Jacques POITEVINEAU & Bruno LECOUTRE

STATISTICAL DISTRIBUTIONS FOR BAYESIAN EXPERIMENTAL DATA ANALYSIS

FORTRAN FUNCTIONS
2. DISCRETE DISTRIBUTIONS

2010

ERIS ♢♢♢ eris62.eu

PRELIMINARY

Download the free FMLIB package of David M. Smith:

`dmsmith.lmu.build/`

The FM package performs multiple-precision real, complex, and integer arithmetic. It provides the intrinsic Fortran numerical functions, as well as many special functions that are not included in the Fortran standard.

In addition to these three basic arithmetic types, multiple-precision exact rational arithmetic and interval arithmetic are also available.

One of the primary uses of the package is to study the accuracy and stability of numerical algorithms by comparing results computed with several different levels of precision.

Contents

Part I
CUMULATIVE DISTRIBUTION FUNCTIONS

Chapter 1
Binomial distribution

Function binomcdf(m, n, p, ier)
File : binomcdf.f90

```
function binomcdf( m, n, p, ier )

!-----------------------------------------------------------------------
!
!     Computes the probability that a random variable distributed
!     according to the binomial distribution with sample size N and
!     parameter P, is less than or equal to M.
!
!     M     - Input . Value of the variable (M >= 0 )         - Integer
!     N     - Input . Sample size (N > 0)                     - Integer
!     P     - Input . parameter of the binomial (0 <= P <= 1) - Real
!     IER   - Output. Return code :                           - Integer
!                     0 = normal
!                     1 = invalid input argument
!                     3 = result out of limits in BETACDF
!
!     External functions called:
!       BETACDF (and DLGAMA if not in Fortran library)
!
!     Fortran functions called:
!       NONE
!
!-----------------------------------------------------------------------

   implicit none

   ! Function
   ! --------

   real(kind=8) :: binomcdf

   ! Arguments
   ! ---------

   real(kind=8), intent(in) :: p
   integer, intent(in) :: m, n
   integer, intent(out) :: ier

   ! Local declarations
   ! ------------------

   real(kind=8), external :: betacdf

   real(kind=8), parameter :: zero=0.0_8, one=1.0_8
   real(kind=8) :: t

   !-----------------------------------------------------------------------

   binomcdf = zero
   ier = 0

   ! Test for valid input arguments
```

```
    if ( n <= 0 .or. m < 0 .or. m > n .or. p < zero .or. p > one ) then
      ier = 1
      binomcdf = -one
      return
    end if

    !  Special cases

    if ( m == n ) then
      binomcdf = one
      return
    end if
    if ( p == one ) return
    if ( p == zero ) then
      binomcdf = one
      return
    end if

    !  General case

    t = n - m
    binomcdf = one - betacdf( p, m+one, t, ier )

  end function binomcdf
```

Chapter 2
Hypergeometric distribution

Function hypergcdf(x, n, m, npop, e, ier)
File : hypergcdf.f90

```
function hypergcdf( x, n, m, npop, e, ier )

!------------------------------------------------------------------------
!
!     Computes the probability that a random variable distributed
!     according to the hypergeometric distribution with parameters
!     N, NPOP, M is less than or equal to X.
!
!     X     - Input . Value of the variable: number of        - Integer
!                     "positive" events in the sample
!                     (0 <= X <= N)
!     N     - Input . Sample size (0 < N <= NPOP)             - Integer
!     M     - Input . Number of "positive" events in the      - Integer
!                     population (0 <= M <= NPOP)
!     NPOP  - Input . Population size (NPOP > 0)              - Integer
!     E     - Input . Accuracy (1E-14 <= E < 1)               - Real
!     IER   - Output. Return code :                           - Integer
!                     0 = normal
!                     1 = invalid input argument
!
!     External functions called:
!       INVJM
!
!     Fortran functions called:
!       MAX
!
!     Uses Berkopec's alogorithm:
!     A. Berkopec, HyperQuick alogorithm for discrete hypergeometric
!     distribution. Journal of Discrete Algorithms, 2007, 5, 341-347,
!
!------------------------------------------------------------------------

  implicit none

  ! Function
  ! --------

  real(kind=8) :: hypergcdf

  ! Arguments
  ! ---------

  real(kind=8), intent(in) :: e
  integer, intent(in) :: x, n, m, npop
  integer, intent(out) :: ier

  ! Local declarations
  ! ------------------

  real(kind=8), external :: invjm

  real(kind=8), parameter :: zero=0.0_8, half=0.5_8, one=1.0_8
  real(kind=8), parameter :: emx=1.0e-14_8   ! maximum accuracy
```

```
real(kind=8) :: ak, bk, ck, ee, epsk, jjm, s
integer :: k

!------------------------------------------------------------------

! Test for valid input arguments

if ( npop <= 0 .or. m < 0 .or. n <= 0 .or. n > npop .or.  &
     x < 0 .or. x > n .or. e <= zero .or. e >= one ) then
   ier = 1
   hypergcdf = -one
   return
end if

hypergcdf = zero
ier = 0

! Special cases

if ( m == 0 ) then
   hypergcdf = one
   return
else if ( m == npop .and. x < m ) then
   return
else if ( m == npop .and. x == m ) then
   hypergcdf = one
   return
else if ( n == npop ) then
   if ( x >= m ) hypergcdf = one
   return
end if

! General case

s = one
do k = x, m-2
   s = s*invjm(n,x,npop,k) + one
end do
ak = s
bk = s
k = m - 2
ee = max( e, emx )
epsk = ee + ee
do
   if ( k >= npop-(n-x)-1 .or. epsk <= ee ) exit
   ck = ak/bk
   k = k + 1
   jjm = invjm(n,x,npop,k)
   bk = bk*jjm + one
   ak = ak*jjm
   epsk = (npop-(n-x)-1-k)*(ck-ak/bk)
end do
hypergcdf = one - (ak/bk-epsk*half)
```

```
  end function hypergcdf
  function invjm( n, x, npop, m )
    real(kind=8) :: invjm
    integer, intent(in) :: n, x, npop, m
    real(kind=8), parameter :: one=1.0_8

    invjm = (one-x/(m+one))/(one-(n-x-one)/(npop-m-one))
  end function invjm
```

Chapter 3
Negative binomial distribution

Function nbinomcdf(k, r, p, ier)
File : nbinomcdf.f90

```
function nbinomcdf( k, r, p, ier )

!-----------------------------------------------------------------------
!
!    Computes the probability that a random variable distributed
!    according to the negative binomial distribution with R successes
!    and probability of success P, is less than or equal to K.
!
!    K     - Input . Value of the variable: number of        - Integer
!                  failures (K >= 0)
!    R     - Input . Number of successes (R > 0)             - Integer
!    P     - Input . Probability of success (0<= P <=1)      - Real
!    IER   - Output. Return code :                           - Integer
!                  0 = normal
!                  1 = invalid input argument
!                  3 = result out of limits in BETACDF
!
!    External functions called:
!      BETACDF (and DLGAMA if not in Fortran library)
!
!    Fortran functions called:
!      NONE
!
!-----------------------------------------------------------------------

   implicit none

   ! Function
   ! --------

   real(kind=8) :: nbinomcdf

   ! Arguments
   ! ---------

   real(kind=8), intent(in) :: p
   integer, intent(in) :: k, r
   integer, intent(out) :: ier

   ! Local declarations
   ! ------------------

   real(kind=8), external :: betacdf

   real(kind=8), parameter :: zero=0.0_8, one=1.0_8
   real(kind=8) :: t

   !-----------------------------------------------------------------------

   nbinomcdf = zero
   ier = 0

   ! Test for valid input arguments
```

```
    if ( r <= 0 .or. k < 0 .or. p <= zero .or. p > one ) then
      ier = 1
      return
    end if

    ! Special case

    if ( k == 0 .and. p == one ) then
      nbinomcdf = one
      return
    end if

    ! General case

    t = r
    nbinomcdf = betacdf( p, t, k+one, ier )

  end function nbinomcdf
```

Chapter 4
Poisson distribution

Function poissoncdf(x, p, ier)
File : poissoncdf.f90

```
function poissoncdf( x, p, ier )

!-----------------------------------------------------------------------
!
!     Computes the probability that a random variable distributed
!     according to the poisson distribution with parameter P, is
!     less than or equal to X.
!
!     X     - Input . Value of the variable          (X >= 0)  - Integer
!     P     - Input . Parameter                      (P >= 0)  - Real
!     IER   - Output. Return code :                            - Integer
!                   0 = normal
!                   1 = invalid input argument
!                       (then POISSONCDF = 0.)
!
!     External functions called:
!       CHI2CDF
!
!     Fortran functions called:
!       EXP  REAL
!
!-----------------------------------------------------------------------

   implicit none

   ! Function
   ! --------

   real(kind=8) :: poissoncdf

   ! Arguments
   ! ---------

   real(kind=8), intent(in) :: p
   integer, intent(in) :: x
   integer, intent(out) :: ier

   ! Local declarations
   ! ------------------

   real(kind=8), external :: chi2cdf

   real(kind=8), parameter :: zero=0.0_8, one=1.0_8
   real(kind=8), parameter :: dflimit=1.0e6_8

   real(kind=8) :: y

   !-----------------------------------------------------------------------

   if ( x < 0 .or. p < zero ) then  ! Test for valid input arguments
      ier = 1
      poissoncdf = zero
      return
```

```
    end if

    ier = 0
    if ( p <= zero ) then
      poissoncdf = one
    else
      if ( x == 0 ) then
        poissoncdf = exp( -p )
      else
        y = real(x+x+2,kind=8)
        poissoncdf = one - chi2cdf( p+p, y, dflimit, ier )
      end if
    end if

  end function poissoncdf
```

Part II
INVERSE DISTRIBUTION FUNCTIONS

Chapter 5
Binomial distribution

Function ibinom(pbin, n, p, ier)
File : ibinom.f90

Chapter 6
Negative binomial distribution

Function inbinom(pnb, r, p, ier)
File : inbinom.f90

Part III
PROBABILITY MASS FUNCTIONS

Chapter 7
Binomial distribution

Function binompmf(m, n, p, ier)
File : binompmf.f90

```
function binompmf( m, n, p, ier )

!-----------------------------------------------------------------------
!
!     Computes the probability that a random variable distributed
!     according to the binomial distribution with sample size N and
!     parameter P, is equal to M (probability mass function).
!
!     M     - Input . Value of the variable (M >= 0 )         - Integer
!     N     - Input . Sample size (N > 0)                     - Integer
!     P     - Input . parameter of the binomial (0 <= P <= 1) - Real
!     IER   - Output. Return code :                           - Integer
!                     0 = normal
!                     1 = invalid input argument
!
!     External functions called:
!       (DLGAMA if not in Fortran library)
!
!     Fortran functions called:
!       DLGAMA
!
!-----------------------------------------------------------------------

   implicit none

   ! Function
   ! --------

   real(kind=8) :: binompmf

   ! Arguments
   ! ---------

   real(kind=8), intent(in) :: p
   integer, intent(in) :: m, n
   integer, intent(out) :: ier

   ! Local declarations
   ! ------------------

!! real(kind=8), external :: dlgama

   real(kind=8), parameter :: zero=0.0_8, one=1.0_8
   real(kind=8), parameter :: explower=-706.893_8
   !     explower = minimum valid argument for the exponential function

   real(kind=8) :: t

   !-----------------------------------------------------------------------

   binompmf = zero
   ier = 0
```

```
    ! Test for valid input arguments

    if ( n <= 0 .or. m < 0 .or. m > n .or. p < zero .or. p > one ) then
      ier = 1
      return
    end if

    ! Special cases

    if ( p == zero ) then
      if ( m == 0 ) binompmf = one
      return
    else if ( p == one ) then
      if ( m == n ) binompmf = one
      return
    end if

    ! General case

    if ( m == 0 ) then
      t = n*log(one-p)
    else if ( m == n ) then
      t = n*log(p)
    else
      t = dlgama(n+one) - dlgama(m+one) - dlgama(n-m+one) + m*log(p) + (n-m)*log(one-p)
    end if
    if ( t > explower ) binompmf = exp(t)
  end function binompmf
```

Chapter 8
Negative binomial distribution

Function nbinompmf(k, r, p, ier)
File : binompmf.f90

```
function nbinompmf( k, r, p, ier )

!-----------------------------------------------------------------------
!
!     Computes the probability that a random variable distributed
!     according to the negative binomial distribution with R successes
!     and probability of success P, is equal to K
!
!     K     - Input . Value of the variable: number of        - Integer
!                     failures (K >= 0)
!     R     - Input . Number of successes (R > 0)             - Integer
!     P     - Input . Probability of success (0<= P <=1)      - Real
!     IER   - Output. Return code :                           - Integer
!                     0 = normal
!                     1 = invalid input argument
!
!     External functions called:
!       (DLGAMA if not in Fortran library)
!
!     Fortran functions called:
!       DLGAMA
!
!-----------------------------------------------------------------------

   implicit none

   ! Function
   ! --------

   real(kind=8) :: nbinompmf

   ! Arguments
   ! ---------

   real(kind=8), intent(in) :: p
   integer, intent(in) :: k, r
   integer, intent(out) :: ier

   ! Local declarations
   ! ------------------

!! real(kind=8), external :: dlgama

   real(kind=8), parameter :: zero=0.0_8, one=1.0_8
   real(kind=8), parameter :: explower=-706.893_8
   !     explower = minimum valid argument for the exponential function

   real(kind=8) :: t, xr

   !-----------------------------------------------------------------------

   nbinompmf = zero
   ier = 0
```

```
  ! Test for valid input arguments

  if ( r <= 0 .or. k < 0 .or. p < zero .or. p > one ) then
    ier = 1
    nbinompmf = -one
    return
  end if

  ! Special cases

  if ( p == zero ) then
    return
  else if ( p == one ) then
    if ( k == 0 ) nbinompmf = one
    return
  end if

  ! General case

  xr = r
  if ( k == 0 ) then
    t = xr*log(p)
  else
    t = dlgama(xr+k) - dlgama(k+one) - dlgama(xr) + xr*log(p) + k*log(one-p)
  end if
  if ( t > explower ) nbinompmf = exp(t)

end function nbinompmf
```

Chapter 9
Poisson distribution

Function poissonpmf(m, p, ier)
File : poissonpmf.f90

```
function poissonpmf( m, p, ier )

!-----------------------------------------------------------------------
!
!     Computes the probability that a random variable distributed
!     according to the poisson distribution with parameter P,
!     is equal to M (probability mass function).
!
!     M     - Input . Value of the variable (M >= 0 )         - Integer
!     P     - Input . parameter of the poisson (P >= 0)       - Real
!     IER   - Output. Return code :                           - Integer
!                     0 = normal
!                     1 = invalid input argument
!
!     External functions called:
!       (DLGAMA if not in Fortran library)
!
!     Fortran functions called:
!       DLGAMA  LOG
!
!-----------------------------------------------------------------------

   implicit none

   ! Function
   ! --------

   real(kind=8) :: poissonpmf

   ! Arguments
   ! ---------

   real(kind=8), intent(in) :: p
   integer, intent(in) :: m
   integer, intent(out) :: ier

   ! Local declarations
   ! ------------------

!! real(kind=8), external :: dlgama

   real(kind=8), parameter :: zero=0.0_8, one=1.0_8
   real(kind=8), parameter :: explower=-706.893_8
   !     explower = minimum valid argument for the exponential function

   real(kind=8) :: t

   !-----------------------------------------------------------------------

   poissonpmf = zero
   ier = 0

   ! Test for valid input arguments
```

```
    if ( m < 0 .or. p < zero ) then
       ier = 1
       return
    end if

    if ( p == zero ) then
       if ( m == 0 ) poissonpmf = one
    else if ( m == 0 ) then
       poissonpmf = exp( -p )
    else
       t = m*log(p) - dlgama(m+one) - p
       if ( t > explower ) poissonpmf = exp(t)
    end if

  end function poissonpmf
```

Part IV
RANDOM GENERATORS

Chapter 10
Binomial

Function bindev(n, prob, idum, ier)
File : bindev.f90

```
function bindev( n, prob, idum, ier )

!------------------------------------------------------------------------
!
!    Returns an integer random variable distributed according to a
!    binomial distribution with parameters n and prob.
!
!    n     - Input . Parameter of the binomial distribution  - Integer
!                    The number of trials in the binomial
!                    distribution (0.0 < n)
!    prob  - Input . Parameter of the binomial distribution  - Real
!                    The probability of an event in each
!                    trial of the binomial distribution
!                    (0.0 < prob < 1.0)
!    idum  - Input . A negative integer to initialize the    - Integer
!            Output. random sequence. Thereafter do not
!                    alter this argument between successive
!                    deviates in a sequence
!    ier   - Output. Return code:                            - Integer
!                    0 = normal
!                    1 = invalid input argument
!                       (then bindev = -1)
!
!    External functions called:
!      RAN1
!    Fortran  functions called:
!      ABS  LOG  MIN  ABS  INT  SQRT
!
!------------------------------------------------------------------------
!
!    INTEGER FUNCTION IGNBIN( N, P )
!
!                GENerate BINomial random deviate
!
!
!                        Function
!
!
!    Generates a single random deviate from a binomial
!    distribution whose number of trials is N and whose
!    probability of an event in each trial is P.
!
!
!                        Arguments
!
!
!    N  --> The number of trials in the binomial distribution
!           from which a random deviate is to be generated.
!                        INTEGER N
!
!    P  --> The probability of an event in each trial of the
!           binomial distribution from which a random deviate
```

```
!          is to be generated.
!                    REAL P
!
!    IGNBIN <-- A random deviate yielding the number of events
!               from N independent trials, each of which has
!               a probability of event P.
!                    INTEGER IGNBIN
!
!
!                    Note
!
!
!    Uses RANF so the value of the seeds, ISEED1 and ISEED2 must be set
!    by a call similar to the following
!         DUM = RANSET( ISEED1, ISEED2 )
!
!
!                    Method
!
!
!    This is algorithm BTPE from:
!
!       Kachitvichyanukul, V. and Schmeiser, B. W.
!
!       Binomial Random Variate Generation.
!       Communications of the ACM, 31, 2
!       (February, 1988) 216.
!
!*****************************************************************
!    SUBROUTINE BTPEC(N,PP,ISEED,JX)
!
!    BINOMIAL RANDOM VARIATE GENERATOR
!    MEAN .LT. 30 -- INVERSE CDF
!      MEAN .GE. 30 -- ALGORITHM BTPE:  ACCEPTANCE-REJECTION VIA
!      FOUR REGION COMPOSITION.  THE FOUR REGIONS ARE A TRIANGLE
!      (SYMMETRIC IN THE CENTER), A PAIR OF PARALLELOGRAMS (ABOVE
!      THE TRIANGLE), AND EXPONENTIAL LEFT AND RIGHT TAILS.
!
!    BTPE REFERS TO BINOMIAL-TRIANGLE-PARALLELOGRAM-EXPONENTIAL.
!    BTPEC REFERS TO BTPE AND "COMBINED."  THUS BTPE IS THE
!      RESEARCH AND BTPEC IS THE IMPLEMENTATION OF A COMPLETE
!      USABLE ALGORITHM.
!    REFERENCE:  VORATAS KACHITVICHYANUKUL AND BRUCE SCHMEISER,
!      "BINOMIAL RANDOM VARIATE GENERATION,"
!      COMMUNICATIONS OF THE ACM, FORTHCOMING
!    WRITTEN:  SEPTEMBER 1980.
!      LAST REVISED:  MAY 1985, JULY 1987
!    REQUIRED SUBPROGRAM:  RAND() -- A UNIFORM (0,1) RANDOM NUMBER
!                          GENERATOR
!    ARGUMENTS
!
!      N : NUMBER OF BERNOULLI TRIALS           (INPUT)
```

```
!     PP : PROBABILITY OF SUCCESS IN EACH TRIAL (INPUT)
!     ISEED:  RANDOM NUMBER SEED              (INPUT AND OUTPUT)
!     JX:  RANDOMLY GENERATED OBSERVATION      (OUTPUT)
!
!    VARIABLES
!     PSAVE: VALUE OF PP FROM THE LAST CALL TO BTPEC
!     NSAVE: VALUE OF N FROM THE LAST CALL TO BTPEC
!     XNP:  VALUE OF THE MEAN FROM THE LAST CALL TO BTPEC
!
!     P: PROBABILITY USED IN THE GENERATION PHASE OF BTPEC
!     FFM: TEMPORARY VARIABLE EQUAL TO XNP + P
!     M:  INTEGER VALUE OF THE CURRENT MODE
!     FM:  FLOATING POINT VALUE OF THE CURRENT MODE
!     XNPQ: TEMPORARY VARIABLE USED IN SETUP AND SQUEEZING STEPS
!     P1:  AREA OF THE TRIANGLE
!     C:  HEIGHT OF THE PARALLELOGRAMS
!     XM:  CENTER OF THE TRIANGLE
!     XL:  LEFT END OF THE TRIANGLE
!     XR:  RIGHT END OF THE TRIANGLE
!     AL:  TEMPORARY VARIABLE
!     XLL:  RATE FOR THE LEFT EXPONENTIAL TAIL
!     XLR:  RATE FOR THE RIGHT EXPONENTIAL TAIL
!     P2:  AREA OF THE PARALLELOGRAMS
!     P3:  AREA OF THE LEFT EXPONENTIAL TAIL
!     P4:  AREA OF THE RIGHT EXPONENTIAL TAIL
!     U:  A U(0,P4) RANDOM VARIATE USED FIRST TO SELECT ONE OF THE
!        FOUR REGIONS AND THEN CONDITIONALLY TO GENERATE A VALUE
!        FROM THE REGION
!     V:  A U(0,1) RANDOM NUMBER USED TO GENERATE THE RANDOM VALUE
!        (REGION 1) OR TRANSFORMED INTO THE VARIATE TO ACCEPT OR
!        REJECT THE CANDIDATE VALUE
!     IX:  INTEGER CANDIDATE VALUE
!     X:  PRELIMINARY CONTINUOUS CANDIDATE VALUE IN REGION 2 LOGIC
!        AND A FLOATING POINT IX IN THE ACCEPT/REJECT LOGIC
!     K:  ABSOLUTE VALUE OF (IX-M)
!     F:  THE HEIGHT OF THE SCALED DENSITY FUNCTION USED IN THE
!        ACCEPT/REJECT DECISION WHEN BOTH M AND IX ARE SMALL
!        ALSO USED IN THE INVERSE TRANSFORMATION
!     R: THE RATIO P/Q
!     G: CONSTANT USED IN CALCULATION OF PROBABILITY
!     MP:  MODE PLUS ONE, THE LOWER INDEX FOR EXPLICIT CALCULATION
!         OF F WHEN IX IS GREATER THAN M
!     IX1:  CANDIDATE VALUE PLUS ONE, THE LOWER INDEX FOR EXPLICIT
!          CALCULATION OF F WHEN IX IS LESS THAN M
!     I:  INDEX FOR EXPLICIT CALCULATION OF F FOR BTPE
!     AMAXP: MAXIMUM ERROR OF THE LOGARITHM OF NORMAL BOUND
!     YNORM: LOGARITHM OF NORMAL BOUND
!     ALV:  NATURAL LOGARITHM OF THE ACCEPT/REJECT VARIATE V
!
!     X1,F1,Z,W,Z2,X2,F2, AND W2 ARE TEMPORARY VARIABLES TO BE
!     USED IN THE FINAL ACCEPT/REJECT TEST
!
```

```
!      QN: PROBABILITY OF NO SUCCESS IN N TRIALS
!
!    REMARK
!      IX AND JX COULD LOGICALLY BE THE SAME VARIABLE, WHICH WOULD
!      SAVE A MEMORY POSITION AND A LINE OF CODE.  HOWEVER, SOME
!      COMPILERS (E.G.,CDC MNF) OPTIMIZE BETTER WHEN THE ARGUMENTS
!      ARE NOT INVOLVED.
!
!    ISEED NEEDS TO BE DOUBLE PRECISION IF THE IMSL ROUTINE
!    GGUBFS IS USED TO GENERATE UNIFORM RANDOM NUMBER, OTHERWISE
!    TYPE OF ISEED SHOULD BE DICTATED BY THE UNIFORM GENERATOR
!
!--------------------------------------------------------------------
!
!    March, 2003: changed to Fortran90 and
!    ranf changed to ran1
!
!--------------------------------------------------------------------

   implicit none

   ! Function
   ! --------

   integer :: bindev

   ! Arguments
   ! ---------

   real(kind=8), intent(in) :: prob
   integer, intent(in) :: n
   integer, intent(inout) :: idum
   integer, intent(out) :: ier

   ! Local declarations
   ! ------------------

   real, external :: ran1
   integer, save :: nsave=-1
   integer :: i, ix, ix1, k, m, mp
   real, save :: psave=-1.0
   real :: al, alv, amaxp, c, f, f1, f2, ffm, fm, g
   real :: p, pp, p1, p2, p3, p4, q, qn, r, u, v, w, w2
   real :: x, x1, x2, xl, xll, xlr, xm, xnp, xnpq, xr, ynorm, z, z2

   !--------------------------------------------------------------------

   pp = prob
   if ( n <= 0 .or. pp <= 0.0 .or. pp >= 1.0 ) then
      ier = 1
      bindev = -1
      return
   end if
```

```
ier = 0

!**** Determine appropriate algorithm and whether setup is necessary

if ( pp /= psave .or. n /= nsave ) then

  !**** Setup, perform only when parameters change

  psave = pp
  p = min(psave,1.-psave)
  q = 1. - p
  xnp = n*p
  nsave = n
  if ( xnp < 30. ) then   !  Inverse cdf logic for mean less than 30
    qn = q**n
    r = p/q
    g = r*(n+1)
  else
    ffm = xnp + p
    m = ffm
    fm = m
    xnpq = xnp*q
    p1 = int(2.195*sqrt(xnpq)-4.6*q) + 0.5
    xm = fm + 0.5
    xl = xm - p1
    xr = xm + p1
    c = 0.134 + 20.5/ (15.3+fm)
    al = (ffm-xl)/ (ffm-xl*p)
    xll = al* (1.+.5*al)
    al = (xr-ffm)/ (xr*q)
    xlr = al* (1.+.5*al)
    p2 = p1* (1.+c+c)
    p3 = p2 + c/xll
    p4 = p3 + c/xlr
  end if
end if

if ( xnp >= 30. ) then

  !**** Generate variate

  do
    u = ran1(idum)*p4
    v = ran1(idum)

    if ( u <= p1 ) then        !  Triangular region
      ix = xm - p1*v + u
      goto 10
    else if ( u <= p2 ) then   !  Parallelogram region
      x = xl + (u-p1)/c
      v = v*c + 1. - abs(xm-x)/p1
      if ( v > 1. .or. v <= 0. ) cycle
      ix = x
```

```
else if ( u <= p3 ) then   ! Left tail
  ix = xl + log(v)/xll
  if ( ix < 0 ) cycle
  v = v* (u-p2)*xll
else                       ! Right tail
  ix = xr - log(v)/xlr
  if ( ix > n ) cycle
  v = v* (u-p3)*xlr
end if

!**** Determine appropriate way to perform accept/reject test

k = abs(ix-m)
if ( k > 20 .and. k < xnpq/2-1 ) then

  ! Squeezing using upper and lower bounds on log(f(x))
  amaxp = (k/xnpq)* ((k* (k/3.+.625)+.1666666666666)/xnpq+.5)
  ynorm = -k*k/ (2.*xnpq)
  alv = log(v)
  if ( alv < ynorm-amaxp ) goto 10
  if ( alv > ynorm+amaxp ) cycle

  ! Stirling's formula to machine accuracy for
  ! the final acceptance/rejection test
  x1 = ix + 1
  f1 = fm + 1.
  z = n + 1 - fm
  w = n - ix + 1.
  z2 = z*z
  x2 = x1*x1
  f2 = f1*f1
  w2 = w*w
  if ( alv > (xm*log(f1/x1)+ (n-m+.5)*log(z/w)+                    &
           (ix-m)*log(w*p/ (x1*q))+ (13860.- (462.- (132.- (99.- &
     140./f2)/f2)/f2)/f2)/f1/166320.+ (13860.- (462.- (132.- (99.- &
     140./z2)/z2)/z2)/z2)/z /166320.+ (13860.- (462.- (132.- (99.- &
     140./x2)/x2)/x2)/x2)/x1/166320.+ (13860.- (462.- (132.- (99.- &
     140./w2)/w2)/w2)/w2)/w /166320.) ) cycle
  goto 10
end if

! Explicit evaluation

f = 1.0
r = p/q
g = (n+1)*r
if ( m < ix ) then
  mp = m + 1
  do i = mp,ix
    f = f* (g/i-r)
  end do
else if ( m > ix ) then
```

```
            ix1 = ix + 1
            do i = ix1,m
               f = f/ (g/i-r)
            end do
         end if
         if ( v <= f ) goto 10
      end do

   else

      do
         ix = 0
         f = qn
         u = ran1(idum)
         if ( u < f ) goto 10
         do ix = 1, 111
            u = u - f
            f = f*(g/ix-r)
            if ( u < f ) goto 10
         end do
      end do

   end if

   10 continue
   if ( psave > 0.5 ) ix = n - ix
   bindev = ix

end function bindev
```

Chapter 11
Poisson

Function poissdev(av, idum, ier)
File : poissdev.f90

```
function poissdev( av, idum, ier )

!-----------------------------------------------------------------------
!
!     Returns an integer random variable distributed according to a
!     Poisson distribution with parameter av.
!
!     av    - Input . Parameter of the Poisson distribution   - Real
!                   (0.0 < av <= 2.0e9)
!     idum  - Input . A negative integer to initialize the    - Integer
!             Output. random sequence. Thereafter do not
!                   alter this argument between successive
!                   deviates in a sequence
!     ier   - Output. Return code:                            - Integer
!                   0 = normal
!                   1 = invalid input argument av
!                      (then poissdev = -1)
!
!     External functions called:
!       EXPDV1  GAUSSBM1  RAN1
!     Fortran  functions called:
!       ABS  LOG  EXP  INT  MAX  MIN  REAL  SIGN  SQRT
!
!-----------------------------------------------------------------------
!
!     INTEGER FUNCTION IGNPOI( AV )
!
!                GENerate POIsson random deviate
!
!
!                        Function
!
!
!     Generates a single random deviate from a Poisson
!     distribution with mean AV.
!
!
!                        Arguments
!
!
!     AV --> The mean of the Poisson distribution from which
!            a random deviate is to be generated.
!                        REAL AV
!
!     GENEXP <-- The random deviate.
!                        REAL GENEXP
!
!
!                        Method
!
!
!     Renames KPOIS from TOMS as slightly modified by BWB to use RANF
```

```
!     instead of SUNIF.
!
!     For details see:
!
!               Ahrens, J.H. and Dieter, U.
!               Computer Generation of Poisson Deviates
!               From Modified Normal Distributions.
!               ACM Trans. Math. Software, 8, 2
!               (June 1982),163-179
!
!**********************************************************************
!
!
!     P O I S S O N  DISTRIBUTION
!
!
!**********************************************************************
!
!     FOR DETAILS SEE:
!
!               AHRENS, J.H. AND DIETER, U.
!               COMPUTER GENERATION OF POISSON DEVIATES
!               FROM MODIFIED NORMAL DISTRIBUTIONS.
!               ACM TRANS. MATH. SOFTWARE, 8,2 (JUNE 1982), 163 - 179.
!
!     (SLIGHTLY MODIFIED VERSION OF THE PROGRAM IN THE ABOVE ARTICLE)
!
!**********************************************************************
!
!      INTEGER FUNCTION IGNPOI(IR,MU)
!
!     INPUT:  IR=CURRENT STATE OF BASIC RANDOM NUMBER GENERATOR
!             MU=MEAN MU OF THE POISSON DISTRIBUTION
!     OUTPUT: IGNPOI=SAMPLE FROM THE POISSON-(MU)-DISTRIBUTION
!
!     MUPREV=PREVIOUS MU, MUOLD=MU AT LAST EXECUTION OF STEP P OR B.
!     TABLES: COEFFICIENTS A0-A7 FOR STEP F. FACTORIALS FACT
!     COEFFICIENTS A(K) - FOR PX = FK*V*V*SUM(A(K)*V**K)-DEL
!
!-----------------------------------------------------------------------
!
!     March, 2003: changed to Fortran90 and
!     sexpo, snorm, ranf changed to expdv1, gaussbm1, ran1
!
!-----------------------------------------------------------------------

    implicit none

    ! Function
    ! --------

    integer :: poissdev
```

```
! Arguments
! ---------

real(kind=8), intent(in) :: av
integer, intent(inout) :: idum
integer, intent(out) :: ier

! Local declarations
! ------------------

real, external :: ran1, expdv1, gaussbm1
real, parameter :: a0=-.5,a1= .3333333,a2=-.2500068,a3= .2000118,a4=-.1661269, &
                   a5= .1421878,a6=-.1384794,a7= .1250060
real :: b1, b2, c, c0, c1, c2, c3, d, del, difmuk, e
real :: fk, fx, fy, g, mu, omega, p, p0, px, py, q, s, t, u, v, x, xx
real :: muprev=0., muold=0.
integer :: j, k, l, m
logical :: kflag, lflag

real :: fact(10)=(/ 1., 1., 2., 6., 24., 120., 720., 5040., 40320., 362880./)
real :: pp(35)
save

!-----------------------------------------------------------------

mu = av
ier = 0
if ( mu <= 0.0 .or. mu > 2.0e9 ) then
  ier = 1
  poissdev = -1
  return
end if

! Separation of cases a and b

if ( mu == muprev .or. mu >= 10.0 ) then

  ! Case  a. (recalculation of s, d, l if mu has changed)

  if ( mu /= muprev ) then
    muprev = mu
    s = sqrt(mu)
    d = 6.0*mu*mu
      ! The poisson probabilities pk exceed the discrete normal
      ! probabilities fk whenever k >= m(mu). l=int(mu-1.1484)
      ! is an upper bound to m(mu) for all mu >= 10 .
    l = int(mu-1.1484)
  end if

  ! Step n. Normal sample - gaussbm1(idum) for standard normal deviate

  g = mu + s*gaussbm1(idum)
  if ( g >= 0.0 ) then
```

```
      poissdev = int(g)

      !  Step i. Immediate acceptance if poissdev is large enough

      if ( poissdev >= l ) return

      !  Step s. Squeeze acceptance - ran1(idum) for (0,1)-sample u

      fk = real(poissdev)
      difmuk = mu - fk
      u = ran1(idum)
      if ( d*u >= difmuk*difmuk*difmuk ) return
    end if

    !  Step p. Preparations for steps q and h.
    !          (recalculations of parameters if necessary)
    !          .3989423=(2*pi)**(-.5)  .416667e-1=1./24.  .1428571=1./7.
    !          the quantities b1, b2, c3, c2, c1, c0 are for the hermite
    !          approximations to the discrete normal probabilities fk.
    !          c=.1069/mu guarantees majorization by the 'hat'-function.

    if ( mu /= muold ) then
      muold = mu
      omega = .3989423/s
      b1 = .4166667e-1/mu
      b2 = .3*b1*b1
      c3 = .1428571*b1*b2
      c2 = b2 - 15.*c3
      c1 = b1 - 6.*b2 + 45.*c3
      c0 = 1. - b1 + 3.*b2 - 15.*c3
      c = .1069/mu
    end if
    if ( g >= 0.0 ) then
      !  'subroutine' f is called (kflag=T for correct test)
      kflag = .true.
      lflag = .false.
    else
      lflag = .true.
    end if
    do
      if ( lflag ) then

        !  Step e. Exponential sample - expdv1(idum) for standard exponential
        !          deviate e and sample t from the laplace 'hat'
        !          (if t <= -.6744 then pk < fk for all mu >= 10.)

        e = expdv1(idum)
        u = ran1(idum)
        u = u + u - 1.0
        t = 1.8 + sign(e,u)
        if ( t <= (-.6744) ) cycle
        poissdev = int(mu+s*t)
        fk = real(poissdev)
```

```
      difmuk = mu - fk
      !  'subroutine' f is called (kflag=F for correct test)
      kflag = .false.
    else
      lflag = .true.
    end if

    !  Step f. 'subroutine' f. Calculation of px, py, fx, fy.

    if ( poissdev < 10 ) then
        !    Case poissdev < 10 uses factorials from table fact
      px = -mu
      py = mu**poissdev/fact(poissdev+1)
    else
        !    Case poissdev  >=  10 uses polynomial approximation
        !    a0-a7 for accuracy when advisable
        !    .8333333e-1=1./12.  .3989423=(2*pi)**(-.5)
      del = .8333333e-1/fk
      del = del - 4.8*del*del*del
      v = difmuk/fk
      if ( abs(v) <= 0.25 ) then
        px = fk*v*v* (((((((a7*v+a6)*v+a5)*v+a4)*v+a3)*v+a2)*v+a1)*v+a0) - del
      else
        px = fk*log(1.0+v) - difmuk - del
      end if
      py = .3989423/sqrt(fk)
    end if
    x = (0.5-difmuk)/s
    xx = x*x
    fx = -0.5*xx
    fy = omega* (((c3*xx+c2)*xx+c1)*xx+c0)
    if ( kflag ) then
      !  Step q. Quotient acceptance (rare case)

      if ( fy-u*fy <= py*exp(px-fx) ) return
    else
      !  Step h. Hat acceptance (e is repeated on rejection)

      if ( c*abs(u) <= py*exp(px+e)-fy*exp(fx+e) ) return
    end if
  end do

else

  !  C a s e  b. (start new table and calculate p0 if necessary)

  muprev = 0.0
  if ( mu /= muold ) then
    muold = mu
    m = max(1, int(mu))
    l = 0
    p = exp(-mu)
```

```
      q = p
      p0 = p
    end if

    ! Step u. Uniform sample for inversion method

    do
      u = ran1(idum)
      poissdev = 0
      if ( u <= p0 ) return

      ! Step t. Table comparison until the end pp(l) of the
      !          pp-table of cumulative poisson probabilities
      !          (0.458=pp(9) for mu=10)

      if ( l /= 0 ) then
        j = 1
        if ( u > 0.458 ) j = min(l,m)
        do k = j, l
          if ( u <= pp(k) ) then
            poissdev = k
            return
          end if
        end do
        if ( l == 35 ) cycle
      end if

      ! Step c. Creation of new poisson probabilities p
      !          and their cumulatives q=pp(k)

      l = l + 1
      do k = l, 35
        p = p*mu/real(k)
        q = q + p
        pp(k) = q
        if ( u <= q ) then
          l = k
          poissdev = k
          return
        end if
      end do
      l = 35
    end do

  end if

end function poissdev
```