```
{$N+}
unit CalcDistSpecFun;

interface

uses Math;

//TODO move this and other support functions in CalcDists into the more general
numerics lib.
//Chaff in here 6/96
const

  FLOATEPS = 3.4E-9; // TODO check this since this is old code...

  CBRT2 = 1.2599210498948731647672; { cube root of 2 }
  CBRT3 = 1.4422495703074083823216; { cube root of 3 }
  D2R = 0.017453292519943295769237; { # radians in a degree }
  E = 2.7182818284590452353603; { base of natural logs }
  EULERC = 0.577215664901532860606 5; { Euler's constant }
  LN2 = 0.693147180559945309417 2; { natural logarithm of 2 }
  LN3 = 1.098612288668109691395 2; { natural logarithm of 3 }
  LN10 = 2.302585092994045684028 0; { natural logarithm of 10 }
  LNPI = 1.144729885849400174143 4; { natural logarithm of PI }
  LOG2E = 1.442695040888963407359 9; { 1/ln(2) }
  LOG10E = 0.434294481903251827651 1; { 1/ln(10) }
  M2R = 0.00029088820866572159615 4; { # rads in a minute }
  PI = 3.141592653589793238462 6; { plain ol' round pi }
  PI_2 = 1.570796326794896619231 3; { pi/2 }
  PI_4 = 0.785398163397448309615 6; { pi/4 }
  R1_E = 0.367879441171442321595 5; { 1/e }
  R1_PI = 0.318309886183790671537 8; { 1/pi }
  R1_SQRTPI = 0.564189583547756286948 1; { 1/sqrt(pi) }
  R2_PI = 0.636619772367581343075 5; { 2/pi }
  R2_SQRTPI = 1.128379167095512573907 0; { 2/sqrt(pi) }
  R2D = 57.295779513082320876798 ; { # degrees in a radian }
  S2R = 0.000004848136811095359936 ; { # rads in a second }
  SQRPI = 9.869604401089358618834 5; { sqr(pi) (not round!) }
  SQRT_2 = 0.707106781186547524401 ; { sqrt(1/2) }
  SQRT2 = 1.414213562373095048801 7; { sqrt(2) }
  SQRT2PI = 2.506628274631000502415 8; { sqrt(2*pi) }
  SQRT3 = 1.732050807568877293527 ; { sqrt(3) }
  SQRT5 = 2.236067977499789696409 ; { sqrt(5) }
  SQRTPI = 1.772453850905516027298 2; { sqrt(pi) = gamma(1/2) }
  TWOPI = 6.283185307179586476925 3; { 2*pi }

type
  float = double;
  xfloat = extended;

function factorial(n: word): xfloat;
```

```
function lnfactorial(n: word): xfloat;

function permutation(n, k: word): xfloat;
function combination(n, k: word): xfloat;

function calcgamma(x: xfloat): xfloat;
function lngamma(x: xfloat): xfloat;

function beta(x, y: xfloat): xfloat;
function lnbeta(x, y: xfloat): xfloat;

function igamma(a, x: xfloat): xfloat;
function igammac(a, x: xfloat): xfloat;
function igammap(a, x: xfloat): xfloat;
function igammaq(a, x: xfloat): xfloat;

function ibeta(a, b, x: xfloat): xfloat;
function ibetap(a, b, x: xfloat): xfloat;
function ibetaq(a, b, x: xfloat): xfloat;

function erf(x: xfloat): xfloat;
function erfc(x: xfloat): xfloat;

implementation

{ IBETACF is a "continued fraction" evaluation of an incomplete
  beta auxiliary function. (See HMF: eqn. 26.5.8; NR: BETACF ) }

function ibetacf(a, b, x: xfloat): xfloat;
var
  am, ap, apb, app, azlast, az, bm, bp, bpp, bz, d, deca, inca: xfloat;
  m, tm: longint;
begin
  am := 1;
  az := 1;
  bm := 1;
  m := 1;
  apb := a + b;
  deca := a - 1;
  inca := a + 1;
  bz := 1 - apb * x / inca;
  tm := 2;
  d := (b - 1) * x / (inca * (a + 2));
  ap := d + 1;
  bp := bz + d;
  d := -inca * (apb + 1) * x / ((a + 2) * (a + 3));
  app := ap + d;
  bpp := bp + d * bz;
  am := ap / bpp;
  bm := bp / bpp;
```

```
  azlast := az;
  az := app / bpp;
  while abs(az - azlast) > (abs(az) * FLOATEPS) do begin
    inc(m);
    inc(tm, 2);
    d := m * (b - m) * x / ((deca + tm) * (a + tm));
    ap := az + d * am;
    bp := d * bm + 1;
    d := -(a + m) * (apb + m) * x / ((a + tm) * (inca + tm));
    app := ap + d * az;
    bpp := bp + d
  end;
  ibetacf := az
end; { ibetacf }

{ IGAMMACF is a "continued fraction" evaluation of an incomplete
  gamma auxiliary function.  ( See HMF: eqn. 6.5.31; NR: GCF ) }

function igammacf(a, x: xfloat): xfloat;
var
  n: longint;
  a0, a1, b0, b1, f, fn, g, glast, na: xfloat;
begin
  a0 := 1;
  a1 := x;
  b0 := 0;
  b1 := 1;
  f := 1;
  g := 0;
  glast := 1;
  n := 0;
  repeat
    inc(n);
    fn := f * n;
    na := n - a;
    a0 := (a1 + a0 * na) * f;
    b0 := (b1 + b0 * na) * f;
    a1 := x * a0 + fn * a1;
    b1 := x * b0 + fn * b1;
    if (a1 <> 0) then begin
      f := 1 / a1;
      glast := g;
      g := b1 * f
    end;
  until abs(g - glast) < (abs(g) * FLOATEPS);
  igammacf := g;
end; { igammacf }

{ IGAMMASER is a "power series" evaluation of the incomplete
  gamma auxiliary function.  ( See HMF: eqn. 6.5.29; NR: GSER ) }
```

```
function igammaser(a, x: xfloat): xfloat;
var
  d, s: xfloat;
begin
  if x = 0 then
    igammaser := 0
  else begin
    d := 1 / a;
    s := d;
    repeat
      a := a + 1;
      d := d * x / a;
      s := s + d;
    until abs(d) < (abs(s) * FLOATEPS);
    igammaser := s
  end
end; { igammaser }

{ *** interface'd functions in alphabetical order *** }

function beta;
begin
  beta := exp(lnbeta(x, y))
end;

function combination;
begin
  combination := round(exp(lnfactorial(n) - lnfactorial(k) -
    lnfactorial(n - k)))
end;

function iif(p: boolean; t, f: xfloat): xfloat;
begin
  if p then
    iif := t
  else
    iif := f
end;

function sgn(const x: xfloat): xfloat;
begin
  if x = 0 then
    sgn := 0
  else
    sgn := iif(x > 0, 1, -1)
end;

function erf;
var
```

```
  y, z: xfloat;
begin
  if abs(x) < FLOATEPS then
    erf := x * R2_SQRTPI
  else
    if abs(x) > 10 then
      erf := sgn(x)
    else begin
      y := sqr(x);
      z := x * exp(-y - SQRTPI);
      if y < 1.5 then
        erf := z * igammaser(0.5, y)
      else
        erf := sgn(x) - z * igammacf(0.5, y)
    end
end; { erf }

function erfc;
begin
  erfc := 1 - erf(x)
end;

function factorial;
const
  MAXATYPE = 32;
type
  atype = array [0 .. MAXATYPE] of xfloat;
var
  a: ^atype;
  atop: shortint;
  j: byte;
begin
  a := nil;
  atop := -1;
  if a = NIL then begin
    new(a);
    if a <> NIL then begin
      a^[0] := 1;
      atop := 0
    end
  end;
  if (a <> NIL) and (n > atop) and (n <= MAXATYPE) then begin
    for j := succ(atop) to n do
      a^[j] := a^[pred(j)] * j;
    atop := n
  end;
  if (n <= atop) then
    factorial := a^[n]
  else
    factorial := round(calcgamma(succ(n)))
```

```
end; { factorial }

function calcgamma;
var
  y: xfloat;
begin
  y := exp(lngamma(x));
  if x < 0 then
    if not odd(trunc(x)) then
      y := -y;
  calcgamma := y
end;

function ibeta;
var
  y: xfloat;
begin
  if x <= 0 then
    ibeta := 0
  else
    if x >= 1 then
      ibeta := beta(a, b)
    else begin
      y := exp(a * ln(x) + b * ln(1 - x));
      if (x * (a + b + 2)) <= (a + 1) then
        ibeta := y * ibetacf(a, b, x) / a
      else
        ibeta := beta(a, b) - y * ibetacf(b, a, 1 - x) / b
    end
end; { ibeta }

function ibetap;
var
  y: xfloat;
begin
  if x <= 0 then
    ibetap := 0
  else
    if x >= 1 then
      ibetap := 1
    else begin
      y := exp(-lnbeta(a, b) + a * ln(x) + b * ln(1 - x));
      if x <= (a + 1) / (a + b + 2) then
        ibetap := y * ibetacf(a, b, x) / a
      else
        ibetap := 1 - y * ibetacf(b, a, 1 - x) / b
    end
end; { ibetap }

function ibetaq;
```

```
begin
  ibetaq := 1 - ibetap(b, a, 1 - x)
end;

function igamma;
begin
  if x <= 0 then
    igamma := 0
  else
    if x < (a + 1) then
      igamma := exp(-x + a * ln(x)) * igammaser(a, x)
    else
      igamma := calcgamma(a) - igammac(a, x)
end;

function igammac;
begin
  if x < (a + 1) then
    igammac := calcgamma(a) - igamma(a, x)
  else
    igammac := exp(-x + a * ln(x)) * igammacf(a, x)
end;

function igammap;
begin
  if x <= 0 then
    igammap := 0
  else
    if x < (a + 1) then
      igammap := exp(-x + a * ln(x) - lngamma(a)) * igammaser(a, x)
    else
      igammap := 1 - igammaq(a, x)
end;

function igammaq;
begin
  if x < (a + 1) then
    igammaq := 1 - igammap(a, x)
  else
    igammaq := exp(-x + a * ln(x) - lngamma(a)) * igammacf(a, x)
end;

function lnbeta;
begin
  lnbeta := lngamma(x) + lngamma(y) - lngamma(x + y)
end;

function lnfactorial;
const
  MAXATYPE = 99;
```

```
type
  atype = array [0 .. MAXATYPE] of xfloat;
var
  a: ^atype;
  j: byte;
begin
  a := nil;
  if a = NIL then begin
    new(a);
    if a <> NIL then
      for j := 0 to MAXATYPE do
        a^[j] := -1
  end;
  if (n < MAXATYPE) and (a <> NIL) then begin
    if a^[n] < 0 then
      a^[n] := lngamma(succ(n));
    lnfactorial := a^[n]
  end
  else
    lnfactorial := lngamma(succ(n))
end; { lnfactorial }

function lngamma;
var
  s, t: xfloat;
begin
  if x < 0 then
    lngamma := LNPI - lngamma(1 - x) - ln(sin(frac(x * 0.5) * TWOPI))
  else
    if x < 1 then
      lngamma := lngamma(x + 1) - ln(x)
    else begin
      t := x + 4.5;
      t := (x - 0.5) * ln(t) - t;
      s := 1 + 76.18009173 / x - 86.50532033 / (x + 1) + 24.01409822 / (x + 2) -
        1.231739516 / (x + 3) + 1.20858003E-3 / (x + 4) - 5.3682E-6 / (x + 5);
      lngamma := t + ln(SQRT2PI * s)
    end;
end; { lngamma }

function permutation;
begin
  permutation := round(exp(lnfactorial(n) - lnfactorial(n - k)))
end;

end. { unit specfun/specf87 }
```