.34590860163899e-2,1.34103833876613e-1,3.55148934601871e-1,4.95161437616298e+0,9.97786775683892e+2},
  /*24*/{5.75078289881087e-4,5.62326751693373e-4,7.09633617151503e-4,5.13320643416410e-4,4.71528485424269e-4,5.64571821275265e-4,3.53221236008443e-4,2.69518682024428e-4,1.72395936186432e-4,1.30464116841065e-4,1.20177794683818e-4,
       1.25536932384369e-3,5.92414714928063e-4,4.13744905593628e-2,8.25419081804578e-3,1.85477729629440e-2,3.04906996977702e-2,6.11171870722782e-2,1.61625457446831e-1,4.23299257859161e-1,5.45811878612784e+0,9.17902914136248e+2},
  /*25*/{5.83082572286858e-4,5.91366594316050e-4,5.46895426575181e-4,5.11418657937554e-4,4.74614880237478e-4,5.64622691440317e-4,3.54605062611135e-4,2.70278060117479e-4,1.71115725814176e-4,1.31257535194521e-4,1.17090005824517e-4,
       1.18098225161752e-3,3.42779627897889e-4,4.07843443915119e-2,7.08500119328183e-3,1.80335201708750e-2,3.29739616993875e-2,7.12846514377978e-2,2.05636228731224e-1,5.36773092851047e-1,6.28525605776943e+0,1.01859969018730e+3},
  /*26*/{8.03217077228361e-4,5.14558199188820e-4,6.83180294910663e-4,5.24907340988175e-4,5.35977255336844e-4,5.65425358065683e-4,4.01953558653857e-4,2.85942619307910e-4,1.71342497860591e-4,1.33308656326734e-4,1.13871142268528e-4,
       1.10083324820213e-3,1.16360213912843e-4,3.52702324999289e-2,1.00163196838456e-2,1.29456093397798e-2,4.17524323616205e-2,1.57164243103783e-1,3.33823431258877e+0,3.11273487038255e+0,9.64073589213485e+1,2.09066904749359e+2},
  /*27*/{8.91081548685291e-4,6.65734853188891e-4,6.33381093776777e-4,6.34779789565597e-4,5.18237394939917e-4,4.93657150620608e-4,3.84707645327670e-4,2.83705741596460e-4,1.29160001898177e-4,1.14484036373828e-4,8.19519054198663e-5,
       1.04838249530449e-4},
  /*28*/{8.13740355095384e-4,7.85617984901167e-4,6.61867818791894e-4,5.83588636051021e-4,5.54025470003370e-4,4.83862709793574e-4,4.02181384460294e-4,3.03697971487739e-4,6.43876871339315e-5,1.15269011019482e-4,4.24072588014610e-5},
  /*29*/{7.90592515936261e-4,7.79017746601403e-4,6.76052462247520e-4,6.22420422230012e-4,5.52407575603340e-4,4.89857795958495e-4,4.49814485374929e-4,3.63946890744461e-4,1.82596053034276e-5,1.31875321413465e-4,1.44047086241372e-5},
  /*30*/{7.81270070513286e-4,7.72166850326333e-4,6.41279647901634e-4,5.74214241461855e-4,5.16841174195515e-4,4.68454047634303e-4,5.00414059010284e-4,4.92357977225901e-4,1.17214783425295e-6,5.56964703342038e-3},
  /*31*/{7.26138180319975e-4,7.67674449710096e-4,6.35784115141580e-4,5.76997870344526e-4,5.37434273688657e-4,4.77709329850056e-4,6.55432818945172e-4,7.83916732822359e-4,6.07335668156076e-7},
  /*32*/{8.25973098113916e-4,8.35464517660097e-4,6.53055167467509e-4,5.59371060941993e-4,5.07499295141321e-4,4.21786631174429e-4,9.88192989606285e-4,1.50787673113359e-3,2.64993359916249e-7},
  /*33*/{7.81173375130960e-4,7.61308044628859e-4,6.24546096337409e-4,5.26629431081286e-4,5.07583781165872e-4,4.16065305915108e-4,1.49331925456778e-3,3.43491485672337e-3,8.07127774988837e-19},
  /*34*/{8.18000083394990e-4,7.59195897480286e-4,6.45181000522865e-4,4.82481917377175e-4,5.02394275714564e-4,3.99125401308838e-4,4.26413781277048e-3,9.00050529578265e-3,7.34934866778169e-31},
  /*35*/{8.18997663114254e-4,8.27585110226621e-4,5.78855768815449e-4,4.27250909473170e-4,3.25030292323644e-4,3.79479084262818e-4,1.25697983024634e-2,2.53108244296941e-2},
  /*36*/{7.79401952737229e-4,8.25910778621563e-4,5.78903945436632e-4,3.61411376451667e-4,2.51548779972992e-4,3.52645468118569e-4,5.93354994060002e-2,4.62167469062600e-2},
  /*37*/{7.78270305049908e-4,7.52215338687270e-4,6.04853600487151e-4,2.88534439223016e-4,1.75633464274798e-4,3.18907839797447e-4,7.32514790771517e-2,5.47436679919678e-2},
  /*38*/{7.65750704929646e-4,7.52091796689232e-4,4.38866215704080e-4,2.18614753889105e-4,1.07448450607963e-4,2.74785400239134e-4,7.66291278646169e-2,6.59147705311995e-2},
  /*39*/{7.64023597421009e-4,7.51229356144522e-4,3.49646622376265e-4,1.43902313724870e-4,5.48517348344653e-5,2.29399041463292e-4,7.95020537877527e-2,8.05511485864390e-2},
  /*40*/{7.65459673086324e-4,7.51353439165879e-4,2.81302445126381e-4,8.55590586029875e-5,2.60004447451835e-5,2.42412021144762e-4,4.08865282926348e-2,5.16683694313073e-2},
  /*41*/{7.75301349534803e-4,5.30617871973037e-4,2.12406167184827e-4,4.35613459782999e-5,5.28341892510263e-6,1.16937449412873e-4,4.31688797072457e-2},
  /*42*/{8.17497701485619e-4,4.82748911669274e-4,1.53919354708562e-4,2.76072240716552e-5,1.06301834398117e-5,6.12398362621449e-5,4.60428229089102e-2},
  /*43*/{7.73692810776138e-4,4.42032203242679e-4,9.88041939485472e-5,7.93435790631841e-6,1.66809546220957e-5,1.84061772105625e-5,4.92492156074971e-2},
  /*44*/{8.23756102732635e-4,4.26785186192883e-4,5.98420498342891e-5,1.31585432464180e-6,4.90360724359819e-7,8.22057163064082e-5},
  /*45*/{7.78011912065956e-4,3.98514229540127e-4,4.00519768261123e-5,1.32901603987120e-5,2.29698944597972e-16,7.34934866778169e-31},
  /*46*/{7.82433002647521e-4,4.34679246989963e-4,3.80055107340647e-5,2.12850487668280e-5,4.43030378008456e-19},
  /*47*/{7.84898579559688e-4,4.04436307825203e-4,4.00930962858175e-5,1.35942727620095e-5,7.34934866778169e-31}
  },
  qarr[48][22]={
  // pinx, p:  0 <= 1e-10     1  <= 1e-9      2  <= 1e-8      3  <= 1e-7      4  <= 1e-6      5  <= 1e-5      6  <= 1e-4      7  <= 1e-3      8  <= 0.005     9  <= 0.01     10  <= 0.05
  //          11  <= 0.10     12  <= 0.15     13  <= 0.20     14  <= 0.30     15  <= 0.40     16  <= 0.43     17  <= 0.46     18  <= 0.47     19  <= 0.48     20  <= 0.49     21  <= 0.495
  /* 0*/{8.87854124328936,7.44481038362450,6.57902913719042,7.63113431435094,7.17474893353696,7.03282689147881,6.82437814943395,6.80442007550663,6.81359676706201,6.87792481734457,6.77556971751486,
       4.63725295170261,2.36201212817973,1.68531143714478,1.77284834661521,2.67415624190147,4.42260006458968,6.66902387890454,10.3107861811833,14.2279874186598,23.6420190139895,40.8682431650054},
  /* 1*/{7.01344307038601,6.79029658470489,6.77386737372586,6.71024164623234,6.61394994822985,6.47069797554031,6.37250566070858,6.51163984053868,6.63516026595354,6.75844811572125,6.68278194221283,
       4.40782797593455,2.11461228645856,1.52113662980309,1.68036529132649,2.62903070181241,4.39987314090226,6.65521317441180,10.2263389028610,14.1361201480334,23.6369450092770,40.5591597877056},
  /* 2*/{6.72245664002385,6.70604533558610,6.48469458729917,6.53748316976248,6.47259853248842,6.48134083416482,6.42485243156380,6.46991219332283,6.59655161137053,6.73546283000056,6.67171625146981,
       4.36698867506903,2.05298111267926,1.47368488609343,1.65379361247154,2.61514500964830,4.39406505020135,6.65214972587036,10.1009333477520,14.2203428239939,23.6377040037382,40.5606465626493},
  /* 3*/{6.61430564667278,6.54632842370406,6.47788426688926,6.41972611011006,6.38547057324695,6.36312151582646,6.36854088341041,6.42915118399196,6.56812562291210,6.72104929558351,6.66758289594242,
       4.34394301557796,2.01003165683161,1.43637709476278,1.63434247463743,2.60360717867606,4.39043002277222,6.65112766340664,10.2275046897013,14.2189224668744,23.6379323567460,40.5359023213746},
  /* 4*/{6.49281541271742,6.43275227645892,6.38623926435695,6.30289529862793,6.31462448851732,6.30678316638884,6.32745276086407,6.39924428557056,6.55168412322323,6.71137129038463,6.66665065044945,
       4.33067981855690,1.97983927975696,1.40459050617319,1.62032834030688,2.59297751955169,4.38894526916783,6.65239840681928,10.2297688415591,14.2217688926421,23.6388446374212,40.5591529501645},
  /* 5*/{6.39605019404746,6.27500847932300,6.20507099145407,6.28279632257516,6.26653196038798,6.26724247503463,6.27950237993690,6.39231364160851,6.53760260564099,6.70453498380830,6.66718958220392,
       4.32317020998742,1.95956489611789,1.37489113040562,1.61100287357729,2.58206501974042,4.38980487015354,6.65653932451904,10.2341392783892,14.2249387099535,23.6408864783377,40.5359936101397},
  /* 6*/{6.32328588341644,6.28703219308151,6.20526286580602,6.18819550843061,6.22749760495728,6.23625648779549,6.27186467451605,6.28981571792594,6.52747258893923,6.69947419514251,6.66827064717160,
       4.31925240099213,1.94805331290613,1.34531232110141,1.60619767428480,2.56974408822813,4.39356976960038,6.66441743767415,10.2372072486603,14.2305255770771,23.6443997986907,40.5591534286400},
  /* 7*/{6.26666039004004,6.23447994289085,6.21370379050607,6.20511285576808,6.19714264354551,6.21177652118485,6.25299514333076,6.21373086393915,6.42585599095876,6.69558402567642,6.66948063922227,
       4.31769645742354,1.94519068749329,1.31209826750327,1.60614706023295,2.55482464405191,4.40057553292613,6.67718831709048,10.2526344893754,14.2392386008316,23.6500391232879,40.5359936124852},
  /* 8*/{6.21582734324654,6.19254704033977,6.17931812151888,6.17016865960192,6.16805748401827,6.19162358033609,6.23379738768727,6.21494772792089,6.51294858794314,6.69248685123133,6.67051219431568,
       4.31784504902580,1.95163172352753,1.27346884985506,1.61140619193283,2.53597989659948,4.41204031279190,6.69631470325224,10.2683050838254,14.2520969900777,23.6579724850461,40.5359936101396},
  /* 9*/{6.17373521650810,6.16322292039639,6.15179394155303,6.14592222817555,6.14763956438644,6.17443532861399,6.22669301071187,6.32566657968967,6.50570415471719,6.68993538508214,6.67209413315764,
       4.31942011592061,1.96866724376741,1.22615750543933,1.62278527704585,2.51172105369977,4.42893575398729,6.72355664515125,10.2918879568919,14.2709571869407,23.6694586880507,40.5360004815137},
  /*10*/{6.15000119946131,6.13503598226072,6.12728753121700,6.12538637290087,6.05652073090007,6.16111547590692,6.21374241449194,6.31782357569516,6.50159194885180,6.68864390518595,6.67325181852407,
       4.32223551723458,1.99815681643625,1.16713360767781,1.64134866038875,2.48033018858620,4.45253742541635,6.76034616712707,10.3259602583736,14.2961610483408,23.6860014576769,40.5591529836697},
  /*11*/{6.12344008826464,5.99940070491569,6.10497802246541,6.10811629050119,6.05454766266111,6.14899685712264,6.20467247493381,6.31104106389272,6.49898540751427,6.68578913873768,6.67557875959602,
       4.32632230452059,2.04254594105425,1.09280853193066,1.66840612338766,2.43986193925530,4.48431223616571,6.81123030540647,10.3698548757673,14.3301424616719,23.7076628994734,40.5358954500006},
  /*12*/{6.03960370278333,5.99940070491569,6.08897569494151,6.09241776437826,6.10816520540109,6.13826596158152,6.19665463173704,6.30514544077269,6.49630217422906,6.68407673946091,6.67739025565658,
       4.33187389088801,2.10484865265371,0.99904467171230,1.70553853613368,2.38809455637592,4.52587741499114,6.87694706115661,10.4275570066557,14.3744797314947,23.7365414106280,40.5359931315007},
  /*13*/{6.04330970353715,5.99854450314134,6.07336510553289,6.05258234127723,6.11082170380573,6.12936944544284,6.18923068545564,6.30002090931961,6.49324568657168,6.68240149794952,6.67799179786182,
       4.33902061681674,2.18864310836233,0.88106081709816,1.75456995296154,2.32252247979411,4.57902642806721,6.96149771426393,10.5027533987976,14.4328514627717,23.7735232084623,40.5591534621446},
  /*14*/{6.06178312279039,6.02345428428862,6.06438226111858,6.06853653746345,6.11020927781830,6.11268100090815,6.18326009540222,6.29542639783669,6.49124659077748,6.68074671366772,6.67949807291229,
       4.34800092720169,2.29810198161909,0.73333732334759,1.82095197512799,2.24033455882126,4.64562394931597,7.06859058510728,10.5974976370442,14.5083378387297,23.8208526762492,40.5373954215507},
  /*15*/{6.04817256135266,6.00802526895933,6.04324963713361,6.06211213341274,6.06959368600346,6.11368912514323,6.17783023913307,6.29141621192774,6.48913947312266,6.67906288044368,6.67989332626067,
       4.35918393090387,2.45236308222042,0.54934711658594,1.91210339164945,2.13843672378485,4.72755052287044,7.20233616451871,10.7081592448405,14.5904480481802,23.8828837782366,40.5372909336694},
  /*16*/{6.05609997871835,6.02172627569192,6.03863468487384,6.06585267424036,6.06814112375445,6.10951949511284,6.17280519254394,6.28778017453796,6.48911335948362,6.67729506879482,6.68074695650746,
       4.37288728427254,2.66074264843239,0.32144034458873,2.02815261521889,2.01348759554475,4.82670306176230,7.36721783524114,10.8726952523929,14.7262612398225,23.9618612368139,40.5359931673506},
  /*17*/{6.06422831396100,6.01411933075097,6.02804412204694,6.06586562562118,5.97416526829371,6.10295544683399,6.16853748356843,6.28441761310393,6.48740750733152,6.67539385358747,6.68181215386277,
       4.38940703126666,2.92810474469757,0.04027625866829,2.17115428867464,1.86159886719170,4.94479707219761,7.56876836751205,11.0649586494736,14.8771712136520,24.0609422635505,40.8681454500004},
  /*18*/{6.05508242210659,5.99854450314134,6.01932654715938,6.06577290537058,5.97495630014503,6.09733944207624,6.16437453059534,6.28141899590928,6.48788403253135,6.67330846734242,6.68258013649756,
       4.40932109714223,3.26356338843213,0.00000018980401,2.34349261422690,1.67897489937170,5.08338337309735,7.81201104028499,11.3001166842567,15.0708854893235,24.1761481491201,40.8915006315006},
  /*19*/{6.00103465018868,5.99854450314134,6.01510736799244,5.93781917612031,6.05226190339961,5.95956676152813,6.16082980820652,6.27870468050789,6.48818087585884,6.67097962135281,6.68265514037803,
       4.43295720505257,3.60276670342539,0.00000001161384,2.54816038691618,1.46092848441968,5.24356653839289,8.10336889092031,11.5938861911117,15.3071195263209,24.3367474212192,40.8915006338457},
  /*20*/{5.99095426248418,5.99595276918714,6.00229209722125,5.96219031087440,6.03500922962744,5.95827495083774,6.15725000671083,6.27614281363544,6.48816591274154,6.66838375695969,6.68303429246756,
       4.46074795704122,3.97270992110382,0.00000000027759,2.78787205419698,1.20291158601904,5.42625546969990,8.44985713251758,11.9538864388160,15.6065853146724,24.5329198605419,41.2236529500006},
  /*21*/{5.98376940712769,5.98796055866009,5.99795919324567,5.96209380588970,6.04304202887356,5.96053493350631,6.15411076793874,6.27375318710620,6.48904333125681,6.66652017594959,6.68286134200142,
       4.49363181925671,4.40538395082806,0.00000010322439,3.06544711725097,0.89962612530879,5.63170408661543,8.85825601968767,12.4045899719621,15.9902129555829,24.7821835647603,41.2005000027099},
  /*22*/{5.98116056405324,5.97926055951944,5.99286761450799,5.96219034398699,6.03455366651134,5.95955462903831,6.15134215732148,6.27150187578252,6.49067528858887,6.66519402381260,6.68229471793458,
       4.53194893553080,4.90561774720062,0.00000000443666,3.38363612947686,0.54527806615278,5.85911527898487,9.33701259393381,12.9592236377316,16.4862875240058,25.1040033761888,41.5326523212102},
  /*23*/{5.96936817004007,5.97747810849139,5.99009850613863,5.96218355320754,6.03481061773266,5.95827495283794,6.14875040688684,6.26937735949999,6.49012219527933,6.65989422082278,6.69225906106343,
       4.57635173058712,5.48137600081426,0.00000096221211,3.74490781160120,0.13334371839973,6.10799124477355,9.89418686956058,13.6531584801986,17.1137148867309,25.6077222716486,41.5341459336950},
  /*24*/{5.96307511839861,5.97021962246782,5.89230900686159,6.00097795011380,6.03148233831580,5.96061864557218,6.14610593537817,6.26734628178204,6.49546590422353,6.65593757712452,6.70580574847700,
       4.62795006952934,6.15590734495998,0.00000001304384,4.15132602895094,0.00007148826131,6.37614090814158,10.5375564485910,14.5226374635541,17.9448071937255,26.1881742022273,41.5814903225689},
  /*25*/{5.95718636579878,5.95252583140651,5.97805048861508,6.00108544675634,6.02803529205798,5.95964420650180,6.14367137745547,6.26539285174156,6.49882349082795,6.65084278870731,6.72431677910470,
       4.68654342193264,6.88500634099562,0.00000001642206,4.60522767362151,0.00020378878954,6.66127966099117,11.2791145620514,15.6313649184611,19.0445083154205,26.9796816871862,42.2645830201633},
  /*26*/{5.85448730957105,5.99372446064607,5.90158950026395,5.98932559230975,5.98174467709243,5.95819167252158,6.09020644134625,6.23812380817126,6.49733137917577,6.63655250539327,6.74408499127174,
       4.75426090168868,8.39144964821183,0.00000003980557,4.25188542264241,0.00000000000000,6.80071288292247,15.2575250753338,27.3634776515014,27.1179885852997,38.6776385289564,41.3789999998523},
  /*27*/{5.81228529286051,5.90230632183072,5.91804717367604,5.91635758746553,5.98688569408781,6.00442020309816,6.10275672468649,6.23834418138512,6.64284594291619,6.69793936969133,6.99128213021060,
       7.12517259181000},
  /*28*/{5.83454104323649,5.84412272858004,5.89858145797886,5.94069081797558,5.95872842784221,6.00825515953151,6.08127112943043,6.20694179742136,7.00810811935604,6.61893633227705,7.49703225163161},
  /*29*/{5.83966348130293,5.84326878225865,5.88834079763417,5.91557720329275,5.95694226051324,6.00077988531161,6.03295051334691,6.12751366592138,7.66591507892928,6.34843413034906,8.33238156778920},
  /*30*/{5.84066030205454,5.84357700275861,5.90346726909142,5.94107101232574,5.97867456755748,6.01534108630344,5.98731724931698,5.99727948086030,9.10407778247709,3.36284887621271},
  /*31*/{5.86133721250225,5.84357123608553,5.90459162871362,5.93769496957478,5.96249105286323,6.00532029818702,5.87391735404321,5.79879232219376,9.38932042930999},
  /*32*/{5.81986927423906,5.81492802844386,5.89439593608853,5.94691152377016,5.98112878512554,6.04966203194317,5.70224029194388,5.52193451334479,9.77265327952435},
  /*33*/{5.83608762602267,5.84369659611463,5.90809053997517,5.96633664112020,5.97848896831906,6.05088590062707,5.53269063775529,5.17790029535176,23.5354829250001},
  /*34*/{5.82083306090915,5.84370276424661,5.89619023539109,5.99516236720821,5.97901477627350,6.06134078232001,5.09245977327932,4.78092827680508,37.8075000000000},
  /*35*/{5.81977502163348,5.81525953342470,5.93148997535904,6.03560563667491,6.13352668832883,6.07316407509525,4.64304248133007,4.36321912500010},
  /*36*/{5.83474912525310,5.81535464653617,5.93041470579881,6.09162229525265,6.22149936654734,6.09097728672380,3.99505387500009,4.15481987341507},
  /*37*/{5.83491981381961,5.84505634789998,5.91481575463319,6.16726348012665,6.34527659760319,6.11550733303921,3.92991637500009,4.15499622711007},
  /*38*/{5.83988944189900,5.84484192687324,6.02109043344602,6.26029372715979,6.51497747611483,6.15366761735408,3.93008175000009,4.15591080000005},
  /*39*/{5.84069483976392,5.84505588363196,6.09625756080710,6.40168870801777,6.74766894110357,6.19686428665092,3.93008175000009,4.15591080000005},
  /*40*/{5.84039199139402,5.84496156880806,6.16775195857928,6.57725093288728,7.00288454436880,6.14005828596616,4.22697732711007,4.45566906561009},
  /*41*/{5.83692993904478,5.95765124408406,6.26025970464196,6.80541662764861,7.56302286698274,6.37001980230300,4.22697732711007},
  /*42*/{5.82101591196501,5.98850766572766,6.36618607605586,6.95366355534892,7.27863464703176,6.54611519377181,4.22680097341507},
  /*43*/{5.83931219021858,6.01752641093029,6.51252431720544,7.37728580476426,7.07673525475883,6.89539258022876,4.22949666561007},
  /*44*/{5.82092780254312,6.02942867625229,6.67770595170929,7.99068425786202,8.30776505467832,6.11644295401205},
  /*45*/{5.84049375687960,6.05236204502707,6.80813769899378,7.16427951107596,16.0462500000000,28.3905000000001},
  /*46*/{5.84071726324034,6.02528839771934,6.82021188123042,6.97935161056820,18.1680427500000},
  /*47*/{5.84211281229260,6.04985742356713,6.79736246727455,7.10774737155178,27.7830000000001}
  },
```

```
  parr[]={1e-10,1e-9,1e-8,1e-7,1e-6,1e-5,1e-4,1e-3,0.005,0.01,0.05,
    0.1,0.15,0.2,0.3,0.4,0.43,0.46,0.47,0.48,0.49,0.5};

  if (nu <= 0) return 0; // Bad argument.
  if (p == 0.5) return 0;
  if (p <= 0) return -1e30;
  if (p >= 1) return  1e30;
  if (nu == 1) return tan(PI*(p-0.5));
  if (nu == 2) return (2*p-1)/sqrt(2*p*(1-p));
  if (nu == 4) {
    t = 2*sqrt(p*(1-p));
    t = (4/t)*cos(acos(t)/3);
    t = -sqrt(t-4);
    if (p > 0.5) return -t;
    return t;
  }
  if (p > 0.5) px = 1 - p; else px = p;
  invdf = 1.0/nu;
  invdf2 = invdf*invdf;
  invdf3 = invdf*invdf2;
  invdf4 = invdf2*invdf2;
  t = ndistinv_ap(px);
  t2 = t*t;
  t3 = t*t2;
  t5 = t3*t2;
  t7 = t5*t2;
  t9 = t7*t2;
  g1 = (t3+t)/4;
  g2 = (5*t5+16*t3+3*t)/96;
  g3 = (3*t7+19*t5+17*t3-15*t)/384;
  g4 = (79*t9+776*t7+1482*t5-1920*t3-945*t)/92160;
  t = t + invdf*g1 + invdf2*g2 + invdf3*g3 + invdf4*g4;
  if (nu >= 250 || px > 0.495) {
    if (p > 0.5) return -t;
    return t;
  }
  // Correction.
  if      (nu ==  3) dinx = 0;
  else if (nu <= 29) dinx = nu - 4;
  else               dinx = nu/10 + 23;
  for (pinx=0; px>parr[pinx]; pinx++) ;
  if (pinx > lastpinx[dinx]) { if (p > 0.5) return -t; return t; }
  t2 = -aarr[dinx][pinx]*pow(-log(px),qarr[dinx][pinx])*invdf3*invdf2;
  if (p > 0.5) return -t - t2;
  return t + t2;
}
```