Jacques POITEVINEAU & Bruno LECOUTRE

STATISTICAL DISTRIBUTIONS FOR BAYESIAN EXPERIMENTAL DATA ANALYSIS

FORTRAN FUNCTIONS
3. MISCELLEANOUS

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

Chapter 1
Chebyshev evaluation

Function chebev(a, b, c, m, x, ier)
File : chebev.f90

```
FUNCTION chebev( a, b, c, m, x, ier )

! Chebyshev evaluation

  implicit none
  REAL(kind=8) :: chebev
  INTEGER, intent(in) :: m
  INTEGER, intent(out) :: ier
  REAL(kind=8), intent(in) :: a, b, x, c(m)
  INTEGER :: j
  REAL(kind=8) :: d, dd, sv, y, y2

  ier = 0
  if ( (x-a)*(x-b) > 0.0_8 ) then      ! x not in range in chebev
    ier = 1
    chebev = 0.0_8
    return
  endif
  d = 0.0_8
  dd = 0.0_8
  y = (2.0_8*x-a-b)/(b-a)
  y2 = 2.0_8*y
  do j = m, 2, -1
    sv = d
    d = y2*d-dd+c(j)
    dd = sv
  end do
  chebev = y*d-dd+0.5_8*c(1)

END FUNCTION chebev
```

Chapter 2
Confluent hypergeometric function 1F1

Function hg1f1m3(x, a, b, ier)
File : hg1f1m3.f90

```
function hg1f1m3( x, a, b, ier )

!----------------------------------------------------------------------
!
!    Computes the confluent hypergeometric function 1F1.
!
!    X     - Input . Value of the variable                - Real
!    A     - Input . 1st (numerator) parameter             - Real
!    B     - Input . 2nd (denominator) parameter           - Real
!                  (B /= 0,-1,-2,...)
!    IER   - Output. Return code:                         - Integer
!                  0 = normal
!                  1 = invalid input argument
!                     (then hg1f1 = 0)
!
!    External functions called:
!      NONE
!
!    Fortran functions called:
!      ABS  EXP  NINT
!
!    See
!    Muller K.E. (2001). Computing the confluent hypergeometric
!       function. Numerische Mathematik, 90, 179-196.
!
!----------------------------------------------------------------------

   implicit none

   ! Function
   ! --------

   real(kind=8) :: hg1f1m3

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

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

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

   real(kind=8), parameter :: zero=0.0_8, one=1.0_8, two=2.0_8
   real(kind=8), parameter :: eps=0.223e-15_8, explower=-706.893_8
   integer, parameter :: maxiter=8   ! maximum number of iterations

   !    These constants are machine dependent:
   !    eps   = machine epsilon
   !          (the smallest real such that 1.0 + eps > 1.0)
   !    explower = minimum valid argument for the exponential function

   real(kind=8) :: aa, d, f1, f2, f3, f4, h
```

```
real(kind=8) :: p, p0, p1, p2, q, q0, q1, q2, r, xx
integer :: i, m

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

  ! Check input
if ( b == zero .or. b == -abs(nint(b)) ) then
  hg1f1m3 = 1.0e+300_8
  ier = 1
  return
end if

hg1f1m3 = zero
ier = 0

  ! Special cases
if ( a == zero .or. x == zero ) then
  hg1f1m3 = one
  return
else if ( abs((a-b)/a) < eps .and. abs(x) > zero ) then
  hg1f1m3 = exp(x)
  return
else if ( abs(a+one) < prec ) then
  hg1f1m3 = one - x/b
  return
else if ( abs(a-b-one) < prec ) then
  hg1f1m3 = (one+x/b)*exp(x)
  return
else if ( abs(a-one) < prec .and. abs(b-two) < prec ) then
  hg1f1m3 = (exp(x)-one)/x
  return
else if ( a == nint(a) .and. a < zero ) then
  m = nint(-a)
  r = one
  h = one
  do i = 1, m
    r = r*(i-1+a)/i/(i-1+b)*x
    h = h + r
  end do
  hg1f1m3 = h
  return
end if

  ! General case: use a rational approximation
if ( x > zero ) then
  aa = a
  xx = x
else
  aa = b - aa
  xx = abs(xx)
end if
d = b - aa
```

```
    x3 = xx*xx*xx
    am3 =
    am2 =
    am1 =
    bm3 =
    bm2 =
    bm1 =
    q0 = one
    q1 = one + xx*(d+one)/(b+b)
    q2 = one + xx*(d+two)/(b+b+two) + xx*xx*(d+one)*(d+two)/(12._8*(b+one))
    p0 = one
    p1 = q1 - xx*d/b
   p2 = q2 -(xx*d/b)*(one+xx*(d+two)/(two*(b+two))) + xx*xx*d*(d+one)/(two*b*(b+one))
    do i = 3, maxit#r
       f1 = (i-2-d)/((4*i-6)*(i-1+b))
       f2 = (i+d)*(i-1+d)*(i-2-d)/(4*(i+i-1)*(i+i-3)*(i-2+b)*(i-1+b))
       f3 = (i-2+d)*(i-1+d)*(i-2+d)/(8*((i+i-3)**2)*(i+i-5)*(i-2+b)*(i-1+b))
       f4 = (i-1+d)*(i-1-b)/(2*(i+i-3)*(i-2+b)*(i-1+b))
       p = (one+f1*xx)*am1 + (f4+f2*xx)*am2 + f3*x3*am3
       q = (one+f1*xx)*bm1 + (f4+f2*xx)*bm2 + f3*x3*bm3
       am1 = am2
       am2 = am3
       am3 =
       bm1 = bm2
       bm2 = bm3
       bm3 =
    end do
    hg1f1m3 = p/q
    if ( x > zero ) hg1f1m3 = hg1f1m3*exp(xx)

  end function hg1f1m3
```

Chapter 3
Confluent hypergeometric function 1F1 (other)

Function hg1f1zj(x, a, b, ier)
File : hg1f1ZJ.f90

```
function hg1f1zj( x, a, b, ier )

!----------------------------------------------------------------------
!
!    Computes the confluent hypergeometric function 1F1.
!
!    X     - Input . Argument                              - Real
!    A     - Input . 1st (numerator) parameter               - Real
!    B     - Input . 2nd (denominator) parameter              - Real
!                  (B /= 0,-1,-2,...)
!    IER   - Output. Return code:                           - Integer
!                  0 = normal
!                  1 = invalid input argument
!                     (then hg1f1 = 0)
!
!    External functions called:
!      GAMMAC
!
!    Fortran functions called:
!      ABS  COS  EXP  INT  NINT
!
!    See
!    Zhang & Jin. (1996). Computation of Special Function. Wiley.
!    Abramowitz M. & Stegun I.A. (1964). Handbook of mathematical
!        mathematical functions.
!
!----------------------------------------------------------------------

   implicit none

   ! Function
   ! --------

   real(kind=8) :: hg1f1zj

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

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

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

   real(kind=8), external :: gammac

   real(kind=8), parameter :: zero=0.0_8, one=1.0_8, two=2.0_8
   real(kind=8), parameter :: pi=3.141592653589793_8
   real(kind=8), parameter :: eps=1.0e-15_8
   integer, parameter :: maxiter=500   ! maximum number of iterations

   real(kind=8) :: a0, aa, hg, hg1, hg2, r, r1, r2, rg
   real(kind=8) :: sum1, sum2, ta, tb, tba, xx, xg, y0, y1
```

```
integer :: i, j, k, la, m, n, nl

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

hg1f1zj = zero
  ! Check input
if ( b == zero .or. b == -abs(nint(b)) ) then
  ier = 1
  return
end if
ier = 0

a0 = a
aa = a
xx = x
if ( b == zero .or. b == -abs(int(b)) ) then
  hg1f1zj = 1.0e+307_8
  return
else if ( aa == zero .or. xx == zero ) then
  hg1f1zj = one
  return
else if ( aa == -one ) then
  hg1f1zj = one - xx/b
  return
else if ( aa == b ) then
  hg1f1zj = exp(xx)
  return
else if ( aa-b == one ) then
  hg1f1zj = (one+xx/b)*exp(xx)
  return
else if ( aa == one .and. b == two ) then
  hg1f1zj = (exp(xx)-one)/xx
  return
else if ( aa == int(aa) .and. aa < zero ) then
  m = int(-aa)
  r = one
  hg = one
  do k = 1, m
    r = r*(aa+k-one)/k/(b+k-one)*xx
    hg = hg + r
  end do
  hg1f1zj = hg
  return
end if

if ( xx < zero ) then
  aa = b - aa
  a0 = aa
  xx = abs(xx)
end if
if ( aa < two) nl = 0
if ( aa >= two ) then
```

```
    nl = 1
    la = int(aa)
    aa = aa - la - one
  end if
  do n = 0, nl
    if ( a0 >= two ) aa = aa + one
    if ( xx <= 30.0_8+abs(b) .or. aa < zero ) then
      hg = one
      rg = one
      do j = 1, maxiter
        rg = rg*(aa+j-one)/(j*(b+j-one))*xx
        hg = hg + rg
        if ( abs(rg/hg) < eps ) exit
      end do
    else
      ta = gammac(aa)
      tb = gammac(b)
      xg = b-aa
      tba = gammac(xg)
      sum1 = one
      sum2 = one
      r1 = one
      r2 = one
      do i = 1, 8
        r1 = -r1*(aa+i-one)*(aa-b+i)/(xx*i)
        r2 = -r2*(b-aa+i-one)*(aa-i)/(xx*i)
        sum1 = sum1 + r1
        sum2 = sum2 + r2
      end do
      hg1 = tb/tba*xx**(-aa)*cos(pi*aa)*sum1
      hg2 = tb/ta*exp(xx)*xx**(aa-b)*sum2
      hg = hg1+hg2
    end if
    if ( n == 0 ) y0 = hg
    if ( n == 1 ) y1 = hg
  end do
  if ( a0 >= two ) then
    do i = 1, la-1
      hg = ((two*aa-b+xx)*y1+(b-aa)*y0)/aa
      y0 = y1
      y1 = hg
      aa = aa + one
    end do
  end if
  if ( x < zero ) hg = hg*exp(x)
  hg1f1zj = hg

end function hg1f1zj
```

Chapter 4
Digamma (psi) function

Function digamma(x)
File : digamma.f90

```
function digamma( x )

!-----------------------------------------------------------------------
!
!     Computes the Digamma (Psi) function.
!
!     X     - Input . Value of the variable                 - Real
!
!     Fortran functions called:
!       ABS  COS  HUGE  INT  LOG  MOD  SIN
!
!     From Zhang S. & Jin J. (1996). Computation of Special Functions.
!          Wiley.
!
!-----------------------------------------------------------------------

  implicit none

  ! Function
  ! --------

  real(kind=8) :: digamma

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

  real(kind=8), intent(in) :: x

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

  real(kind=8), parameter :: zero=0.0_8, half=0.5_8, one=1.0_8, ten=10.0_8
  real(kind=8), parameter :: pi=3.141592653589793_8
  real(kind=8), parameter :: el=0.5772156649015329_8     ! Euler constant
  real(kind=8), parameter :: twol2=1.386294361119891_8   ! 2*log(2)
  real(kind=8), parameter :: a1=-0.8333333333333e-1_8,      &
                             a2= 0.8333333333333333e-2_8,   &
                             a3=-0.3968253968253968e-2_8,   &
                             a4= 0.4166666666666667e-2_8,   &
                             a5=-0.7575757575757576e-2_8,   &
                             a6= 0.2109279609279609e-1_8,   &
                             a7=-0.8333333333333333e-1_8,   &
                             a8= 0.4432598039215686_8
  real(kind=8), parameter :: xlimit=1.0e5_8 ! The limit over which
                                            ! integral and half-integral values
                                            ! are not treated specifically

  real(kind=8) :: pix, ps, s, xa, x2
  integer :: k, n

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

  if ( x <= zero .and. x == int(x) ) then
```

```
    digamma = -huge(x)
    return
  end if

  xa = abs(x)
  s = zero

  if ( xa <= xlimit .and. xa == int(xa) ) then   ! |x| = n <= xlimit
    n = xa
    do k = 1, n-1
      s = s + one/k
    end do
    ps = s - el
  else if ( xa <= xlimit .and. xa+half == int(xa+half) ) then   ! |x| = n+1/2 < xlimit
    n = xa - half
    do k = 1, n
      s = s + one/(k+k-1)
    end do
    ps = s + s - el - twol2
  else
    if ( xa < ten ) then   ! When |x| < 10, add an integer to
      n = 10 - int(xa)     ! |x| such that |x| + n > 10
      do k = 0, n-1
        s = s + one/(xa+k)
      end do
      xa = xa + n
    end if
    x2 = one/(xa*xa)
      ! Calculate Psi(|x|+n)
   ps = log(xa) - half/xa + x2*(((((((a8*x2+a7)*x2+a6)*x2+a5)*x2+a4)*x2+a3)*x2+a2)*x2+a1)
      ! Calculate Psi(|x|)
    ps = ps - s
  end if
    ! Calculate Psi(x)
  if ( x < zero ) then
    pix = mod(pi*x,pi)
    ps = ps - pi*cos(pix)/sin(pix) - one/x
  end if
  digamma = ps

end function digamma
```

Chapter 5
Gamma (complete) function

Function dgamma(x)
File : dgamma.f90

```
   function dgamma( x )

!----------------------------------------------------------------------
!
!     Returns the complete gamma function
!
!     X     - Input . Argument ( X < XBIG )                  - Real
!
!----------------------------------------------------------------------
!
! This routine calculates the GAMMA function for a real argument X.
!   Computation is based on an algorithm outlined in reference 1.
!   The program uses rational functions that approximate the GAMMA
!   function to at least 20 significant decimal digits.  Coefficients
!   for the approximation over the interval (1,2) are unpublished.
!   Those for the approximation for X .GE. 12 are from reference 2.
!   The accuracy achieved depends on the arithmetic system, the
!   compiler, the intrinsic functions, and proper selection of the
!   machine-dependent constants.
!
!*******************************************************************
!
! Explanation of machine-dependent constants
!
! beta   - radix for the floating-point representation
! maxexp - the smallest positive power of beta that overflows
! XBIG   - the largest argument for which GAMMA(X) is representable
!          in the machine, i.e., the solution to the equation
!                  GAMMA(XBIG) = beta**maxexp
! XINF   - the largest machine representable floating-point number;
!          approximately beta**maxexp
! EPS    - the smallest positive floating-point number such that
!          1.0+EPS .GT. 1.0
! XMININ - the smallest positive floating-point number such that
!          1/XMININ is machine representable
!
!     Approximate values for some important machines are:
!
!                          beta       maxexp        XBIG
!
! CRAY-1         (S.P.)        2         8191        966.961
! Cyber 180/855
!   under NOS    (S.P.)        2         1070        177.803
! IEEE (IBM/XT,
!   SUN, etc.)   (S.P.)        2          128        35.040
! IEEE (IBM/XT,
!   SUN, etc.)   (D.P.)        2         1024        171.624
! IBM 3033       (D.P.)       16           63        57.574
! VAX D-Format   (D.P.)        2          127        34.844
! VAX G-Format   (D.P.)        2         1023        171.489
!
!                          XINF         EPS        XMININ
```

```
!
! CRAY-1         (S.P.)   5.45E+2465   7.11E-15    1.84E-2466
! Cyber 180/855
!   under NOS    (S.P.)   1.26E+322    3.55E-15    3.14E-294
! IEEE (IBM/XT,
!   SUN, etc.)   (S.P.)   3.40E+38     1.19E-7     1.18E-38
! IEEE (IBM/XT,
!   SUN, etc.)   (D.P.)   1.79D+308    2.22D-16    2.23D-308
! IBM 3033       (D.P.)   7.23D+75     2.22D-16    1.39D-76
! VAX D-Format   (D.P.)   1.70D+38     1.39D-17    5.88D-39
! VAX G-Format   (D.P.)   8.98D+307    1.11D-16    1.12D-308
!
!*****************************************************************
!
! Error returns
!
!  The program returns the value XINF for singularities or
!     when overflow would occur.  The computation is believed
!     to be free of underflow and overflow.
!
!  Intrinsic functions required are:
!
!     INT, DBLE, EXP, LOG, REAL, SIN
!
! References: "An Overview of Software Development for Special
!              Functions", W. J. Cody, Lecture Notes in Mathematics,
!              506, Numerical Analysis Dundee, 1975, G. A. Watson
!              (ed.), Springer Verlag, Berlin, 1976.
!
!              Computer Approximations, Hart, Et. Al., Wiley and
!              sons, New York, 1968.
!
!  Latest modification: October 12, 1989
!
!  Authors: W. J. Cody and L. Stoltz
!           Applied Mathematics Division
!           Argonne National Laboratory
!           Argonne, IL 60439
!
!-----------------------------------------------------------------------

implicit none

!  Function
!  --------

real(kind=8) :: dgamma

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

real(kind=8), intent(in) :: x
```

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

integer :: i, n
logical :: parity

! Mathematical constants:
real(kind=8), parameter :: zero=0.0_8, half=0.5_8, one=1.0_8, two=2.0_8, twelve=12.0_8
real(kind=8), parameter :: pi=3.14159265358979323846264338_8
real(kind=8), parameter :: sqrtpi=0.91893853320467274178032973_8
! Machine dependent parameters:
real(kind=8), parameter :: eps=2.22e-16_8, xbig=171.624_8, xinf=1.79e308_8, xminin=2.23e-308_8

real(kind=8) :: conv, fact, res, sum, xden, xnum, y, y1, ysq, z

! Numerator and denominator coefficients for rational minimax
!     approximation over (1,2):
real(kind=8), dimension(8) :: p=(/                                    &
-1.71618513886549492533811e+0_8, 2.476565080557591991083314e+1_8, &
-3.79804256470945635097577e+2_8, 6.293311553128184426610520e+2_8, &
8.669662027904132112950640e+2_8,-3.145127296884836752543570e+4_8, &
-3.614441341869117298070690e+4_8, 6.645614382024054406278550e+4_8/)
real(kind=8), dimension(8) :: q=(/                                    &
-3.08402300119738975254353e+1_8, 3.153506269796041615291440e+2_8, &
-1.01515636749021914166146e+3_8,-3.107771671572311094404440e+3_8, &
2.253811842098015103301120e+4_8, 4.755846277527881107678150e+3_8, &
-1.346599598649693063924560e+5_8,-1.151322596755534834972110e+5_8/)
! Coefficients for minimax approximation over (12, INF):
real(kind=8), dimension(7) :: c=(/                                    &
-1.910444077728e-03_8,8.4171387781295e-04_8,                      &
-5.952379913043012e-04_8,7.9365079350035024800e-04_8,             &
-2.777777777777681622553e-03_8,8.333333333333333331554247e-02_8, &
5.7083835261e-03_8/)


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

parity = .false.
fact = one
n = 0
y = x
if (y <= zero) then
!------------------------
! Argument is negative
!------------------------
y = -x
y1 = aint(y)
res = y - y1
if (res /= zero) then
if (y1 /= aint(y1*half)*two) parity = .true.
fact = -pi / sin(pi*res)
```

```
y = y + one
else
dgamma = xinf
return
end if
end if
!------------------------
!  Argument is positive
!------------------------
if (y < eps) then
!-----------------
!  Argument < EPS
!-----------------
if (y >= xminin) then
res = one / y
else
dgamma = xinf
return
end if
else if (y < twelve) then
y1 = y
if (y < one) then
!------------------------
!  0.0 < argument < 1.0
!------------------------
z = y
y = y + one
else

          !-----------------------------------------------------
!  1.0 < argument < 12.0, reduce argument if necessary

          !-----------------------------------------------------
n = int(y) - 1
y = y - dble(n)
z = y - one
end if

        !------------------------------------------------
!  Evaluate approximation for 1.0 < argument < 2.0

        !------------------------------------------------
xnum = zero
xden = one
do i = 1, 8
xnum = (xnum + p(i)) * z
xden = xden * z + q(i)
end do
res = xnum / xden + one
if (y1 < y) then
```

```
            !------------------------------------------------
!  Adjust result for case  0.0 < argument < 1.0

            !------------------------------------------------
res = res / y1
else if (y1 > y) then

            !------------------------------------------------
!  Adjust result for case  2.0 < argument < 12.0

            !------------------------------------------------
do i = 1, n
res = res * y
y = y + one
end do
end if
else
!---------------------------------
!  Evaluate for argument >= 12.0
!---------------------------------
if (y <= xbig) then
ysq = y * y
sum = c(7)
do i = 1, 6
sum = sum / ysq + c(i)
end do
sum = sum/y - y + sqrtpi
sum = sum + (y-half)*log(y)
res = exp(sum)
else
dgamma = xinf   ! Error return
return
end if
end if

!  Final adjustments
if (parity) res = -res
if (fact /= one) res = fact / res
dgamma = res

end function dgamma
```

Chapter 6
Gamma (complete) function (other)

Function gammac(x)
File : gammac.f90

```
   function gammac( x )

!------------------------------------------------------------------
!
!     Returns the complete gamma function
!
!     X     - Input . Argument ( X < XBIG )                  - Real
!
!------------------------------------------------------------------
!
! This routine calculates the GAMMA function for a real argument X.
!   Computation is based on an algorithm outlined in reference 1.
!   The program uses rational functions that approximate the GAMMA
!   function to at least 20 significant decimal digits.  Coefficients
!   for the approximation over the interval (1,2) are unpublished.
!   Those for the approximation for X .GE. 12 are from reference 2.
!   The accuracy achieved depends on the arithmetic system, the
!   compiler, the intrinsic functions, and proper selection of the
!   machine-dependent constants.
!
!*******************************************************************
!
! Explanation of machine-dependent constants
!
! beta   - radix for the floating-point representation
! maxexp - the smallest positive power of beta that overflows
! XBIG   - the largest argument for which GAMMA(X) is representable
!          in the machine, i.e., the solution to the equation
!                  GAMMA(XBIG) = beta**maxexp
! XINF   - the largest machine representable floating-point number;
!          approximately beta**maxexp
! EPS    - the smallest positive floating-point number such that
!          1.0+EPS .GT. 1.0
! XMININ - the smallest positive floating-point number such that
!          1/XMININ is machine representable
!
!     Approximate values for some important machines are:
!
!                           beta       maxexp        XBIG
!
! CRAY-1         (S.P.)        2         8191        966.961
! Cyber 180/855
!   under NOS    (S.P.)        2         1070        177.803
! IEEE (IBM/XT,
!   SUN, etc.)   (S.P.)        2          128        35.040
! IEEE (IBM/XT,
!   SUN, etc.)   (D.P.)        2         1024        171.624
! IBM 3033       (D.P.)       16           63        57.574
! VAX D-Format   (D.P.)        2          127        34.844
! VAX G-Format   (D.P.)        2         1023        171.489
!
!                           XINF         EPS        XMININ
```

```
!
! CRAY-1         (S.P.)   5.45E+2465   7.11E-15    1.84E-2466
! Cyber 180/855
!   under NOS    (S.P.)   1.26E+322    3.55E-15    3.14E-294
! IEEE (IBM/XT,
!   SUN, etc.)   (S.P.)   3.40E+38     1.19E-7     1.18E-38
! IEEE (IBM/XT,
!   SUN, etc.)   (D.P.)   1.79D+308    2.22D-16    2.23D-308
! IBM 3033       (D.P.)   7.23D+75     2.22D-16    1.39D-76
! VAX D-Format   (D.P.)   1.70D+38     1.39D-17    5.88D-39
! VAX G-Format   (D.P.)   8.98D+307    1.11D-16    1.12D-308
!
!*****************************************************************
!
! Error returns
!
!  The program returns the value XINF for singularities or
!     when overflow would occur.  The computation is believed
!     to be free of underflow and overflow.
!
!  Intrinsic functions required are:
!
!     INT, DBLE, EXP, LOG, REAL, SIN
!
! References: "An Overview of Software Development for Special
!              Functions", W. J. Cody, Lecture Notes in Mathematics,
!              506, Numerical Analysis Dundee, 1975, G. A. Watson
!              (ed.), Springer Verlag, Berlin, 1976.
!
!              Computer Approximations, Hart, Et. Al., Wiley and
!              sons, New York, 1968.
!
!  Latest modification: October 12, 1989
!
!  Authors: W. J. Cody and L. Stoltz
!           Applied Mathematics Division
!           Argonne National Laboratory
!           Argonne, IL 60439
!
!-----------------------------------------------------------------------

implicit none

!  Function
!  --------

real(kind=8) :: gammac

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

real(kind=8), intent(in) :: x
```

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

integer :: i, n
logical :: parity

! Mathematical constants:
real(kind=8), parameter :: zero=0.0_8, half=0.5_8, one=1.0_8, two=2.0_8, twelve=12.0_8
real(kind=8), parameter :: pi=3.14159265358979323846264338_8
real(kind=8), parameter :: sqrtpi=0.91893853320467274178032973_8
! Machine dependent parameters:
real(kind=8), parameter :: eps=2.22e-16_8, xbig=171.624_8, xinf=1.79e308_8, xminin=2.23e-308_8

real(kind=8) :: conv, fact, res, sum, xden, xnum, y, y1, ysq, z

! Numerator and denominator coefficients for rational minimax
!    approximation over (1,2):
real(kind=8), dimension(8) :: p=(/                                    &
-1.71618513886549492533811e+0_8, 2.476565080557591991083l4e+1_8, &
-3.79804256470945635097577e+2_8, 6.293311553128184426610l2e+2_8, &
8.669662027904132ll295064e+2_8,-3.145127296884836752543l7e+4_8, &
-3.614441341869117298070l9e+4_8, 6.645614382024054406278l5e+4_8/)
real(kind=8), dimension(8) :: q=(/                                    &
-3.084023001197389752543l3e+1_8, 3.153506269796041615291l4e+2_8, &
-1.015156367490219141661l6e+3_8,-3.107771671572311094404l4e+3_8, &
2.253811842098015103301l2e+4_8, 4.755846277527881107678l5e+3_8, &
-1.346599598649693063924l6e+5_8,-1.151322596755534834972l1e+5_8/)
! Coefficients for minimax approximation over (12, INF):
real(kind=8), dimension(7) :: c=(/                                    &
-1.910444077728e-03_8,8.4171387781295e-04_8,                    &
-5.952379913043012e-04_8,7.9365079350035024 8e-04_8,            &
-2.77777777777681622553e-03_8,8.3333333333333333l554247e-02_8, &
5.7083835261e-03_8/)


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

parity = .false.
fact = one
n = 0
y = x
if (y <= zero) then
!------------------------
! Argument is negative
!------------------------
y = -x
y1 = aint(y)
res = y - y1
if (res /= zero) then
if (y1 /= aint(y1*half)*two) parity = .true.
fact = -pi / sin(pi*res)
```

```
y = y + one
else
gammac = xinf
return
end if
end if
!------------------------
!  Argument is positive
!------------------------
if (y < eps) then
!-----------------
!  Argument < EPS
!-----------------
if (y >= xminin) then
res = one / y
else
gammac = xinf
return
end if
else if (y < twelve) then
y1 = y
if (y < one) then
!------------------------
!  0.0 < argument < 1.0
!------------------------
z = y
y = y + one
else

          !-----------------------------------------------------
!  1.0 < argument < 12.0, reduce argument if necessary

          !-----------------------------------------------------
n = int(y) - 1
y = y - dble(n)
z = y - one
end if

        !------------------------------------------------
!  Evaluate approximation for 1.0 < argument < 2.0

        !------------------------------------------------
xnum = zero
xden = one
do i = 1, 8
xnum = (xnum + p(i)) * z
xden = xden * z + q(i)
end do
res = xnum / xden + one
if (y1 < y) then
```

```
            !------------------------------------------------
!  Adjust result for case  0.0 < argument < 1.0

            !------------------------------------------------
res = res / y1
else if (y1 > y) then

            !------------------------------------------------
!  Adjust result for case  2.0 < argument < 12.0

            !------------------------------------------------
do i = 1, n
res = res * y
y = y + one
end do
end if
else
!--------------------------------
!  Evaluate for argument >= 12.0
!--------------------------------
if (y <= xbig) then
ysq = y * y
sum = c(7)
do i = 1, 6
sum = sum / ysq + c(i)
end do
sum = sum/y - y + sqrtpi
sum = sum + (y-half)*log(y)
res = exp(sum)
else
gammac = xinf   ! Error return
return
end if
end if

!  Final adjustments
if (parity) res = -res
if (fact /= one) res = fact / res
gammac = res

end function gammac
```

Chapter 7
Gauss hypergeometric function 2F1

Function hg2f1(x, a, b, c, ier)
File : hg2f1.f90

```
function hg2f1( x, a, b, c, ier )

!------------------------------------------------------------------------
!
!    Computes the Gauss hypergeometric function 2F1.
!
!    X     - Input . Value of the variable (|X| <= 1)        - Real
!                  (|X| < 1 if c < a+b)
!    A     - Input . 1st (numerator) parameter               - Real
!    B     - Input . 2nd (numerator) parameter               - Real
!    C     - Input . 3rd (denominator) parameter             - Real
!                  (C /= 0,-1,-2,...)
!    IER   - Output. Return code:                            - Integer
!                  0 = normal
!                  1 = invalid input argument
!                     (then hg2f1 = 0)
!                  2 = the hypergeometric series is divergent
!                  3 = maximum number of iterations reached
!                     (then hg2f1 = value at last iteration)
!
!    External functions called:
!      DIGAMMA  GAMMAC  DLGAMA if not in FORTRAN library
!
!    Fortran functions called:
!      ABS  EXP  HUGE  LOG  MAX  MIN  MOD  NINT
!
!    See Abramowitz M. & Stegun I.A. (1964).
!       Handbook of mathematical functions.
!
!------------------------------------------------------------------------

   implicit none

   ! Function
   ! --------

   real(kind=8) :: hg2f1

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

   real(kind=8), intent(in) :: x, a, b, c
   integer, intent(out) :: ier

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

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

   real(kind=8), parameter :: zero=0.0_8, half=0.5_8, one=1.0_8, two=2.0_8
   real(kind=8), parameter :: epsil=epsilon(one)
   real(kind=8), parameter :: eps=1.0e-14_8
```

```
real(kind=8), parameter :: xlimit=0.95_8
                        ! xlimit: The limit for x over which Psi function
                        ! expansion is used if c-a-b is an integer number
real(kind=8), parameter :: el=0.5772156649015329_8            ! Euler constant
real(kind=8), parameter :: sqrpi=1.77245385090551603_8        ! sqrt(pi)
real(kind=8), parameter :: sqrpilog=0.572364942924700087_8    ! log(sqrt(pi))
real(kind=8), parameter :: twolog=0.693147180559945309_8      ! log(2)
real(kind=8), parameter :: explower=-706.893_8, expupper=-explower
!     explower = minimum valid argument for the exponential function
!     expupper = maximum valid argument for the exponential function
real(kind=8), parameter :: gamupper=171.624_8
!     gamupper = maximum valid argument for the gamma function
integer, parameter :: limgam=169      ! int(gamupper) - 2

integer, parameter :: maxiter=5000    ! maximum number of iterations

real(kind=8) :: aa, bb, cc, d, e, f, pow, s, t, u, v, xl, xx, y
integer :: i, m
logical :: aneg, bneg, caneg, cbneg, xneg

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

hg2f1 = zero
ier = 0
xx = one - x

  ! Check input
if ( c == zero .or. c == -abs(nint(c)) .or. abs(x) > one .or. &
    xx < eps .and. c-a-b <= zero ) then
  ier = 1
  return
end if

  ! Special cases
aneg = a < zero .and. a == nint(a)
bneg = b < zero .and. b == nint(b)
caneg = c-a <= zero .and. c-a == nint(c-a)
cbneg = c-b <= zero .and. c-b == nint(c-b)
if ( x == zero .or. a == zero .or. b == zero ) then
  hg2f1 = one
  return
else if ( abs(c-a) <= eps ) then
  hg2f1 = xx**(-b)
  return
else if ( abs(c-b) <= eps ) then
  hg2f1 = xx**(-a)
  return
else if ( abs(xx) <= eps .and. c-a-b > zero ) then
  d = max( c, c-a, c-b, c-a-b )
  if ( d <= gamupper ) then
    hg2f1 = gammac(c) / gammac(c-a) * gammac(c-a-b) / gammac(c-b)
  else
```

```
      d = min( c, c-a, c-b, c-a-b )
      if ( d > zero ) then
        s = dlgama(c) + dlgama(c-a-b) - dlgama(c-a) - dlgama(c-b)
        if ( s >= explower .and. s <= expupper ) then
          hg2f1 = exp(s)
        else if ( s > expupper ) then
          hg2f1 = huge(one)
          ier = 2
        end if
      else
        hg2f1 = huge(one)
        ier = 2
      end if
    end if
    return
  else if ( abs(one+x) <= eps .and. abs(c-a+b-one) <= eps ) then
    d = max( c, one+a*half-b, half+half*a )
    if ( d <= gamupper ) then
      hg2f1 = gammac(c) / gammac(one+a*half-b) / gammac( half+half*a) * sqrpi / two**a
    else
      d = min( c, one+a*half-b, half+half*a )
      if ( d > zero ) then
        s = (-a)*twolog+sqrpilog + dlgama(c) - dlgama(one+a*half-b) - dlgama(half+half*a)
        if ( s >= explower .and. s <= expupper ) then
          hg2f1 = exp(s)
        else if ( s > expupper ) then
          hg2f1 = huge(one)
          ier = 2
        end if
      else
        hg2f1 = huge(one)
        ier = 2
      end if
    end if
    return
  else if ( aneg .or. bneg ) then
    if ( aneg ) then
      m = abs(nint(a))
      if ( bneg ) m = min(m,abs(nint(b)))
    else
      m = abs(nint(b))
    end if
    s = one
    t = one
    do i = 0, m-1
      t = t * (a+i)/(c+i)*(b+i)/(i+1)*x
      s = s + t
    end do
    hg2f1 = s
    return
  else if ( caneg .or. cbneg ) then
    if ( caneg ) then
```

```
      m = abs(nint(c-a))
      if ( cbneg ) m = min(m,abs(nint(c-b)))
    else
      m = abs(nint(c-b))
    end if
    s = one
    t = one
    do i = 0, m-1
      t = t * (c-a+i)/(c+i)*(c-b+i)/(i+1)*x
      s = s + t
    end do
    s = xx**(c-a-b) * s
    hg2f1 = s
    return
  end if

  m = nint(c-a-b)

  if ( x <= xlimit .or. abs(c-a-b-m) > eps .or. abs(m) > limgam  &
       .or. max(c,max(a,b)+max(m,0)) > gamupper ) then
      ! Use power series expansion

    xx = x
    aa = a
    bb = b
    cc = c
    xneg = x < zero
    if ( xneg ) then
      xx = x/(x-one)
      if ( a <= b ) then
        pow = -a
        bb = c - b
      else
        pow = -b
        aa = c - a
      end if
    end if
    s = one
    y = one
    do i = 0, maxiter-1
      y = y * (aa+i)/(cc+i)*(bb+i)*xx/(i+1)
      t = s + y
      if ( s == t ) then
        if ( xneg ) s = (one-x)**pow * s
        hg2f1 = s
        return
      end if
      s = t
    end do
    if ( xneg ) s = (one-x)**pow * s
    hg2f1 = s
    ier = 3
```

```
      else
          ! Use Psi function expansion if x exceeds the limit and
          ! m=c-a-b is an integer number

        xl = log(xx)
        d = abs(m)
        if ( m >= 0 ) then
          e = d
          f = zero
          v = one
          if ( m > 0 ) then
            u = xx**d
          else
            u = one
          end if
        else
          e = zero
          f = m
          u = one
          v = xx**f
        end if
        u = gammac(c) / gammac(a+f) / gammac(b+f) * u
        if ( u == zero ) return
       s = (xl+el-digamma(d+one)+digamma(a+e)+digamma(b+e))/gammac(d+one)   ! sum for
i=0
        t = (a+e)*(b+e)/gammac(d+one+one)*xx   ! Pochhammer, factorial and power term for
i=1
        ier = 3
        do i = 1, maxiter
          y = t * (xl-digamma(i+one)-digamma(d+i+one)+digamma(a+e+i)+digamma(b+e+i))
          if ( abs(y) < epsil*abs(s) ) then
            ier = 0
            exit
          end if
          s = s + y
          t = t * (a+e+i)*(b+e+i)/(i+1)*xx/(d+i+1)
        end do
        if ( mod(m,2) == 0 ) then
          u = -u*s
        else
          u = u*s
        end if
        hg2f1 = u
        if ( m == 0 ) return
        v = gammac(c) / gammac(a+e) * gammac(d) / gammac(b+e) * v
        if ( v == zero ) return
        d = one - d
        s = one
        t = one
        do i = 0, abs(m)-2
          t = t * (a+f+i)*(b+f+i)/(i+1)*xx/(d+i)
          s = s + t
```

```
        end do
        hg2f1 = v*s + u

    end if

end function hg2f1
```

Chapter 8
Gauss hypergeometric function 2F1 (other)

Function hypg191(x, a, b, c, ier)
File : hypg191.f90

```
function hypg191( x, a, b, c, ier )

!------------------------------------------------------------------------
!
!     Computes the Gauss hypergeometric function 2F1.
!
!     X     - Input . Value of the variable (|X| <= 1)        - Real
!                   (|X| < 1 if c < a+b)
!     A     - Input . 1st (numerator) parameter               - Real
!     B     - Input . 2nd (numerator) parameter               - Real
!     C     - Input . 3rd (denominator) parameter             - Real
!                   (C /= 0,-1,-2,...)
!     IER   - Output. Return code :                           - Integer
!                   0 = normal
!                   1 = invalid input argument
!                      (then hypg191 = 0)
!                   2 = maximum number of iterations reached
!                      (then hypg191 = value at last iteration)
!
!     Fortran functions called:
!       ABS  NINT
!
!     From Relph A. P. (1963). Algorithm 191 Hypergeometric.
!          Communications of the ACM, 6, 388.
!
!------------------------------------------------------------------------

   implicit none

   ! Function
   ! --------

   real(kind=8) :: hypg191

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

   real(kind=8), intent(in) :: x, a, b, c
   integer, intent(out) :: ier

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

   real(kind=8), parameter :: zero=0.0_8, one=1.0_8
   real(kind=8) :: aa, bb, cc, d, s, t, y, xx
!! real(kind=16) :: aa, bb, cc, d, s, t, y, xx
   integer :: i

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

   ier = 0

   if ( c == zero .or. c == -abs(nint(c)) .or. abs(x) > one .or.  &
```

```
        one-x <  1.0e-15_8 .and. c-a-b <= zero ) then
      hypg191 = zero
      ier = 1                 ! Invalid parameter
      return
    else if ( x == zero ) then
      hypg191 = one
      return
    end if

    aa = a
    bb = b
    cc = c
    xx = x
    s = one
    y = one
!! s = 1.0_16
!! y = 1.0_16
    do i = 0, 5000
      d = (i+1)*((i+cc)**2)
      y = y * ((i+aa)/d)*(i+bb)*(i+cc)*xx
      t = s + y
      if ( s == t ) then
        hypg191 = s
        return
      end if
      s = t
    end do
    hypg191 = s
    ier = 2

  end function hypg191
```

Chapter 9
Gauss hypergeometric function 2F1 (multiple precision)

Function hg2f1fm(x, a, b, c, ier)
File : hg2f1FM.f90

```
function hg2f1fm( x, a, b, c, ier )

!---------------------------------------------------------------------
!!!!!!!!!!!! This the Multiple Precision version of hg2f1     !!!!!!!!!!!
!!!!!!!!!!!! using David Smith's Multiple Precision software !!!!!!!!!!!
!!!!!!!!!!!! alg 814 TOMS, 27, 4, 377-387 (2001)             !!!!!!!!!!!
!---------------------------------------------------------------------
!
!    Computes the Gauss hypergeometric function 2F1.
!
!    X     - Input . Value of the variable (|X| <= 1)        - Real
!                  (|X| < 1 if c < a+b)
!    A     - Input . 1st (numerator) parameter               - Real
!    B     - Input . 2nd (numerator) parameter               - Real
!    C     - Input . 3rd (denominator) parameter             - Real
!                  (C /= 0,-1,-2,...)
!    IER   - Output. Return code:                            - Integer
!                  0 = normal
!                  1 = invalid input argument
!                     (then hg2f1fm = 0)
!                  2 = the hypergeometric series is divergent
!                  3 = maximum number of iterations reached
!                     (then hg2f1fm = value at last iteration)
!
!    External functions called:
!      FMEULR  FMPI  GAMMA  LOG_GAMMA  PSI  TO_FM
!
!    Fortran functions called:
!      ABS  EPSILON  EXP  HUGE  LOG  MAX  MIN  MOD  NINT
!
!    See Abramowitz M. & Stegun I.A. (1964).
!       Handbook of mathematical functions.
!
!---------------------------------------------------------------------

   use fmzm  ! Modules for derived-types

   implicit none

   ! Function
   ! --------

   type(fm) :: hg2f1fm

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

   type(fm), intent(in) :: x, a, b, c
   integer, intent(out) :: ier

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

```
!! type(fm), external :: gamma, log_gamma, psi
   type(fm), external :: fmeulr, fmpi

   type(fm) :: zero, half, one, two
   type(fm) :: epsil
   type(fm) :: eps
   type(fm) :: xlimit
                        ! xlimit: The limit for x over which Psi function
                        ! expansion is used if c-a-b is an integer number
   type(fm) :: el
   type(fm) :: pi
   type(fm) :: sqrpi
   type(fm) :: sqrpilog
   type(fm) :: twolog
   type(fm) :: explower, expupper
   type(fm) :: gamupper

   integer, parameter :: maxiter=9000   ! maximum number of iterations

   type(fm) :: aa, bb, cc, d, e, f, pow, s, t, u, v, xl, xx, y
   type(fm) :: i_fm
   integer :: limgam
   integer :: i, m
   logical :: aneg, bneg, caneg, cbneg, xneg

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

     ! Specify constants
   zero = to_fm('0.0')
   half = to_fm('0.5')
   one  = to_fm('1.0')
   two  = to_fm('2.0')
   el = fmeulr(el)  ! Euler constant
   pi = fmpi(pi)    ! Pi
   sqrpi = sqrt(pi)
   sqrpilog = log(sqrt(pi))
   twolog = log(two)
   epsil = epsilon(one)
   eps = to_fm('1.0e-14')
   xlimit = to_fm('0.95')   ! The limit for x over which Psi function
                        ! expansion is used if c-a-b is an integer number
   explower = to_fm('-706.893') ! Minimum argument for the exponential function
   expupper = -explower         ! Maximum argument for the exponential function
   gamupper = to_fm('171.624')  ! Maximum argument for the gamma function
   limgam = int(gamupper) - 2

   hg2f1fm = zero
   ier = 0
   xx = one - x

     ! Check input
   if ( c == zero .or. c == -abs(nint(c)) .or. abs(x) > one .or. &
```

```
     xx < eps .and. c-a-b <= zero ) then
   ier = 1
   return
end if

   ! Special cases
aneg = a < zero .and. a == nint(a)
bneg = b < zero .and. b == nint(b)
caneg = c-a <= zero .and. c-a == nint(c-a)
cbneg = c-b <= zero .and. c-b == nint(c-b)
if ( x == zero .or. a == zero .or. b == zero ) then
   hg2f1fm = one
   return
else if ( abs(c-a) <= eps ) then
   hg2f1fm = xx**(-b)
   return
else if ( abs(c-b) <= eps ) then
   hg2f1fm = xx**(-a)
   return
else if ( abs(xx) <= eps .and. c-a-b > zero ) then
   d = max( c, c-a, c-b, c-a-b )
   if ( d <= gamupper ) then
      hg2f1fm = gamma(c) / gamma(c-a) * gamma(c-a-b) / gamma(c-b)
   else
      d = min( c, c-a, c-b, c-a-b )
      if ( d > zero ) then
         s = log_gamma(c) + log_gamma(c-a-b) - log_gamma(c-a) - log_gamma(c-b)
         if ( s >= explower .and. s <= expupper ) then
            hg2f1fm = exp(s)
         else if ( s > expupper ) then
            hg2f1fm = huge(one)
            ier = 2
         end if
      else
         hg2f1fm = huge(one)
         ier = 2
      end if
   end if
   return
else if ( abs(one+x) <= eps .and. abs(c-a+b-one) <= eps ) then
   d = max( c, one+a*half-b, half+half*a )
   if ( d <= gamupper ) then
      hg2f1fm = gamma(c) / gamma(one+a*half-b) / gamma( half+half*a) * sqrpi / two**a
   else
      d = min( c, one+a*half-b, half+half*a )
      if ( d > zero ) then
     s = (-a)*twolog+sqrpilog + log_gamma(c) - log_gamma(one+a*half-b) - log_gamma(half+half*a)
         if ( s >= explower .and. s <= expupper ) then
            hg2f1fm = exp(s)
         else if ( s > expupper ) then
            hg2f1fm = huge(one)
            ier = 2
```

```
          end if
        else
          hg2f1fm = huge(one)
          ier = 2
        end if
      end if
      return
    else if ( aneg .or. bneg ) then
      if ( aneg ) then
        m = abs(nint(a))
!!      if ( bneg ) m = min(m,abs(nint(b)))
        if ( bneg ) m = min(abs(nint(a)),abs(nint(b)))
      else
        m = abs(nint(b))
      end if
      s = one
      t = one
      do i = 0, m-1
        i_fm = to_fm(i)
        t = t * (a+i_fm)/(c+i_fm)*(b+i_fm)/(i_fm+one)*x
        s = s + t
      end do
      hg2f1fm = s
      return
    else if ( caneg .or. cbneg ) then
      if ( caneg ) then
        m = abs(nint(c-a))
!!      if ( cbneg ) m = min(m,abs(nint(c-b)))
        if ( cbneg ) m = min(abs(nint(c-a)),abs(nint(c-b)))
      else
        m = abs(nint(c-b))
      end if
      s = one
      t = one
      do i = 0, m-1
        i_fm = to_fm(i)
        t = t * (c-a+i_fm)/(c+i_fm)*(c-b+i_fm)/(i_fm+one)*x
        s = s + t
      end do
      s = xx**(c-a-b) * s
      hg2f1fm = s
      return
    end if

    m = nint(c-a-b)

    if ( x <= xlimit .or. abs(c-a-b-m) > eps .or. abs(m) > limgam  &
       .or. max(c,max(a,b)+max(m,0)) > gamupper ) then
       ! Use power series expansion

      xx = x
      aa = a
```

```
      bb = b
      cc = c
      xneg = x < zero
      if ( xneg ) then
        xx = x/(x-one)
        if ( a <= b ) then
          pow = -a
          bb = c - b
        else
          pow = -b
          aa = c - a
        end if
      end if
      s = one
      y = one
      do i = 0, maxiter-1
        y = y * (aa+i)/(cc+i)*(bb+i)*xx/(i+1)
        t = s + y
        if ( s == t ) then
          if ( xneg ) s = (one-x)**pow * s
          hg2f1fm = s
          return
        end if
        s = t
      end do
      if ( xneg ) s = (one-x)**pow * s
      hg2f1fm = s
      ier = 3

    else
        ! Use Psi function expansion if x exceeds the limit and
        ! m=c-a-b is an integer number

      xl = log(xx)
      d = abs(m)
      if ( m >= 0 ) then
        e = d
        f = zero
        v = one
        if ( m > 0 ) then
          u = xx**d
        else
          u = one
        end if
      else
        e = zero
        f = m
        u = one
        v = xx**f
      end if
      u = gamma(c) / gamma(a+f) / gamma(b+f) * u
      if ( u == zero ) return
```

```
      s = (xl+el-psi(d+one)+psi(a+e)+psi(b+e))/gamma(d+one)                ! sum for i=0
      t = (a+e)*(b+e)/gamma(d+one+one)*xx    ! Pochhammer, factorial and power term for
i=1
      ier = 3
      do i = 1, maxiter
        i_fm = to_fm(i)
        y = t * (xl-psi(i_fm+one)-psi(d+i_fm+one)+psi(a+e+i_fm)+psi(b+e+i_fm))
        if ( abs(y) < epsil*abs(s) ) then
          ier = 0
          exit
        end if
        s = s + y
        t = t * (a+e+i_fm)*(b+e+i_fm)/(i_fm+one)*xx/(d+i_fm+one)
      end do
      if ( mod(m,2) == 0 ) then
        u = -u*s
      else
        u = u*s
      end if
      hg2f1fm = u
      if ( m == 0 ) return
      v = gamma(c) / gamma(a+e) * gamma(d) / gamma(b+e) * v
      if ( v == zero ) return
      d = one - d
      s = one
      t = one
      do i = 0, abs(m)-2
        i_fm = to_fm(i)
        t = t * (a+f+i_fm)*(b+f+i_fm)/(i_fm+one)*xx/(d+i_fm)
        s = s + t
      end do
      hg2f1fm = v*s + u

    end if

  end function hg2f1fm
```

Chapter 10
Generalized harmonic number

Function harmong(a, n)
File : harmong.f90

```
function harmong( a, n )

!-----------------------------------------------------------------------
!
!    Computes generalized harmonic number
!
!    A     - INPUT . Power parameter                        - Real
!    N     - INPUT . Argument                    (N > 0) - Integer
!
!-----------------------------------------------------------------------

   implicit none

   !    Function
   !    --------

   real(kind=8) :: harmong

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

   real(kind=8), intent(in) :: a
   integer, intent(in) :: n

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

   real(kind=8), parameter :: zero=0.0_8, one=1.0_8, alim=5.0_8
   real(kind=8), save :: a0, h0=one
   real(kind=8) :: aa, h, h1
   integer, save :: n0=1
   integer :: i

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

   if ( n < 1 ) then
      harmong = -one
      return
   endif
   if ( a /= a0 ) then
      n0 = 1
      h0 = one
   endif
   h = h0
   if ( a == zero ) then
      h = n
   else if ( a > zero ) then
      if ( a == one ) then
         if ( n >= n0 ) then
            do i = n0+1, n
               h = h + one/i
            end do
         else
```

```
            do i = n+1, n0
              h = h - one/i
            end do
          end if
        else if ( a < alim ) then
          if ( n >= n0 ) then
            do i = n0+1, n
              h = h + one/(i**a)
            end do
          else
            do i = n+1, n0
              h = h - one/(i**a)
            end do
          end if
        else
          if ( n >= n0 ) then
            h1 = h
            do i = n0+1, n
              h = h + one/(i**a)
              if ( h == h1 ) exit
              h1 = h
            end do
          else
            do i = n+1, n0
              h = h - one/(i**a)
            end do
          end if
        end if
      else
        if ( a == -one ) then
          h = n*(n+1)/2
        else
          aa = -a
          if ( n >= n0 ) then
            do i = n0+1, n
              h = h + i**aa
            end do
          else
            do i = n+1, n0
              h = h - i**aa
            end do
          end if
        end if
      end if
      n0 = n
      h0 = h
      a0 = a
      harmong = h

    end function harmong
```

Chapter 11
Modified Bessel functions of the first kind with non-negative argument

Function bessel_i(x, no, alpha, ier)
File : bessel_i.f90

```
   function bessel_i( x, no, alpha, ier )

!-----------------------------------------------------------------------
!
!     Computes the modified Bessel function of the first kind,
!     Ino+alpha (x), for non-negative argument x, and non-negative
!     order no+alpha
!
!     X     - Input . Argument (0 <= X < exparg see below)    - Real
!     NO    - Input . Integer part of order (NO >= 0)         - Integer
!     ALPHA - Input . Fractional part of order                - Real
!                   (0 <= ALPHA < 1)
!     IER   - Output. Return code:                            - Integer
!                   0 = normal
!                   1 = invalid input argument
!                   <0= Requested function value could not
!                      be calculated accurately
!
!     Fortran functions called:
!       EXP  DGAMMA  INT  MAX  MIN  REAL  SQRT
!
!     From Netlib library subroutine
!
!-----------------------------------------------------------------------
!
!  This routine calculates Bessel functions I SUB(N+ALPHA) (X)
!  for non-negative argument X, and non-negative order N+ALPHA,
!  with or without exponential scaling.
!
!
! Explanation of variables in the calling sequence
!
! X     - Working precision non-negative real argument for which
!         I's or exponentially scaled I's (I*EXP(-X))
!         are to be calculated.  If I's are to be calculated,
!         X must be less than EXPARG (see below).
! ALPHA - Working precision fractional part of order for which
!         I's or exponentially scaled I's (I*EXP(-X)) are
!         to be calculated.  0 .LE. ALPHA .LT. 1.0.
! NB    - Integer number of functions to be calculated, NB .GT. 0.
!         The first function calculated is of order ALPHA, and the
!         last is of order (NB - 1 + ALPHA).
! IZE   - Integer type.  IZE = 1 if unscaled I's are to calculated,
!         and 2 if exponentially scaled I's are to be calculated.
! B     - Working precision output vector of length NB.  If the routine
!         terminates normally (NCALC=NB), the vector B contains the
!         functions I(ALPHA,X) through I(NB-1+ALPHA,X), or the
!         corresponding exponentially scaled functions.
! NCALC - Integer output variable indicating possible errors.
!         Before using the vector B, the user should check that
!         NCALC=NB, i.e., all orders have been calculated to
!         the desired accuracy.  See error returns below.
```

```
!
!*****************************************************************
!
! Explanation of machine-dependent constants
!
!   beta   = Radix for the floating-point system
!   minexp = Smallest representable power of beta
!   maxexp = Smallest power of beta that overflows
!   it     = Number of bits in the mantissa of a working precision
!            variable
!   NSIG   = Decimal significance desired.  Should be set to
!            INT(LOG10(2)*it+1).  Setting NSIG lower will result
!            in decreased accuracy while setting NSIG higher will
!            increase CPU time without increasing accuracy.  The
!            truncation error is limited to a relative error of
!            T=.5*10**(-NSIG).
!   ENTEN  = 10.0 ** K, where K is the largest integer such that
!            ENTEN is machine-representable in working precision
!   ENSIG  = 10.0 ** NSIG
!   RTNSIG = 10.0 ** (-K) for the smallest integer K such that
!            K .GE. NSIG/4
!   ENMTEN = Smallest ABS(X) such that X/4 does not underflow
!   XLARGE = Upper limit on the magnitude of X when IZE=2.  Bear
!            in mind that if ABS(X)=N, then at least N iterations
!            of the backward recursion will be executed.  The value
!            of 10.0 ** 4 is used on every machine.
!   EXPARG = Largest working precision argument that the library
!            EXP routine can handle and upper limit on the
!            magnitude of X when IZE=1; approximately
!            LOG(beta**maxexp)
!
!     Approximate values for some important machines are:
!
!                        beta       minexp      maxexp       it
!
!  CRAY-1         (S.P.)    2        -8193        8191        48
!  Cyber 180/855
!    under NOS    (S.P.)    2         -975        1070        48
!  IEEE (IBM/XT,
!    SUN, etc.)   (S.P.)    2         -126         128        24
!  IEEE (IBM/XT,
!    SUN, etc.)   (D.P.)    2        -1022        1024        53
!  IBM 3033       (D.P.)   16          -65          63        14
!  VAX            (S.P.)    2         -128         127        24
!  VAX D-Format   (D.P.)    2         -128         127        56
!  VAX G-Format   (D.P.)    2        -1024        1023        53
!
!                       NSIG        ENTEN       ENSIG       RTNSIG
!
!  CRAY-1         (S.P.)   15       1.0E+2465   1.0E+15     1.0E-4
!  Cyber 180/855
!    under NOS    (S.P.)   15        1.0E+322   1.0E+15     1.0E-4
```

```
! IEEE (IBM/XT,
!   SUN, etc.)  (S.P.)     8       1.0E+38     1.0E+8      1.0E-2
! IEEE (IBM/XT,
!   SUN, etc.)  (D.P.)    16       1.0D+308    1.0D+16     1.0D-4
! IBM 3033      (D.P.)     5       1.0D+75     1.0D+5      1.0D-2
! VAX           (S.P.)     8       1.0E+38     1.0E+8      1.0E-2
! VAX D-Format  (D.P.)    17       1.0D+38     1.0D+17     1.0D-5
! VAX G-Format  (D.P.)    16       1.0D+307    1.0D+16     1.0D-4
!
!                     ENMTEN      XLARGE   EXPARG
!
! CRAY-1        (S.P.)   1.84E-2466   1.0E+4    5677
! Cyber 180/855
!   under NOS   (S.P.)   1.25E-293    1.0E+4     741
! IEEE (IBM/XT,
!   SUN, etc.)  (S.P.)   4.70E-38     1.0E+4      88
! IEEE (IBM/XT,
!   SUN, etc.)  (D.P.)   8.90D-308    1.0D+4     709
! IBM 3033      (D.P.)   2.16D-78     1.0D+4     174
! VAX           (S.P.)   1.17E-38     1.0E+4      88
! VAX D-Format  (D.P.)   1.17D-38     1.0D+4      88
! VAX G-Format  (D.P.)   2.22D-308    1.0D+4     709
!
!*****************************************************************
!
! Error returns
!
!  In case of an error,  NCALC .NE. NB, and not all I's are
!  calculated to the desired accuracy.
!
!  NCALC .LT. 0:  An argument is out of range. For example,
!     NB .LE. 0, IZE is not 1 or 2, or IZE=1 and ABS(X) .GE. EXPARG.
!     In this case, the B-vector is not calculated, and NCALC is
!     set to MIN0(NB,0)-1 so that NCALC .NE. NB.
!
!  NB .GT. NCALC .GT. 0: Not all requested function values could
!     be calculated accurately.  This usually occurs because NB is
!     much larger than ABS(X).  In this case, B(N) is calculated
!     to the desired accuracy for N .LE. NCALC, but precision
!     is lost for NCALC .LT. N .LE. NB.  If B(N) does not vanish
!     for N .GT. NCALC (because it is too small to be represented),
!     and B(N)/B(NCALC) = 10**(-K), then only the first NSIG-K
!     significant figures of B(N) can be trusted.
!
! Intrinsic functions required are:
!
!     DBLE, EXP, DGAMMA, GAMMA, INT, MAX, MIN, REAL, SQRT
!
! Acknowledgement
!
!  This program is based on a program written by David J.
!  Sookne (2) that computes values of the Bessel functions J or
```

```
! I of real argument and integer order.  Modifications include
! the restriction of the computation to the I Bessel function
! of non-negative real argument, the extension of the computation
! to arbitrary positive order, the inclusion of optional
! exponential scaling, and the elimination of most underflow.
! An earlier version was published in (3).
!
! References: "A Note on Backward Recurrence Algorithms," Olver,
!              F. W. J., and Sookne, D. J., Math. Comp. 26, 1972,
!              pp 941-947.
!
!             "Bessel Functions of Real Argument and Integer Order,"
!              Sookne, D. J., NBS Jour. of Res. B. 77B, 1973, pp
!              125-132.
!
!             "ALGORITHM 597, Sequence of Modified Bessel Functions
!              of the First Kind," Cody, W. J., Trans. Math. Soft.,
!              1983, pp. 242-245.
!
! Latest modification: May 30, 1989
!
! Modified by: W. J. Cody and L. Stoltz
!              Applied Mathematics Division
!              Argonne National Laboratory
!              Argonne, IL  60439
!
!-----------------------------------------------------------------------

implicit none

! Function
! --------

real(kind=8) :: bessel_i

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

real(kind=8), intent(in) :: x, alpha
integer, intent(in) :: no
integer, intent(out) :: ier

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

! Mathematical constants
real(kind=8), parameter :: zero=0.0_8, half=0.5_8, one=1.0_8, two=2.0_8, const=1.585_8
! Machine dependent parameters
real(kind=8), parameter :: exparg=709.0_8, xlarge=1.0e4_8
real(kind=8), parameter :: enmten=8.9e-308_8, enten=1.0e308_8, ensig=1.0e16_8, rtnsig=1.0e-4_8
integer, parameter :: nsig=16
```

```
real(kind=8) :: em, empal, emp2al, en, halfx, p, plast, pold, psave, psavel
real(kind=8) :: sum, tempa, tempb, tempc, test, tover
integer :: k, l, magx, n, nb, nbmx, ncalc, nend, nstart
real(kind=8) :: b(no+1)


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

bessel_i = zero
nb = no + 1
if ( nb <= 0 .or. alpha < zero .or. alpha >= one .or. x > exparg .or. x < zero ) then
ier = 1
return
end if

         !-----------------------------------------
! Use 2-term ascending series for small x

         !-----------------------------------------
ncalc = nb
magx = int(x)
if ( x >= rtnsig ) then

            !---------------------------------------------------------
! Initialize the forward sweep, the p-sequence of Olver

            !---------------------------------------------------------
nbmx = nb-magx
n = magx+1
en = n+n + (alpha+alpha)
plast = one
p = en / x

            !---------------------------------------
! Calculate general significance test

            !---------------------------------------
test = ensig + ensig
if ( 2*magx > 5*nsig ) then
test = sqrt(test*p)

             else
test = test / const**magx
end if
if ( nbmx >= 3 ) then

               !---------------------------------------------------------------
! Calculate p-sequence until n = nb-1.  Check for possible overflow

               !---------------------------------------------------------------
tover = enten / ensig
nstart = magx+2
```

```
nend = nb - 1
do k = nstart, nend
n = k
en = en + two
pold = plast
plast = p
p = en * plast/x + pold
if ( p > tover ) then

                    !------------------------------------------------------------
! To avoid overflow, divide p-sequence by tover.  Calculate
! p-sequence until abs(p) > 1

                    !------------------------------------------------------------
tover = enten
p = p / tover
plast = plast / tover
psave = p
psavel = plast
nstart = n + 1
do
n = n + 1
en = en + two
pold = plast
plast = p
p = en * plast/x + pold
if ( p > one ) exit
end do
tempb = en / x

                    !------------------------------------------------------
! Calculate backward test, and find ncalc, the highest n
! such that the test is passed.

                    !------------------------------------------------------
test = pold*plast / ensig
test = test*(half-half/(tempb*tempb))
p = plast * tover
n = n - 1
en = en - two
nend = min0(nb,n)
do l = nstart, nend
ncalc = l
pold = psavel
psavel = psave
psave = en * psavel/x + pold
if ( psave*psavel > test ) then
ncalc = ncalc - 1
goto 10
end if
end do
ncalc = nend
```

```
goto 10
end if
end do
n = nend
en = n+n + (alpha+alpha)

              !------------------------------------------------
! Calculate special significance test for nbmx > 2

              !------------------------------------------------
test = max(test,sqrt(plast*ensig)*sqrt(p+p))
end if

            !-------------------------------------------------
! Calculate p-sequence until significance test passed

            !-------------------------------------------------
do
n = n + 1
en = en + two
pold = plast
plast = p
p = en * plast/x + pold
if ( p >= test ) exit
end do
10 continue

            !-----------------------------------------------------------
! Initialize the backward recursion and the normalization sum

            !-----------------------------------------------------------
n = n + 1
en = en + two
tempb = zero
tempa = one / p
em = n - one
empal = em + alpha
emp2al = (em - one) + (alpha + alpha)
sum = tempa * empal * emp2al / em
nend = n - nb
if ( nend < 0 ) then

              !-------------------------------------------------
! n < nb, so store b(n) and set higher orders to zero

              !-------------------------------------------------
b(n) = tempa
nend = -nend
do l = 1, nend
b(n+l) = zero
end do
else
```

```
if ( nend > 0 ) then

                !-------------------------------------------------------
! Recur backward via difference equation, calculating (but
! not storing) b(n), until n = nb.

                !-------------------------------------------------------
do l = 1, nend
n = n - 1
en = en - two
tempc = tempb
tempb = tempa
tempa = (en*tempb) / x + tempc
em = em - one
emp2al = emp2al - one
if ( n == 1 ) goto 20
if ( n == 2 ) emp2al = one
empal = empal - one
sum = (sum + tempa*empal) * emp2al / em
end do
end if
20 continue
!--------------
! Store b(nb)
!--------------
b(n) = tempa
if ( nb <= 1 ) then
sum = (sum + sum) + tempa
goto 40
end if
!------------------------------
! Calculate and store b(nb-1)
!------------------------------
n = n - 1
en = en - two
b(n)  = (en*tempa) / x + tempb
if ( n == 1 ) goto 30
em = em - one
emp2al = emp2al - one
if ( n == 2 ) emp2al = one
empal = empal - one
sum = (sum + b(n)*empal) * emp2al / em
end if
nend = n - 2
if ( nend > 0 ) then

              !-----------------------------------------------------------
! Calculate via difference equation and store b(n), until n = 2

              !-----------------------------------------------------------
do l = 1, nend
n = n - 1
```

```
en = en - two
b(n) = (en*b(n+1)) / x +b(n+2)
em = em - one
emp2al = emp2al - one
if ( n == 2 ) emp2al = one
empal = empal - one
sum = (sum + b(n)*empal) * emp2al / em
end do
end if
!-----------------
! Calculate b(1)
!-----------------
b(1) = two*empal*b(2) / x + b(3)
30 continue
sum = (sum + sum) + b(1)
40 continue

          !---------------------------------------
! Normalize.  Divide all b(n) by sum

          !---------------------------------------
if ( alpha /= zero ) sum = sum * dgamma(one+alpha) * (x*half)**(-alpha)
sum = sum * exp(-x)
tempa = enmten
if ( sum > one ) tempa = tempa * sum
do n = 1, nb
if ( b(n) < tempa ) b(n) = zero
b(n) = b(n) / sum
end do
ier = ncalc - nb
if ( ier == 0 ) bessel_i = b(nb)
return
else

          !---------------------------------------
! Two-term ascending series for small x

          !---------------------------------------
tempa = one
empal = one + alpha
halfx = zero
if ( x > enmten ) halfx = half * x
if ( alpha /= zero ) tempa = halfx**alpha /dgamma(empal)
tempb = zero
if ( (x+one) > one ) tempb = halfx * halfx
b(1) = tempa + tempa*tempb / empal
if ( (x /= zero) .and. (b(1) == zero) ) ncalc = 0
if ( nb > 1 ) then
if ( x == zero ) then
do n = 2, nb
b(n) = zero
end do
```

```
else

                !-----------------------------------
! Calculate higher-order functions

                !-----------------------------------
tempc = halfx
tover = (enmten + enmten) / x
if ( tempb /= zero ) tover = enmten / tempb
do n = 2, nb
tempa = tempa / empal
empal = empal + one
tempa = tempa * tempc
if ( tempa <= tover*empal ) tempa = zero
b(n) = tempa + tempa*tempb / empal
if ( (b(n) == zero) .and. (ncalc > n) ) ncalc = n-1
end do
end if
end if
end if
ier = ncalc - nb
if ( ier == 0 ) bessel_i = b(nb)

end function bessel_i
```

Chapter 12
Modified Bessel functions of the first kind with positive argument

Function bessel_k(x, no, alpha, ier)

File : bessel_k.f90

```
   function bessel_k( x, no, alpha, ier )

   !------------------------------------------------------------------------
!
!    Computes the modified Bessel function of the second kind,
!    Kno+alpha (x), for positive argument x, and non-negative
!    order no+alpha
!
!    X     - Input . Argument (0 < X < xmax see below)       - Real
!    NO    - Input . Integer part of order (NO >= 0)         - Integer
!    ALPHA - Input . Fractional part of order                - Real
!                  (0 <= ALPHA < 1)
!    IER   - Output. Return code:                            - Integer
!                  0 = normal
!                  1 = invalid input argument
!                  <0= Requested function value could not
!                     be calculated accurately
!
!    Fortran functions called:
!      ABS  AINT  EXP  INT  LOG  MAX  MIN  SINH  SQRT
!
!    From Netlib library subroutine
!
!------------------------------------------------------------------------
!
! This FORTRAN 77 routine calculates modified Bessel functions
! of the second kind, K SUB(N+ALPHA) (X), for non-negative
! argument X, and non-negative order N+ALPHA, with or without
! exponential scaling.
!
! Explanation of variables in the calling sequence
!
! Description of output values ..
!
! X     - Working precision non-negative real argument for which
!         K's or exponentially scaled K's (K*EXP(X))
!         are to be calculated.  If K's are to be calculated,
!         X must not be greater than XMAX (see below).
! ALPHA - Working precision fractional part of order for which
!         K's or exponentially scaled K's (K*EXP(X)) are
!         to be calculated.  0 .LE. ALPHA .LT. 1.0.
! NB    - Integer number of functions to be calculated, NB .GT. 0.
!         The first function calculated is of order ALPHA, and the
!         last is of order (NB - 1 + ALPHA).
! IZE   - Integer type.  IZE = 1 if unscaled K's are to be calculated,
!         and 2 if exponentially scaled K's are to be calculated.
! BK    - Working precision output vector of length NB.  If the
!         routine terminates normally (NCALC=NB), the vector BK
!         contains the functions K(ALPHA,X), ... , K(NB-1+ALPHA,X),
!         or the corresponding exponentially scaled functions.
!         If (0 .LT. NCALC .LT. NB), BK(I) contains correct function
!         values for I .LE. NCALC, and contains the ratios
```

```
!         K(ALPHA+I-1,X)/K(ALPHA+I-2,X) for the rest of the array.
! NCALC - Integer output variable indicating possible errors.
!         Before using the vector BK, the user should check that
!         NCALC=NB, i.e., all orders have been calculated to
!         the desired accuracy.  See error returns below.
!
!*******************************************************************
!
! Explanation of machine-dependent constants
!
!   beta   = Radix for the floating-point system
!   minexp = Smallest representable power of beta
!   maxexp = Smallest power of beta that overflows
!   EPS    = The smallest positive floating-point number such that
!            1.0+EPS .GT. 1.0
!   XMAX   = Upper limit on the magnitude of X when IZE=1;  Solution
!            to equation:
!               W(X) * (1-1/8X+9/128X**2) = beta**minexp
!            where  W(X) = EXP(-X)*SQRT(PI/2X)
!   SQXMIN = Square root of beta**minexp
!   XINF   = Largest positive machine number; approximately
!            beta**maxexp
!   XMIN   = Smallest positive machine number; approximately
!            beta**minexp
!
!     Approximate values for some important machines are:
!
!                          beta       minexp      maxexp      EPS
!
!  CRAY-1         (S.P.)      2        -8193        8191    7.11E-15
!  Cyber 180/185
!    under NOS    (S.P.)      2         -975        1070    3.55E-15
!  IEEE (IBM/XT,
!    SUN, etc.)   (S.P.)      2         -126         128    1.19E-7
!  IEEE (IBM/XT,
!    SUN, etc.)   (D.P.)      2        -1022        1024    2.22D-16
!  IBM 3033       (D.P.)     16          -65          63    2.22D-16
!  VAX            (S.P.)      2         -128         127    5.96E-8
!  VAX D-Format   (D.P.)      2         -128         127    1.39D-17
!  VAX G-Format   (D.P.)      2        -1024        1023    1.11D-16
!
!                        SQXMIN       XINF        XMIN       XMAX
!
! CRAY-1         (S.P.)  6.77E-1234  5.45E+2465  4.59E-2467 5674.858
! Cyber 180/855
!   under NOS    (S.P.)  1.77E-147   1.26E+322   3.14E-294   672.788
! IEEE (IBM/XT,
!   SUN, etc.)   (S.P.)  1.08E-19    3.40E+38    1.18E-38     85.337
! IEEE (IBM/XT,
!   SUN, etc.)   (D.P.)  1.49D-154   1.79D+308   2.23D-308   705.342
! IBM 3033       (D.P.)  7.35D-40    7.23D+75    5.40D-79    177.852
! VAX            (S.P.)  5.42E-20    1.70E+38    2.94E-39     86.715
```

```
! VAX D-Format  (D.P.)  5.42D-20    1.70D+38    2.94D-39     86.715
! VAX G-Format  (D.P.)  7.46D-155   8.98D+307   5.57D-309    706.728
!
!*****************************************************************
!
! Error returns
!
!  In case of an error, NCALC .NE. NB, and not all K's are
!  calculated to the desired accuracy.
!
!  NCALC .LT. -1:  An argument is out of range. For example,
!       NB .LE. 0, IZE is not 1 or 2, or IZE=1 and ABS(X) .GE.
!       XMAX.  In this case, the B-vector is not calculated,
!       and NCALC is set to MIN0(NB,0)-2  so that NCALC .NE. NB.
!  NCALC = -1:  Either  K(ALPHA,X) .GE. XINF  or
!       K(ALPHA+NB-1,X)/K(ALPHA+NB-2,X) .GE. XINF.  In this case,
!       the B-vector is not calculated.  Note that again
!       NCALC .NE. NB.
!
!  0 .LT. NCALC .LT. NB: Not all requested function values could
!       be calculated accurately.  BK(I) contains correct function
!       values for I .LE. NCALC, and contains the ratios
!       K(ALPHA+I-1,X)/K(ALPHA+I-2,X) for the rest of the array.
!
! Intrinsic functions required are:
!
!     ABS, AINT, EXP, INT, LOG, MAX, MIN, SINH, SQRT
!
! Acknowledgement
!
!  This program is based on a program written by J. B. Campbell
!  (2) that computes values of the Bessel functions K of real
!  argument and real order.  Modifications include the addition
!  of non-scaled functions, parameterization of machine
!  dependencies, and the use of more accurate approximations
!  for SINH and SIN.
!
! References: "On Temme's Algorithm for the Modified Bessel
!              Functions of the Third Kind," Campbell, J. B.,
!              TOMS 6(4), Dec. 1980, pp. 581-586.
!
!             "A FORTRAN IV Subroutine for the Modified Bessel
!              Functions of the Third Kind of Real Order and Real
!              Argument," Campbell, J. B., Report NRC/ERB-925,
!              National Research Council, Canada.
!
!  Latest modification: May 30, 1989
!
!  Modified by: W. J. Cody and L. Stoltz
!               Applied Mathematics Division
!               Argonne National Laboratory
!               Argonne, IL  60439
```

```
!
!-----------------------------------------------------------------------

implicit none

! Function
! --------

real(kind=8) :: bessel_k

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

real(kind=8), intent(in) :: x, alpha
integer, intent(in) :: no
integer, intent(out) :: ier

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

! Mathematical constants
!   a = log(2.d0) - Euler's constant
!   d = sqrt(2.d0/pi)
real(kind=8), parameter :: zero=0.0_8, half=0.5_8, one=1.0_8, two=2.0_8, four=4.0_8
real(kind=8), parameter :: a=0.11593151565841244881_8, d=0.797884560802865364_8
real(kind=8), parameter :: tinyx=1.0e-10_8
! Machine dependent parameters
real(kind=8), parameter :: eps=2.22e-16_8, sqxmin=1.49e-154_8, xinf=1.79e+308_8
real(kind=8), parameter :: xmin=2.23e-308_8, xmax=705.342_8

real(kind=8) :: blpha, bk1, bk2, c, dm, d1, d2, d3, enu, ex
real(kind=8) :: f0, f1, f2, p0, q0, ratio, twonu, twox, t1, t2, wminf, x2by4
integer :: i, iend, itemp, j, k, m, mplus1, nb, ncalc
real(kind=8) :: bk(no+1)

! p, q - approximation for log(gamma(1+alpha))/alpha
!                                    + Euler's constant
!        Coefficients converted from hex to decimal and modified
!        by W. J. Cody, 2/26/82
! r, s - approximation for (1-alpha*pi/sin(alpha*pi))/(2*alpha)
! t    - approximation for sinh(y)/y
real(kind=8) :: p(8)=(/  &
0.805629875690432845_8   ,  0.204045500205365151e02_8, &
0.157705605106676174e03_8,  0.536671116469207504e03_8, &
0.900382759291288778e03_8,  0.730923886650660393e03_8, &
0.229299301509425145e03_8,  0.822467033424113231_8/)
real(kind=8) :: q(7)=(/  &
0.294601986247850434e02_8,  0.277577868510221208e03_8, &
0.120670325591027438e04_8,  0.276291444159791519e04_8, &
0.344374050506564618e04_8,  0.221063190113378647e04_8, &
0.572267338359892221e03_8/)
real(kind=8) :: r(5)=(/  &
```

```
-0.48672575865218401848_8    ,0.13079485869097804016e+2_8, &
-0.10196490580880537526e+3_8,0.34765409106507813131e+3_8, &
0.34958981245219347820e-3_8/)
real(kind=8) :: s(4)=(/  &
-0.25579105509976461286e+2_8,0.21257260432226544008e+3_8, &
-0.61069018684944109624e+3_8,0.42269668805777760407e+3_8/)
real(kind=8) :: t(6)=(/  &
0.16125990452916363814e-9_8,0.25051878502858255354e-7_8, &
0.27557319615147964774e-5_8,0.19841269840928373686e-3_8, &
0.83333333333334751799e-2_8,0.16666666666666666446_8/)
real(kind=8) :: estf(7)=(/  &
4.18341e1_8, 7.1075_8, 6.4306_8, 4.25110e1_8, 1.35633_8, 8.45096e1_8, 2.0e1_8/)
real(kind=8) :: estm(6)=(/  &
5.20583e1_8, 5.7607_8, 2.7782_8, 1.44303e1_8, 1.853004e2_8, 9.3715_8/)


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

bessel_k = zero
nb = no + 1
ncalc = min(nb,0)-2
if ( nb <= 0 .or. alpha
     < zero .or. alpha >= one .or. x > xmax .or. x <= zero ) then
ier = 1
return
end if
ier = 0
ex = x
enu = alpha
k = 0
if (enu < sqxmin) enu = zero
if ( enu > half ) then
k = 1
enu = enu - one
end if
twonu = enu+enu
iend = nb+k-1
c = enu*enu
d3 = -c
if ( ex <= one ) then

            !-----------------------------------------------
!  Calculation of p0 = gamma(1+alpha) * (2/x)**alpha
!                  q0 = gamma(1-alpha) * (x/2)**alpha

            !-----------------------------------------------
d1 = zero
d2 = p(1)
t1 = one
t2 = q(1)
do i = 2, 7, 2
d1 = c*d1+p(i)
```

```
d2 = c*d2+p(i+1)
t1 = c*t1+q(i)
t2 = c*t2+q(i+1)
end do
d1 = enu*d1
t1 = enu*t1
f1 = log(ex)
f0 = a+enu*(p(8)-enu*(d1+d2)/(t1+t2))-f1
q0 = exp(-enu*(a-enu*(p(8)+enu*(d1-d2)/(t1-t2))-f1))
f1 = enu*f0
p0 = exp(f1)
!----------------------
!  Calculation of f0 =
!----------------------
d1 = r(5)
t1 = one
do i = 1, 4
d1 = c*d1+r(i)
t1 = c*t1+s(i)
end do
if ( abs(f1) <= half ) then
f1 = f1*f1
d2 = zero
do i = 1, 6
d2 = f1*d2+t(i)
end do
d2 = f0+f0*f1*d2
else
d2 = sinh(f1)/enu
end if
f0 = d2-enu*d1/(t1*p0)
if ( ex <= tinyx ) then

            !------------------------------------------------------------
!  x <= 1.0e-10
!  Calculation of K(alpha,x) and x*K(alpha+1,x)/K(alpha,x)

            !------------------------------------------------------------
bk(1) = f0+ex*f0
bk(1) = bk(1)-ex*bk(1)
ratio = p0/f0
c = ex*xinf
if ( k /= 0 ) then

              !------------------------------------------------------------
!  Calculation of K(alpha,x) and x*K(alpha+1,x)/K(alpha,x),
!  alpha >= 1/2

              !------------------------------------------------------------
ncalc = -1
if ( bk(1) >= c/ratio ) goto 50
bk(1) = ratio*bk(1)/ex
```

```
twonu = twonu+two
ratio = twonu
end if
ncalc = 1
if ( nb == 1) goto 50

              !-------------------------------------------------------------------
!  Calculate  K(alpha+l,x)/K(alpha+l-1,x),  l  =  1, 2, ... , nb-1

              !-------------------------------------------------------------------
ncalc = -1
do i = 2, nb
if (ratio >= c ) goto 50
bk(i) = ratio/ex
twonu = twonu+two
ratio = twonu
end do
ncalc = 1
goto 20
else
!----------------------
!  1.0e-10 < x <= 1.0
!----------------------
c = one
x2by4 = ex*ex/four
p0 = half*p0
q0 = half*q0
d1 = -one
d2 = zero
bk1 = zero
bk2 = zero
f1 = f0
f2 = p0
do
d1 = d1+two
d2 = d2+one
d3 = d1+d3
c = x2by4*c/d2
f0 = (d2*f0+p0+q0)/d3
p0 = p0/(d2-enu)
q0 = q0/(d2+enu)
t1 = c*f0
t2 = c*(p0-d2*f0)
bk1 = bk1+t1
bk2 = bk2+t2
if ( (abs(t1/(f1+bk1)) <= eps) .and. (abs(t2/(f2+bk2)) <= eps) ) exit
end do
bk1 = f1+bk1
bk2 = two*(f2+bk2)/ex
wminf = estf(1)*ex+estf(2)
end if
else if ( eps*ex > one ) then
```

```
!---------------
!  x > one/eps
!---------------
ncalc = nb
bk1 = one / (d*sqrt(ex))
do i = 1, nb
bk(i) = bk1
end do
goto 50
else
!-----------
!  x > 1.0
!-----------
twox = ex+ex
blpha = zero
ratio = zero
if ( ex <= four ) then

              !------------------------------------------------------------
!  Calculation of K(alpha+1,x)/K(alpha,x),  1.0 <= x <= 4.0

              !------------------------------------------------------------
d2 = aint(estm(1)/ex+estm(2))
m = int(d2)
d1 = d2+d2
d2 = d2-half
d2 = d2*d2
do i = 2, m
d1 = d1-two
d2 = d2-d1
ratio = (d3+d2)/(twox+d1-ratio)
end do

              !------------------------------------------------------------
!  Calculation of I(|alpha|,x) and I(|alpha|+1,x) by backward
!    recurrence and K(alpha,x) from the wronskian

              !------------------------------------------------------------
d2 = aint(estm(3)*ex+estm(4))
m = int(d2)
c = abs(enu)
d3 = c+c
d1 = d3-one
f1 = xmin
f0 = (two*(c+d2)/ex+half*ex/(c+d2+one))*xmin
do i = 3, m
d2 = d2-one
f2 = (d3+d2+d2)*f0
blpha = (one+d1/d2)*(f2+blpha)
f2 = f2/ex+f1
f1 = f0
f0 = f2
```

```
end do
f1 = (d3+two)*f0/ex+f1
d1 = zero
t1 = one
do i = 1, 7
d1 = c*d1+p(i)
t1 = c*t1+q(i)
end do
p0 = exp(c*(a+c*(p(8)-c*d1/t1)-log(ex)))/ex
f2 = (c+half-ratio)*f1/ex
bk1 = p0+(d3*f0-f2+f0+blpha)/(f2+f1+f0)*p0
bk1 = bk1*exp(-ex)
wminf = estf(3)*ex+estf(4)
else

            !------------------------------------------------------------------
!  Calculation of K(alpha,x) and K(alpha+1,x)/K(alpha,x), by backward
!  recurrence, for  x > 4.0

            !------------------------------------------------------------------
dm = aint(estm(5)/ex+estm(6))
m = int(dm)
d2 = dm-half
d2 = d2*d2
d1 = dm+dm
do i = 2, m
dm = dm-one
d1 = d1-two
d2 = d2-d1
ratio = (d3+d2)/(twox+d1-ratio)
blpha = (ratio+ratio*blpha)/dm
end do
bk1 = one/((d+d*blpha)*sqrt(ex))
bk1 = bk1*exp(-ex)
wminf = estf(5)*(ex-abs(ex-estf(7)))+estf(6)
end if

          !-------------------------------------------------
!  Calculation of K(alpha+1,x) from K(alpha,x) and
!    K(alpha+1,x)/K(alpha,x)

          !-------------------------------------------------
bk2 = bk1+bk1*(enu+half-ratio)/ex
end if

        !-------------------------------------------------------------------
!  Calculation of 'ncalc', K(alpha+i,x), i  =  0, 1, ... , ncalc-1,
!  K(alpha+i,x)/K(alpha+i-1,x), i  =  ncalc, ncalc+1, ... , nb-1

        !-------------------------------------------------------------------
ncalc = nb
bk(1) = bk1
```

```
if ( iend == 0 ) goto 50
j = 2-k
if ( j > 0 ) bk(j) = bk2
if ( iend == 1 ) goto 50
m = min(int(wminf-enu),iend)
do i = 2, m
t1 = bk1
bk1 = bk2
twonu = twonu+two
if ( ex < one ) then
if ( bk1 >= (xinf/twonu)*ex ) exit
else
if ( bk1/ex >= xinf/twonu ) exit
end if
bk2 = twonu/ex*bk1+t1
itemp = i
j = j+1
if ( j > 0 ) bk(j) = bk2
end do
m = itemp
if ( m == iend ) goto 50
ratio = bk2/bk1
mplus1 = m+1
ncalc = -1
do i = mplus1, iend
twonu = twonu+two
ratio = twonu/ex+one/ratio
j = j+1
if ( j > 1 ) then
bk(j) = ratio
else
if ( bk2 >= xinf/ratio ) goto 50
bk2 = ratio*bk2
end if
end do
ncalc = max(mplus1-k,1)
if ( ncalc == 1 ) bk(1) = bk2
if ( nb == 1 ) goto 50
20 continue
j = ncalc+1
do i = j, nb
if ( bk(ncalc) >= xinf/bk(i) ) goto 50
bk(i) = bk(ncalc)*bk(i)
ncalc = i
end do

50 continue
ier = ncalc - nb
if ( ier == 0 ) bessel_k = bk(nb)

end function bessel_k
```

Chapter 13
Modified Bessel functions of the first kind and order zero

Function CALCI0(ARG,RESULT,JINT)
File : NETLIBI0.f90

```
      SUBROUTINE CALCI0(ARG,RESULT,JINT)
      !------------------------------------------------------------------
      !
      ! This packet computes modified Bessel functions of the first kind
      !   and order zero, I0(X) and EXP(-ABS(X))*I0(X), for real
      !   arguments X.  It contains two function type subprograms, BESI0
      !   and BESEI0, and one subroutine type subprogram, CALCI0.
      !   The calling statements for the primary entries are
      !
      !                  Y=BESI0(X)
      !   and
      !                  Y=BESEI0(X)
      !
      !   where the entry points correspond to the functions I0(X) and
      !   EXP(-ABS(X))*I0(X), respectively.  The routine CALCI0 is
      !   intended for internal packet use only, all computations within
      !   the packet being concentrated in this routine.  The function
      !   subprograms invoke CALCI0 with the statement
      !          CALL CALCI0(ARG,RESULT,JINT)
      !   where the parameter usage is as follows
      !
      !     Function                    Parameters for CALCI0
      !      Call              ARG                 RESULT          JINT
      !
      !    BESI0(ARG)    ABS(ARG) .LE. XMAX        I0(ARG)           1
      !    BESEI0(ARG)    any real ARG        EXP(-ABS(ARG))*I0(ARG) 2
      !
      !   The main computation evaluates slightly modified forms of
      !   minimax approximations generated by Blair and Edwards, Chalk
      !   River (Atomic Energy of Canada Limited) Report AECL-4928,
      !   October, 1974.  This transportable program is patterned after
      !   the machine-dependent FUNPACK packet NATSI0, but cannot match
      !   that version for efficiency or accuracy.  This version uses
      !   rational functions that theoretically approximate I-SUB-0(X)
      !   to at least 18 significant decimal digits.  The accuracy
      !   achieved depends on the arithmetic system, the compiler, the
      !   intrinsic functions, and proper selection of the machine-
      !   dependent constants.
      !
      !*******************************************************************
      !*******************************************************************
      !
      ! Explanation of machine-dependent constants
      !
      !   beta   = Radix for the floating-point system
      !   maxexp = Smallest power of beta that overflows
      !   XSMALL = Positive argument such that 1.0 - X = 1.0 to
      !            machine precision for all ABS(X) .LE. XSMALL.
      !   XINF =   Largest positive machine number; approximately
      !            beta**maxexp
      !   XMAX =   Largest argument acceptable to BESI0;  Solution to
      !            equation:
```

```
!             W(X) * (1+1/(8*X)+9/(128*X**2) = beta**maxexp
!          where  W(X) = EXP(X)/SQRT(2*PI*X)
!
!
!    Approximate values for some important machines are:
!
!                       beta       maxexp       XSMALL
!
! CRAY-1        (S.P.)       2         8191       3.55E-15
! Cyber 180/855
!   under NOS   (S.P.)       2         1070       3.55E-15
! IEEE (IBM/XT,
!   SUN, etc.)  (S.P.)       2          128       2.98E-8
! IEEE (IBM/XT,
!   SUN, etc.)  (D.P.)       2         1024       5.55D-17
! IBM 3033      (D.P.)      16           63       6.95D-18
! VAX           (S.P.)       2          127       2.98E-8
! VAX D-Format  (D.P.)       2          127       6.95D-18
! VAX G-Format  (D.P.)       2         1023       5.55D-17
!
!
!                          XINF          XMAX
!
! CRAY-1        (S.P.)       5.45E+2465     5682.810
! Cyber 180/855
!   under NOS   (S.P.)       1.26E+322       745.893
! IEEE (IBM/XT,
!   SUN, etc.)  (S.P.)       3.40E+38         91.900
! IEEE (IBM/XT,
!   SUN, etc.)  (D.P.)       1.79D+308       713.986
! IBM 3033      (D.P.)       7.23D+75        178.182
! VAX           (S.P.)       1.70D+38         91.203
! VAX D-Format  (D.P.)       1.70D+38         91.203
! VAX G-Format  (D.P.)       8.98D+307       713.293
!
!*****************************************************************
!*****************************************************************
!
! Error returns
!
!  The program returns XINF for BESI0 for ABS(ARG) .GT. XMAX.
!
!
!  Intrinsic functions required are:
!
!     ABS, SQRT, EXP
!
!
!  Authors: W. J. Cody and L. Stoltz
!           Mathematics and Computer Science Division
!           Argonne National Laboratory
!           Argonne, IL 60439
```

```
!
!  Latest modification: June 7, 1988
!
!---------------------------------------------------------------------
      INTEGER I,JINT
!S    REAL
!D    DOUBLE PRECISION
     1       A,ARG,B,EXP40,FORTY,ONE,ONE5,P,PP,Q,QQ,RESULT,
     2       REC15,SUMP,SUMQ,TWO25,X,XINF,XMAX,XSMALL,XX
      DIMENSION P(15),PP(8),Q(5),QQ(7)
!---------------------------------------------------------------------
!  Mathematical constants
!---------------------------------------------------------------------
!S    DATA ONE/1.0E0/,ONE5/15.0E0/,EXP40/2.353852668370199854E17/,
!S   1     FORTY/40.0E0/,REC15/6.666666666666666666E-2/,
!S   2     TWO25/225.0E0/
!D    DATA ONE/1.0D0/,ONE5/15.0D0/,EXP40/2.353852668370199854D17/,
!D   1     FORTY/40.0D0/,REC15/6.666666666666666666D-2/,
!D   2     TWO25/225.0D0/
!---------------------------------------------------------------------
!  Machine-dependent constants
!---------------------------------------------------------------------
!S    DATA XSMALL/2.98E-8/,XINF/3.40E38/,XMAX/91.9E0/
!D    DATA XSMALL/5.55D-17/,XINF/1.79D308/,XMAX/713.986D0/
!---------------------------------------------------------------------
!  Coefficients for XSMALL .LE. ABS(ARG) .LT. 15.0
!---------------------------------------------------------------------
!S    DATA  P/-5.248786627945699800E-18,-1.598222667565318464 6E-14,
!S   1        -2.6843448573468483278E-11,-3.0517226450451067446E-08,
!S   2        -2.5172644670688975051E-05,-1.5453977791786851041E-02,
!S   3        -7.0935347449210549190E+00,-2.4125195876041896775E+03,
!S   4        -5.9545626019847898221E+05,-1.0313066708737980747E+08,
!S   5        -1.1912746104985237192E+10,-8.4925101247114157499E+11,
!S   6        -3.2940087627407749166E+13,-5.5050369673018427753E+14,
!S   7        -2.2335582639474375249E+15/
!S    DATA  Q/-3.7277560179962773046E+03, 6.5158506418655165707E+06,
!S   1        -6.5626560740833869295E+09, 3.7604188704092954661E+12,
!S   2        -9.7087946179594019126E+14/
!D    DATA  P/-5.248786627945699800D-18,-1.598222667565318464 6D-14,
!D   1        -2.6843448573468483278D-11,-3.0517226450451067446D-08,
!D   2        -2.5172644670688975051D-05,-1.5453977791786851041D-02,
!D   3        -7.0935347449210549190D+00,-2.4125195876041896775D+03,
!D   4        -5.9545626019847898221D+05,-1.0313066708737980747D+08,
!D   5        -1.1912746104985237192D+10,-8.4925101247114157499D+11,
!D   6        -3.2940087627407749166D+13,-5.5050369673018427753D+14,
!D   7        -2.2335582639474375249D+15/
!D    DATA  Q/-3.7277560179962773046D+03, 6.5158506418655165707D+06,
!D   1        -6.5626560740833869295D+09, 3.7604188704092954661D+12,
!D   2        -9.7087946179594019126D+14/
!---------------------------------------------------------------------
!  Coefficients for 15.0 .LE. ABS(ARG)
!---------------------------------------------------------------------
```

```
!S    DATA PP/-3.984375000000000000E-01, 2.920538459633679 3945E+00,
!S   1        -2.4708469169133954315E+00, 4.7914889422856814203E-01,
!S   2        -3.7384991926068969150E-03,-2.6801520353328635310E-03,
!S   3         9.9168777670983678974E-05,-2.1877128189032726730E-06/
!S    DATA QQ/-3.1446690275135491500E+01, 8.553956325801292 9600E+01,
!S   1        -6.0228002066743340583E+01, 1.3982595353892851542E+01,
!S   2        -1.1151759188741312645E+00, 3.2547697594819615062E-02,
!S   3        -5.5194330231005480228E-04/
!D    DATA PP/-3.98437500000000000000D-01, 2.92053845963367 93945D+00,
!D   1        -2.4708469169133954315D+00, 4.7914889422856814203D-01,
!D   2        -3.7384991926068969150D-03,-2.6801520353328635310D-03,
!D   3         9.9168777670983678974D-05,-2.1877128189032726730D-06/
!D    DATA QQ/-3.1446690275135491500D+01, 8.553956325801292 9600D+01,
!D   1        -6.0228002066743340583D+01, 1.3982595353892851542D+01,
!D   2        -1.1151759188741312645D+00, 3.2547697594819615062D-02,
!D   3        -5.5194330231005480228D-04/
!---------------------------------------------------------------------
      X = ABS(ARG)
      IF (X .LT. XSMALL) THEN
           RESULT = ONE
        ELSE IF (X .LT. ONE5) THEN
!---------------------------------------------------------------------
!  XSMALL .LE.  ABS(ARG)  .LT. 15.0
!---------------------------------------------------------------------
           XX = X * X
           SUMP = P(1)
           DO 50 I = 2, 15
             SUMP = SUMP * XX + P(I)
   50      CONTINUE
           XX = XX - TWO25
           SUMQ = ((((XX+Q(1))*XX+Q(2))*XX+Q(3))*XX+Q(4))*XX+Q(5)
           RESULT = SUMP / SUMQ
           IF (JINT .EQ. 2) RESULT = RESULT * EXP(-X)
         ELSE IF (X .GE. ONE5) THEN
           IF ((JINT .EQ. 1) .AND. (X .GT. XMAX)) THEN
                RESULT = XINF
             ELSE
!---------------------------------------------------------------------
!  15.0  .LE.  ABS(ARG)
!---------------------------------------------------------------------
                XX = ONE / X - REC15
                SUMP = ((((((PP(1)*XX+PP(2))*XX+PP(3))*XX+PP(4))*XX+
     1               PP(5))*XX+PP(6))*XX+PP(7))*XX+PP(8)
                SUMQ = ((((((XX+QQ(1))*XX+QQ(2))*XX+QQ(3))*XX+
     1               QQ(4))*XX+QQ(5))*XX+QQ(6))*XX+QQ(7)
                RESULT = SUMP / SUMQ
                IF (JINT .EQ. 2) THEN
                     RESULT = (RESULT - PP(1)) / SQRT(X)
                  ELSE
!---------------------------------------------------------------------
!  Calculation reformulated to avoid premature overflow
!---------------------------------------------------------------------
```

```
                  IF (X .LE.(XMAX-ONE5)) THEN
                     A = EXP(X)
                     B = ONE
                   ELSE
                     A = EXP(X-FORTY)
                     B = EXP40
                  END IF
                  RESULT = ((RESULT*A-PP(1)*A)/SQRT(X))*B
               END IF
            END IF
         END IF
!---------------------------------------------------------------------
!  Return for ABS(ARG) .LT. XSMALL
!---------------------------------------------------------------------
      RETURN
!---------- Last line of CALCI0 ----------
      END
!S    REAL
!D    DOUBLE PRECISION
     1    FUNCTION BESI0(X)
!---------------------------------------------------------------------
!
! This long precision subprogram computes approximate values for
!   modified Bessel functions of the first kind of order zero for
!   arguments ABS(ARG) .LE. XMAX  (see comments heading CALCI0).
!
!---------------------------------------------------------------------
      INTEGER JINT
!S    REAL
!D    DOUBLE PRECISION
     1    X, RESULT
!---------------------------------------------------------------------
      JINT=1
      CALL CALCI0(X,RESULT,JINT)
      BESI0=RESULT
      RETURN
!---------- Last line of BESI0 ----------
      END
!S    REAL
!D    DOUBLE PRECISION
     1    FUNCTION BESEI0(X)
!---------------------------------------------------------------------
!
! This function program computes approximate values for the
!   modified Bessel function of the first kind of order zero
!   multiplied by EXP(-ABS(X)), where EXP is the
!   exponential function, ABS is the absolute value, and X
!   is any argument.
!
!---------------------------------------------------------------------
      INTEGER JINT
!S    REAL
```

```
!D    DOUBLE PRECISION
     1    X, RESULT
!-----------------------------------------------------------------------
      JINT=2
      CALL CALCI0(X,RESULT,JINT)
      BESEI0=RESULT
      RETURN
!---------- Last line of BESEI0 ----------
      END
```

```
DOUBLE PRECISION FUNCTION netlibk0(ARG)
!-----------------------------------------------------------------
!
! This function program computes approximate values for the
!   modified Bessel function of the second kind of order zero
!   for arguments 0.0 .LT. ARG .LE. XMAX (see comments heading
!   CALCK0).
!
!  Authors: W. J. Cody and Laura Stoltz
!
!  Latest Modification: January 19, 1988
!
!-----------------------------------------------------------------
!-----------------------------------------------------------------
!
! This packet computes modified Bessel functions of the second kind
!   and order zero, K0(X) and EXP(X)*K0(X), for real
!   arguments X.  It contains two function type subprograms, BESK0
!   and BESEK0, and one subroutine type subprogram, CALCK0.
!   the calling statements for the primary entries are
!
!                  Y=BESK0(X)
!   and
!                  Y=BESEK0(X)
!
!   where the entry points correspond to the functions K0(X) and
!   EXP(X)*K0(X), respectively.  The routine CALCK0 is
!   intended for internal packet use only, all computations within
!   the packet being concentrated in this routine.  The function
!   subprograms invoke CALCK0 with the statement
!          CALL CALCK0(ARG,RESULT,JINT)
!   where the parameter usage is as follows
!
!      Function                    Parameters for CALCK0
!       Call              ARG                 RESULT          JINT
!
!     BESK0(ARG)    0 .LT. ARG .LE. XMAX       K0(ARG)           1
!     BESEK0(ARG)      0 .LT. ARG           EXP(ARG)*K0(ARG)     2
!
!   The main computation evaluates slightly modified forms of near
!   minimax rational approximations generated by Russon and Blair,
!   Chalk River (Atomic Energy of Canada Limited) Report AECL-3461,
!   1969.  This transportable program is patterned after the
!   machine-dependent FUNPACK packet NATSK0, but cannot match that
!   version for efficiency or accuracy.  This version uses rational
!   functions that theoretically approximate K-SUB-0(X) to at
!   least 18 significant decimal digits.  The accuracy achieved
!   depends on the arithmetic system, the compiler, the intrinsic
!   functions, and proper selection of the machine-dependent
!   constants.
!
!*******************************************************************
```

```
!*****************************************************************
!
! Explanation of machine-dependent constants
!
!   beta   = Radix for the floating-point system
!   minexp = Smallest representable power of beta
!   maxexp = Smallest power of beta that overflows
!   XSMALL = Argument below which BESK0 and BESEK0 may
!            each be represented by a constant and a log.
!            largest X such that  1.0 + X = 1.0  to machine
!            precision.
!   XINF   = Largest positive machine number; approximately
!            beta**maxexp
!   XMAX   = Largest argument acceptable to BESK0;  Solution to
!            equation:
!               W(X) * (1-1/8X+9/128X**2) = beta**minexp
!            where  W(X) = EXP(-X)*SQRT(PI/2X)
!
!
!     Approximate values for some important machines are:
!
!
!                          beta       minexp       maxexp
!
!  CRAY-1         (S.P.)       2        -8193         8191
!  Cyber 180/185
!    under NOS    (S.P.)       2         -975         1070
!  IEEE (IBM/XT,
!    SUN, etc.)   (S.P.)       2         -126          128
!  IEEE (IBM/XT,
!    SUN, etc.)   (D.P.)       2        -1022         1024
!  IBM 3033       (D.P.)      16          -65           63
!  VAX D-Format   (D.P.)       2         -128          127
!  VAX G-Format   (D.P.)       2        -1024         1023
!
!
!                        XSMALL       XINF         XMAX
!
! CRAY-1         (S.P.)    3.55E-15   5.45E+2465    5674.858
! Cyber 180/855
!   under NOS    (S.P.)    1.77E-15   1.26E+322      672.788
! IEEE (IBM/XT,
!   SUN, etc.)   (S.P.)    5.95E-8    3.40E+38        85.337
! IEEE (IBM/XT,
!   SUN, etc.)   (D.P.)    1.11D-16   1.79D+308      705.342
! IBM 3033       (D.P.)    1.11D-16   7.23D+75       177.852
! VAX D-Format   (D.P.)    6.95D-18   1.70D+38        86.715
! VAX G-Format   (D.P.)    5.55D-17   8.98D+307      706.728
!
!*****************************************************************
!*****************************************************************
!
```

```
! Error returns
!
!  The program returns the value XINF for ARG .LE. 0.0, and the
!  BESK0 entry returns the value 0.0 for ARG .GT. XMAX.
!
!
!  Intrinsic functions required are:
!
!     EXP, LOG, SQRT
!
!  Latest modification: March 19, 1990
!
!  Authors: W. J. Cody and Laura Stoltz
!           Mathematics and Computer Science Division
!           Argonne National Laboratory
!           Argonne, IL 60439
!
!----------------------------------------------------------------------
      INTEGER I
      DOUBLE PRECISION   &
         ARG,F,G,ONE,P,PP,Q,QQ,RESULT,SUMF,SUMG,SUMP,SUMQ,TEMP,  &
         X,XINF,XMAX,XSMALL,XX,ZERO
      DIMENSION P(6),Q(2),PP(10),QQ(10),F(4),G(3)
!----------------------------------------------------------------------
!  Mathematical constants
!----------------------------------------------------------------------
      DATA ONE/1.0D0/,ZERO/0.0D0/
!----------------------------------------------------------------------
!  Machine-dependent constants
!----------------------------------------------------------------------
      DATA XSMALL/1.11D-16/,XINF/1.79D+308/,XMAX/705.342D0/
!----------------------------------------------------------------------
!
!     Coefficients for XSMALL .LE.  ARG  .LE. 1.0
!
!----------------------------------------------------------------------
      DATA    P/ 5.8599221412826100000D-04, 1.1660525649895718 50D-01, &
              1.1999463724910714109D+01, 4.6850901201934832188D+02, &
              5.9169059852270512312D+03, 2.4708152720399552679D+03/
      DATA    Q/-2.4994418972832303646D+02, 2.1312714303849120380D+04/
      DATA    F/-1.6414452837299064100D+00,-2.9601657892958843866D+02, &
             -1.7733784684952985886D+04,-4.0320340761145482298D+05/
      DATA    G/-2.5064972445877992730D+02, 2.9865713163054025489D+04, &
             -1.6128136304458193998D+06/
!----------------------------------------------------------------------
!
!     Coefficients for  1.0 .LT. ARG
!
!----------------------------------------------------------------------
      DATA  PP/ 1.1394980557384778174D+02, 3.6832589957340267940D+03, &
              3.1075408980684392399D+04, 1.0577068948034021957D+05, &
              1.7398867902565686251D+05, 1.5097646353289914539D+05, &
```

```
     &       7.155706278376403754 1D+04, 1.832152587018353772 5D+04, &
     &       2.344473876419931502 1D+03, 1.160024942507603555 8D+02/
      DATA  QQ/ 2.001344306494924249 1D+02, 4.432962888974640885 8D+03, &
     &       3.147465575029527882 5D+04, 9.741882976226807578 4D+04, &
     &       1.514464467352015780 1D+05, 1.268983958797759872 7D+05, &
     &       5.882461678585702775 2D+04, 1.484722837180236095 7D+04, &
     &       1.882189084098271369 6D+03, 9.255659917730483981 1D+01/
!---------------------------------------------------------------------
      X = ARG
      IF (X .GT. ZERO) THEN
           IF (X .LE. ONE) THEN
!---------------------------------------------------------------------
!     0.0 .LT.  ARG  .LE. 1.0
!---------------------------------------------------------------------
                TEMP = LOG(X)
                IF (X .LT. XSMALL) THEN
!---------------------------------------------------------------------
!     Return for small ARG
!---------------------------------------------------------------------
                   RESULT = P(6)/Q(2) - TEMP
                ELSE
                   XX = X * X
                   SUMP = ((((P(1)*XX + P(2))*XX + P(3))*XX +  &
                         P(4))*XX + P(5))*XX + P(6)
                   SUMQ = (XX + Q(1))*XX + Q(2)
                   SUMF = ((F(1)*XX + F(2))*XX + F(3))*XX + F(4)
                   SUMG = ((XX + G(1))*XX + G(2))*XX + G(3)
                   RESULT = SUMP/SUMQ - XX*SUMF*TEMP/SUMG - TEMP
                END IF
             ELSE IF (X .GT. XMAX) THEN
!---------------------------------------------------------------------
!     Error return for ARG .GT. XMAX
!---------------------------------------------------------------------
                RESULT = ZERO
             ELSE
!---------------------------------------------------------------------
!     1.0 .LT. ARG
!---------------------------------------------------------------------
                XX = ONE / X
                SUMP = PP(1)
                DO 120 I = 2, 10
                   SUMP = SUMP*XX + PP(I)
  120            CONTINUE
                SUMQ = XX
                DO 140 I = 1, 9
                   SUMQ = (SUMQ + QQ(I))*XX
  140           CONTINUE
                SUMQ = SUMQ + QQ(10)
                RESULT = SUMP / SUMQ / SQRT(X)
                RESULT = RESULT * EXP(-X)
           END IF
        ELSE
```

```
!-----------------------------------------------------------------------
!     Error return for ARG .LE. 0.0
!-----------------------------------------------------------------------
            RESULT = XINF
      END IF
      netlibk0 = RESULT
!-----------------------------------------------------------------------
!     Update error counts, etc.
!-----------------------------------------------------------------------
      END
```

Chapter 14
Approximate value for Modified Bessel functions of the first kind and order zero

Function netlibk0(ARG)
File : NETLIBK0.f90

```
DOUBLE PRECISION FUNCTION netlibk0(ARG)
!---------------------------------------------------------------------
!
! This function program computes approximate values for the
!   modified Bessel function of the second kind of order zero
!   for arguments 0.0 .LT. ARG .LE. XMAX (see comments heading
!   CALCK0).
!
!  Authors: W. J. Cody and Laura Stoltz
!
!  Latest Modification: January 19, 1988
!
!---------------------------------------------------------------------
!---------------------------------------------------------------------
!
! This packet computes modified Bessel functions of the second kind
!   and order zero, K0(X) and EXP(X)*K0(X), for real
!   arguments X.  It contains two function type subprograms, BESK0
!   and BESEK0, and one subroutine type subprogram, CALCK0.
!   the calling statements for the primary entries are
!
!                  Y=BESK0(X)
!   and
!                  Y=BESEK0(X)
!
!   where the entry points correspond to the functions K0(X) and
!   EXP(X)*K0(X), respectively.  The routine CALCK0 is
!   intended for internal packet use only, all computations within
!   the packet being concentrated in this routine.  The function
!   subprograms invoke CALCK0 with the statement
!         CALL CALCK0(ARG,RESULT,JINT)
!   where the parameter usage is as follows
!
!      Function                    Parameters for CALCK0
!       Call              ARG                 RESULT          JINT
!
!     BESK0(ARG)    0 .LT. ARG .LE. XMAX       K0(ARG)           1
!     BESEK0(ARG)      0 .LT. ARG           EXP(ARG)*K0(ARG)     2
!
!   The main computation evaluates slightly modified forms of near
!   minimax rational approximations generated by Russon and Blair,
!   Chalk River (Atomic Energy of Canada Limited) Report AECL-3461,
!   1969.  This transportable program is patterned after the
!   machine-dependent FUNPACK packet NATSK0, but cannot match that
!   version for efficiency or accuracy.  This version uses rational
!   functions that theoretically approximate K-SUB-0(X) to at
!   least 18 significant decimal digits.  The accuracy achieved
!   depends on the arithmetic system, the compiler, the intrinsic
!   functions, and proper selection of the machine-dependent
!   constants.
!
!*******************************************************************
```

```
!*****************************************************************
!
! Explanation of machine-dependent constants
!
!   beta   = Radix for the floating-point system
!   minexp = Smallest representable power of beta
!   maxexp = Smallest power of beta that overflows
!   XSMALL = Argument below which BESK0 and BESEK0 may
!            each be represented by a constant and a log.
!            largest X such that  1.0 + X = 1.0  to machine
!            precision.
!   XINF   = Largest positive machine number; approximately
!            beta**maxexp
!   XMAX   = Largest argument acceptable to BESK0;  Solution to
!            equation:
!               W(X) * (1-1/8X+9/128X**2) = beta**minexp
!            where  W(X) = EXP(-X)*SQRT(PI/2X)
!
!
!     Approximate values for some important machines are:
!
!
!                          beta       minexp       maxexp
!
!  CRAY-1         (S.P.)       2        -8193         8191
!  Cyber 180/185
!    under NOS    (S.P.)       2         -975         1070
!  IEEE (IBM/XT,
!    SUN, etc.)   (S.P.)       2         -126          128
!  IEEE (IBM/XT,
!    SUN, etc.)   (D.P.)       2        -1022         1024
!  IBM 3033       (D.P.)      16          -65           63
!  VAX D-Format   (D.P.)       2         -128          127
!  VAX G-Format   (D.P.)       2        -1024         1023
!
!
!                        XSMALL       XINF         XMAX
!
! CRAY-1         (S.P.)    3.55E-15   5.45E+2465    5674.858
! Cyber 180/855
!   under NOS    (S.P.)    1.77E-15   1.26E+322      672.788
! IEEE (IBM/XT,
!   SUN, etc.)   (S.P.)    5.95E-8    3.40E+38        85.337
! IEEE (IBM/XT,
!   SUN, etc.)   (D.P.)    1.11D-16   1.79D+308      705.342
! IBM 3033       (D.P.)    1.11D-16   7.23D+75       177.852
! VAX D-Format   (D.P.)    6.95D-18   1.70D+38        86.715
! VAX G-Format   (D.P.)    5.55D-17   8.98D+307      706.728
!
!*****************************************************************
!*****************************************************************
!
```

```
! Error returns
!
!  The program returns the value XINF for ARG .LE. 0.0, and the
!  BESK0 entry returns the value 0.0 for ARG .GT. XMAX.
!
!
!  Intrinsic functions required are:
!
!     EXP, LOG, SQRT
!
!  Latest modification: March 19, 1990
!
!  Authors: W. J. Cody and Laura Stoltz
!           Mathematics and Computer Science Division
!           Argonne National Laboratory
!           Argonne, IL 60439
!
!----------------------------------------------------------------------
      INTEGER I
      DOUBLE PRECISION   &
         ARG,F,G,ONE,P,PP,Q,QQ,RESULT,SUMF,SUMG,SUMP,SUMQ,TEMP,  &
         X,XINF,XMAX,XSMALL,XX,ZERO
      DIMENSION P(6),Q(2),PP(10),QQ(10),F(4),G(3)
!----------------------------------------------------------------------
!  Mathematical constants
!----------------------------------------------------------------------
      DATA ONE/1.0D0/,ZERO/0.0D0/
!----------------------------------------------------------------------
!  Machine-dependent constants
!----------------------------------------------------------------------
      DATA XSMALL/1.11D-16/,XINF/1.79D+308/,XMAX/705.342D0/
!----------------------------------------------------------------------
!
!     Coefficients for XSMALL .LE.  ARG  .LE. 1.0
!
!----------------------------------------------------------------------
      DATA    P/ 5.8599221412826100000D-04, 1.1660525649895718 50D-01, &
              1.1999463724910714109D+01, 4.6850901201934832188D+02, &
              5.9169059852270512312D+03, 2.4708152720399552679D+03/
      DATA    Q/-2.4994418972832303646D+02, 2.1312714303849120380D+04/
      DATA    F/-1.6414452837299064100D+00,-2.9601657892958843866D+02, &
             -1.7733784684952985886D+04,-4.0320340761145482298D+05/
      DATA    G/-2.5064972445877992730D+02, 2.9865713163054025489D+04, &
             -1.6128136304458193998D+06/
!----------------------------------------------------------------------
!
!     Coefficients for  1.0 .LT. ARG
!
!----------------------------------------------------------------------
      DATA  PP/ 1.1394980557384778174D+02, 3.6832589957340267940D+03, &
              3.1075408980684392399D+04, 1.0577068948034021957D+05, &
              1.7398867902565686251D+05, 1.5097646353289914539D+05, &
```

```
             7.155706278376403754 1D+04, 1.832152587018353772 5D+04, &
             2.344473876419931502 1D+03, 1.160024942507603555 8D+02/
      DATA  QQ/ 2.001344306494924249 1D+02, 4.432962888974640885 8D+03, &
             3.147465575029527882 5D+04, 9.741882976226807578 4D+04, &
             1.514464467352015780 1D+05, 1.268983958797759872 7D+05, &
             5.882461678585702775 2D+04, 1.484722837180236095 7D+04, &
             1.882189084098271369 6D+03, 9.255659917730483981 1D+01/
!-----------------------------------------------------------------------
      X = ARG
      IF (X .GT. ZERO) THEN
          IF (X .LE. ONE) THEN
!-----------------------------------------------------------------------
!     0.0 .LT.  ARG  .LE. 1.0
!-----------------------------------------------------------------------
              TEMP = LOG(X)
              IF (X .LT. XSMALL) THEN
!-----------------------------------------------------------------------
!     Return for small ARG
!-----------------------------------------------------------------------
                  RESULT = P(6)/Q(2) - TEMP
                ELSE
                  XX = X * X
                  SUMP = ((((P(1)*XX + P(2))*XX + P(3))*XX +  &
                        P(4))*XX + P(5))*XX + P(6)
                  SUMQ = (XX + Q(1))*XX + Q(2)
                  SUMF = ((F(1)*XX + F(2))*XX + F(3))*XX + F(4)
                  SUMG = ((XX + G(1))*XX + G(2))*XX + G(3)
                  RESULT = SUMP/SUMQ - XX*SUMF*TEMP/SUMG - TEMP
              END IF
            ELSE IF (X .GT. XMAX) THEN
!-----------------------------------------------------------------------
!     Error return for ARG .GT. XMAX
!-----------------------------------------------------------------------
              RESULT = ZERO
            ELSE
!-----------------------------------------------------------------------
!     1.0 .LT. ARG
!-----------------------------------------------------------------------
              XX = ONE / X
              SUMP = PP(1)
              DO 120 I = 2, 10
                SUMP = SUMP*XX + PP(I)
 120           CONTINUE
              SUMQ = XX
              DO 140 I = 1, 9
                SUMQ = (SUMQ + QQ(I))*XX
 140           CONTINUE
              SUMQ = SUMQ + QQ(10)
              RESULT = SUMP / SUMQ / SQRT(X)
              RESULT = RESULT * EXP(-X)
          END IF
        ELSE
```

```
!-----------------------------------------------------------------------
!     Error return for ARG .LE. 0.0
!-----------------------------------------------------------------------
            RESULT = XINF
      END IF
      netlibk0 = RESULT
!-----------------------------------------------------------------------
!     Update error counts, etc.
!-----------------------------------------------------------------------
      END
```

Chapter 15
Polygamma function of order m

Function polygam(m, x, ierr)
File : polygam.f90

```
function polygam( m, x, ierr )

!------------------------------------------------------------------------
!
!     Computes the Polygamma function of order m, i.e.
!     the mth derivative of the digamma (psi) function, i.e.
!     the (m+1)th derivative of the logarithm of the gamma function.
!
!     M     - Input . Order of the function       (0<=M<=100) - Integer
!     X     - Input . Value of the variable            (X>0) - Real
!     IER   - Output. Return code :                          - Integer
!                     0 = normal
!                     1 = invalid input argument
!                         (then fcdf = 0.)
!                     2 = fatal error (underflow or overflow)
!                     3 = auxiliary array in dpsifn not large
!                         enough
!
!     Subroutines called:
!       DPSIFN
!     Fortran functions called:
!       GAMMA  MOD
!
!     Uses Algorithm 610, collected algorithms from ACM.
!     Algorithm appeared in ACM-Trans. Math. Software, vol.9, no. 4,
!     dec., 1983, p. 494-502.
!
!------------------------------------------------------------------------

   implicit none

   ! Function
   ! --------

   real(kind=8) :: polygam

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

   real(kind=8), intent(in) :: x
   integer, intent(in) :: m
   integer, intent(out) :: ierr

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

   real(kind=8), parameter :: zero=0.0_8, one=1.0_8
   real(kind=8) :: w(1)
   integer :: kode, n

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

   if ( x <= zero .or. m < 0 .or. m > 100 ) then
```

```
      ierr = 1
      polygam = zero
      return
    end if
    call dpsifn( x, m, 1, 1, w, ierr )
    if ( ierr == 0 ) then
      polygam = w(1)*gamma(m+one)
      if ( mod(m,2) == 0 ) polygam = -polygam
    else
      polygam = zero
    end if

  end function polygam

  subroutine dpsifn(x, n, kode, m, ans, iflag)

  !    WRITTEN BY D.E. AMOS, JUNE, 1982.
  !
  !    REFERENCES
  !       HANDBOOK OF MATHEMATICAL FUNCTIONS, AMS 55, NATIONAL BUREAU
  !       OF STANDARDS BY M. ABRAMOWITZ AND I.A. STEGUN, 1964, PP.
  !       258-260, EQTNS. 6.3.5, 6.3.18, 6.4.6, 6.4.9, 6.4.10
  !
  !       ACM TRANS. MATH SOFTWARE, 1983.
  !
  !    ABSTRACT     *** A DOUBLE PRECISION ROUTINE ***
  !       DPSIFN COMPUTES M MEMBER SEQUENCES OF SCALED DERIVATIVES OF
  !       THE PSI FUNCTION
  !
  !           W(K,X)=(-1)**(K+1)*PSI(K,X)/GAMMA(K+1)
  !
  !       K=N,...,N+M-1 WHERE PSI(K,X) IS THE K-TH DERIVATIVE OF THE PSI
  !       FUNCTION.  ON KODE=1, DPSIFN RETURNS THE SCALED DERIVATIVES
  !       AS DESCRIBED.  KODE=2 IS OPERATIVE ONLY WHEN K=0 AND DPSIFN
  !       RETURNS -PSI(X) + LN(X).  THAT IS, THE LOGARITHMIC BEHAVIOR
  !       FOR LARGE X IS REMOVED WHEN KODE=2 AND K=0. WHEN SUMS OR
  !       DIFFERENCES OF PSI FUNCTIONS ARE COMPUTED, THE LOGARITHMIC
  !       TERMS CAN BE COMBINED ANALYTICALLY AND COMPUTED SEFARATELY
  !       TO HELP RETAIN SIGNIFICANT DIGITS.
  !
  !       THE BASIC METHOD OF EVALUATION IS THE ASYMPTOTIC EXPANSION
  !       FOR LARGE X.GE.XMIN FOLLOWED BY BACKWARD RECURSION ON A TWO
  !       TERM RECURSION RELATION
  !
  !               W(X+1) + X**(-N-1) = W(X).
  !
  !       THIS IS SUPPLEMENTED BY A SERIES
  !
  !               SUM( (X+K)**(-N-1) , K=0,1,2,... )
  !
  !       WHICH CONVERGES RAPIDLY FOR LARGE N. BOTH XMIN AND THE
  !       NUMBER OF TERMS OF THE SERIES ARE CALCULATED FROM THE UNIT
```

```
!       ROUND OFF OF THE MACHINE ENVIRONMENT.
!
!       FUNCTIONS I1MACH AND D1MACH MUST BE INITIALIZED ACCORDING
!       TO  PROLOGUE INSTRUCTIONS. FIFTEEN MACHINE ENVIRONMENTS
!       CAN BE DEFINED BY MAKING APPROPRIATE COMMENT CARDS ACTIVE
!       IN I1MACH AND D1MACH.
!
!       THE NOMINAL COMPUTATIONAL ACCURACY IS THE MAXIMUM OF UNIT
!       ROUNDOFF (=D1MACH(4)) AND 1.0D-18 SINCE CRITICAL CONSTANTS
!       ARE GIVEN TO ONLY 18 DIGITS.
!
!       DPSIFN CALLS I1MACH, D1MACH, XERROR PACKAGE(18 ROUTINES)
!
!       PSIFN IS THE SINGLE PRECISION VERSION OF DPSIFN.
!
!    DESCRIPTION OF ARGUMENTS
!
!       INPUT      X IS DOUBLE PRECISION
!         X      - ARGUMENT, X.GT.0.0D0
!         N      - FIRST MEMBER OF THE SEQUENCE, 0.LE.N.LE.100
!                  N=0 GIVES ANS(1) = -PSI(X)       ON KODE=1
!                                     -PSI(X)+LN(X) ON KODE=2
!         KODE   - SELECTION PARAMETER
!                  KODE=1 RETURNS SCALED DERIVATIVES OF THE PSI
!                  FUNCTION
!                  KODE=2 RETURNS SCALED DERIVATIVES OF THE PSI
!                  FUNCTION EXCEPT WHEN N=0. IN THIS CASE,
!                  ANS(1) = -PSI(X) + LN(X) IS RETURNED.
!         M      - NUMBER OF MEMBERS OF THE SEQUENCE, M.GE.1
!
!       OUTPUT     ANS IS DOUBLE PRECISION
!         ANS    - A VECTOR OF LENGTH AT LEAST M WHOSE FIRST M
!                  COMPONENTS CONTAIN THE SEQUENCE OF  DERIVATIVES
!                  SCALED ACCORDING TO KODE
!
!         iflag  - error condition
!       ERROR CONDITIONS
!         AN IMPROPER INPUT IS A FATAL ERROR
!         AN OVERFLOW IS A FATAL ERROR
!***END PROLOGUE

   double precision, intent(in) :: x
   integer, intent(in) :: n, kode, m
   double precision, intent(out) :: ans(m)
   integer, intent(out) :: iflag

!! double precision, external :: d1mach
!! integer, external :: i1mach

   integer, parameter :: nmax=100
   integer :: i, j, k, kontr, mx, nn, np, nx
   double precision, parameter :: zero=0.0d0, half=0.5d0, one=1.0d0
```

```
      double precision :: arg, den, elim, eps, fln, fn, fnp, fns,      &
        fx, rln, rxsq, r1m4, r1m5, s, slope, t, ta, tk, tol, tols,     &
        tss, tst, tt, t1, t2, wdtol, xdmln, xdmy, xinc, xln,           &
        xm, xmin, xq, yint
      double precision :: trm(22), trmr(nmax)
!-----------------------------------------------------------------------
!            BERNOULLI NUMBERS
!-----------------------------------------------------------------------
      double precision :: b(22)=(/1.00000000000000000d+00, &
       -5.00000000000000000d-01,1.66666666666666667d-01,   &
       -3.33333333333333333d-02,2.38095238095238095d-02,   &
       -3.33333333333333333d-02,7.57575757575757576d-02,   &
       -2.53113553113553114d-01,1.16666666666666667d+00,   &
       -7.09215686274509804d+00,5.49711779448621554d+01,   &
       -5.29124242424242424d+02,6.19212318840579710d+03,   &
       -8.65802531135531136d+04,1.42551716666666667d+06,   &
       -2.72982310678160920d+07,6.01580873900642368d+08,   &
       -1.51163157670921569d+10,4.29614643061166667d+11,   &
       -1.37116552050883328d+13,4.88332318973593167d+14,   &
       -1.92965793419400681d+16/)

      iflag = 1
      if (x.le.zero .or. n.lt.0 .or. kode.lt.1 .or. kode.gt.2 .or. m.lt.1) return
      iflag = 0
      nn = n + m - 1
      fn = dble(float(nn))
      fnp = fn + one
!! nx = min0(-i1mach(15),i1mach(16))
      nx = min0(-minexponent(zero),maxexponent(zero))
!! r1m5 = d1mach(5)
      r1m5 = log10(dble(radix(zero)))
!! r1m4 = d1mach(4)*half
      r1m4 = epsilon(zero)*half
      wdtol = dmax1(r1m4,0.5d-18)
!-----------------------------------------------------------------------
!     ELIM = APPROXIMATE EXPONENTIAL OVER AND UNDERFLOW LIMIT
!-----------------------------------------------------------------------
      elim = 2.302d0*(dble(float(nx))*r1m5-3.0d0)
      xln = dlog(x)
      t = fnp*xln
!-----------------------------------------------------------------------
!     OVERFLOW AND UNDERFLOW TEST FOR SMALL AND LARGE X
!-----------------------------------------------------------------------
      if (dabs(t).gt.elim) then
        iflag = 2
        return
      end if
      if (x.lt.wdtol) go to 40
!-----------------------------------------------------------------------
!     COMPUTE XMIN AND THE NUMBER OF TERMS OF THE SERIES, FLN+1
!-----------------------------------------------------------------------
!! rln = r1m5*dble(float(i1mach(14)))
```

```
      rln = r1m5*dble(float(digits(zero)))
      rln = dmin1(rln,18.06d0)
      fln = dmax1(rln,3.0d0) - 3.0d0
      yint = 3.50d0 + 0.40d0*fln
      slope = 0.21d0 + fln*(0.0006038d0*fln+0.008677d0)
      xm = yint + slope*fn
      mx = int(sngl(xm)) + 1
      xmin = dble(float(mx))
      if (n > 0) then
        xm = -2.302d0*rln - dmin1(zero,xln)
        fns = dble(float(n))
        arg = xm/fns
        arg = dmin1(zero,arg)
        eps = dexp(arg)
        xm = one - eps
        if (dabs(arg).lt.1.0d-3) xm = -arg
        fln = x*xm/eps
        xm = xmin - x
        if (xm.gt.7.0d0 .and. fln.lt.15.0d0) go to 30
      end if
      xdmy = x
      xdmln = xln
      xinc = zero
      if (x < xmin) then
        nx = int(sngl(x))
        xinc = xmin - dble(float(nx))
        xdmy = x + xinc
        xdmln = dlog(xdmy)
      end if
!-----------------------------------------------------------------------
!     GENERATE W(N+M-1,X) BY THE ASYMPTOTIC EXPANSION
!-----------------------------------------------------------------------
      t = fn*xdmln
      t1 = xdmln + xdmln
      t2 = t + xdmln
      tk = dmax1(dabs(t),dabs(t1),dabs(t2))
      if (tk.gt.elim) then
        iflag = 2
        return
      end if
      tss = dexp(-t)
      tt = half/xdmy
      t1 = tt
      tst = wdtol*tt
      if (nn.ne.0) t1 = tt + one/fn
      rxsq = one/(xdmy*xdmy)
      ta = half*rxsq
      t = fnp*ta
      s = t*b(3)
      if (dabs(s) >= tst) then
        tk = 2.0d0
        do k=4,22
```

```
        t = t*((tk+fn+one)/(tk+one))*((tk+fn)/(tk+2.0d0))*rxsq
        trm(k) = t*b(k)
        if (dabs(trm(k)).lt.tst) exit
        s = s + trm(k)
        tk = tk + 2.0d0
      end do
    end if
    s = (s+t1)*tss
    if (xinc /= zero) then
  !---------------------------------------------------------------------
  !     BACKWARD RECUR FROM XDMY TO X
  !---------------------------------------------------------------------
      nx = int(sngl(xinc))
      np = nn + 1
      if (nx.gt.nmax) then
        iflag = 3
        return
      end if
      if (nn.eq.0) go to 10
      xm = xinc - one
      fx = x + xm
  !---------------------------------------------------------------------
  !     THIS LOOP SHOULD NOT BE CHANGED. FX IS ACCURATE WHEN X IS SMALL
  !---------------------------------------------------------------------
      do i=1,nx
        trmr(i) = fx**(-np)
        s = s + trmr(i)
        xm = xm - one
        fx = x + xm
      end do
    end if
    ans(m) = s
    if (fn.eq.zero) go to 20
  !---------------------------------------------------------------------
  !     GENERATE LOWER DERIVATIVES, J.LT.N+M-1
  !---------------------------------------------------------------------
    if (m.eq.1) return
    do j=2,m
      fnp = fn
      fn = fn - one
      tss = tss*xdmy
      t1 = tt
      if (fn.ne.zero) t1 = tt + one/fn
      t = fnp*ta
      s = t*b(3)
      if (dabs(s) >= tst) then
        tk = 3.0d0 + fnp
        do k=4,22
          trm(k) = trm(k)*fnp/tk
          if (dabs(trm(k)).lt.tst) exit
          s = s + trm(k)
          tk = tk + 2.0d0
```

```
        end do
      end if
      s = (s+t1)*tss
      if (xinc /= zero) then
        if (fn.eq.zero) go to 10
        xm = xinc - one
        fx = x + xm
        do i=1,nx
          trmr(i) = trmr(i)*fx
          s = s + trmr(i)
          xm = xm - one
          fx = x + xm
        end do
      end if
      mx = m - j + 1
      ans(mx) = s
      if (fn.eq.zero) go to 20
    end do
    return
!-----------------------------------------------------------------------
!     RECURSION FOR N = 0
!-----------------------------------------------------------------------
10  continue
    do i=1,nx
      s = s + one/(x+dble(float(nx-i)))
    end do
20  continue
    if (kode == 1) then
      ans(1) = s - xdmln
      return
    end if
    if (xdmy.eq.x) return
    xq = xdmy/x
    ans(1) = s - dlog(xq)
    return
!-----------------------------------------------------------------------
!     COMPUTE BY SERIES (X+K)**(-(N+1)) , K=0,1,2,...
!-----------------------------------------------------------------------
30  continue
    nn = int(sngl(fln)) + 1
    np = n + 1
    t1 = (fns+one)*xln
    t = dexp(-t1)
    s = t
    den = x
    do i=1,nn
      den = den + one
      trm(i) = den**(-np)
      s = s + trm(i)
    end do
    ans(1) = s
    if (n == 0) then
```

```
      if (kode.eq.2) ans(1) = s + xln
    end if
    if (m.eq.1) return
!-----------------------------------------------------------------------
!     GENERATE HIGHER DERIVATIVES, J.GT.N
!-----------------------------------------------------------------------
    tol = wdtol/5.0d0
    do j=2,m
      t = t/x
      s = t
      tols = t*tol
      den = x
      do i=1,nn
        den = den + one
        trm(i) = trm(i)/den
        s = s + trm(i)
        if (trm(i).lt.tols) exit
      end do
      ans(j) = s
    end do
    return
!-----------------------------------------------------------------------
!     SMALL X.LT.UNIT ROUND OFF
!-----------------------------------------------------------------------
  40 continue
    ans(1) = x**(-n-1)
    if (m > 1) then
      k = 1
      do i=2,m
        ans(k+1) = ans(k)/x
        k = k + 1
      end do
    end if
    if (n.ne.0) return
    if (kode.eq.2) ans(1) = ans(1) + xln

  end subroutine dpsifn
```