```
unit SafeWaterCalcDist;

interface

uses Math,CalcDistSpecFun,SysUtils;

const

   dNone       = 299;
   dCustom     = 300;
   dNormal     = 301;   dTriangular  = 302;   dPoisson  = 303;
   dBinomial   = 304;   dLogNormal   = 305;   dUniform  = 306;
   dExponential= 307;   dGeometric   = 308;   dWeibull  = 309;
   dGamma      = 310;   dLogistic    = 311;   dCauchy   = 314;
   dPareto     = 313;   dBeta        = 312;   dHML      = 315;
   dVariableCustom  = 316;
   dLogUniform =317;

   tiny        = 1.0e-11;
   error_value = -99.9;

{functions from Numerical Recipes, used to calc Beta, Binomial, and Poisson}

function  gammp(a,x: double): double;
function  gammq(a,x:double):double;
procedure gser(a,x:double; var gamser,gln: double);
procedure gcf(a,x: double; var gammcf,gln: double);
function  betacf(a,b,x: double): double;
function  betai(a,b,x:double):double;

{These three functions provide a generalized method of calling icdf,
 cdf, and random functions when the distribution type is variable}
function icdf(dtype:integer;y,parm1,parm2,parm3:double):double;
function cdf(dtype:integer;x,parm1,parm2,parm3:double):double;

{The following suite of functions return a randomnly selected
 value from the respective distrubtion}
function rNormal(Mean, StandardDeviation : double) : double;
function rLogNormal(GM, GSD : double) : double;
function rPoisson(Rate : double) : double;
function rExponential(Rate : double) : double;
function rTriangular(Minimum, Maximum, MostLikely : double) : double;
function rBinomial(Probability, Trials : double) : double;
function rUniform(Minimum, Maximum : double) : double;
function rGeometric(Probability : double) : double;
function rWeibull(scale,shape:double):double;
function rLogistic(Mean, Scale : double) : double;
function rGamma(a,Lamda : double) : double;
function rBeta(aa,bb, Scale : double) : double;
function rCauchy(a,b:double): double;
```

```
function rPareto(x0,Theta : double) : double;
function rHML(High,Medium,Low : double) : double;
function rLogUniform(Minimum, Maximum : double) : double;

{The next set of functions return CVM Stat of the fit for each
 distribution type and parameter estimates. }

function cdfNormal(y,mean,dev:double) : single;
function icdfNormal(Prob,mean,dev:double) : double;

function cdfLogNormal(x,GM,GSD: double): single;
function icdfLogNormal (Prob,GM,GSD:double) : double;

function  icdfTriangular(y,A,B,ML:double): single;
function  cdfTriangular(X,A,B,ML : double) : single;

function cdfUniform(X,A,B : double) : single;
function icdfUniform(Prob,A,B : double) : double;

function cdfExponential(x,lamda : double): single;
function icdfExponential(Prob,lamda : double): double;

function cdfGamma(X,A : double) : single;
function icdfgamma(P,A : double) : double;

function cdfWeibull(XD,scale,shape : double) : single;
function icdfweibull(y,scale,shape:double): double;

function cdfLogistic(x,a,b:double): single;
function icdfLogistic(Prob,a,b:double): double;

function cdfPoisson(xd,lambda:double) : single;
function icdfPoisson(y,lambda:double) : double;

function cdfBinomial(xd,p : double; n : integer): single;
function icdfBinomial(y,p : double; n: integer): double;

function cdfGeometric(xd,p : double): single;
function icdfGeometric(y,p : double): double;

function cdfBeta(x,a,b,scale:double): single;
function icdfBeta(y,a,b,scale:double):double;

function cdfCauchy(x,a,b:double):single;
function icdfCauchy(prob,a,b:double):double;

function cdfPareto(x,x0,theta:double):single;
function icdfPareto(y,x0,theta:double):double;

function cdfHML(x,High, Medium, Low:double):single;
```

```
function icdfHML(prob,High, Medium, Low:double):double;

function cdfLogUniform(x,A,B :double):single;
function icdfLogUniform(prob,A,B:double):double;


implementation

{------------------------------------------------------------------------}

{ NUMERICAL RECIPES FUNCTIONS }

procedure gser(a,x:double; var gamser,gln: double);
   const itmax=1000; eps=3.0e-7;
   var
      n:integer; sum,del,ap: double;
begin
  if x<0 then raise exception.Create('x<0 in CalcDist.gser');
  gln:=lngamma(a);

  if x=0 then begin
    gamser := 0
  end else begin
    ap := a; sum:=1/a; del:=sum;
    for n:=1 to itmax do begin
      ap:= ap+1; del:=del*x/ap; sum:=sum+del;
      if (abs(del) < abs(sum)*eps) then break;
      if n=itmax then raise exception.Create('error in CalcDist.gser ');
    end; {for do loop}
    Gamser:=sum*exp(-x+a*ln(x)-gln)
  end {else}
end;  {gser}

procedure gcf(a,x: double; var gammcf,gln: double);
   const itmax=1000; eps=3.0e-7;
   var n:integer; gold,g,fac,b1,b0,anf,ana,an,a1,a0:double;
begin
  gln:=lngamma(a);
  gold:=0.0; a0:=1.0; a1:=x; b0:=0.0; b1:=1.0; fac:=1.0;
  for n:=1 to itmax do begin
    an:=1.0*n; ana:=an-a; a0:=(a1+a0*ana)*fac;
    b0:=(b1+b0*ana)*fac; anf := an*fac; a1 := x*a0+anf*a1;
    b1:=x*b0+anf*b1;
    if (a1<>0) then begin
      fac:= 1.0/a1; g:=b1*fac;
      if (abs((g-gold)/g) < eps) then break;
      gold :=g
    end;
    if n=itmax then raise exception.Create('error in CalcDist.gcf');
  end;
```

```
  gammcf := exp(-x+a*ln(x)-gln)*g
end;  {gcf}

function gammp(a,x: double): double;
var
  gammcf,gln: double;
begin
   if ((x < 0.0) OR (a <= 0.0)) then begin
     raise exception.Create('calcdist.GAMMP - invalid arguments');
   end;
   if (x < (a+1.0)) then begin
     gser(a,x,gammcf,gln);
     gammp := gammcf
   end else begin
     gcf(a,x,gammcf,gln);
     gammp := 1.0-gammcf
   end
end;

function gammq(a,x:double):double;
var gamser,gln:double;
begin
  if (x<0) or (a<=0) then raise exception.Create('error in CalcDist.gammq');
  if (x<a+1) then begin
    gser(a,x,gamser,gln); gammq:=1-gamser end
  else begin
    gcf(a,x,gamser,gln);
    gammq:=gamser
  end;
end;   { gammq }


function betacf(a,b,x: double): double;
  const
     itmax=1000; eps=3.0e-7;
  var
     tem,qap,qam,qab,em,d,bz,bpp,bp,bm,az,app,am,aold,ap: double;
     m: integer;
begin
  am:=1; bm:=1; az:=1; qab:=a+b; qap:=a+1;
  qam := a-1; bz:=1-qab*x/qap;

  for m:= 1 to itmax do begin
    em:=m; tem:=em+em; d:= em*(b-m)*x/((qam+tem)*(a+tem));
    ap:=az+d*am; bp:=bz+d*bm;
    d:=-(a+em)*(qab+em)*x/((a+tem)*(qap+tem)); app:=ap+d*az;
    bpp:=bp+d*bz; aold:=az; am:=ap/bpp; bm:=bp/bpp;
    az := app/bpp; bz:=1;
    if ((abs(az-aold))<(eps*abs(az))) then break;
    if m=itmax then raise exception.Create('error in CaclDist.betacf');
```

```
  end;
  Betacf:=az;
end; {betacf}


function betai(a,b,x:double):double;
var Bt: double;
begin
  if (a<=0) or (b<=0) then
    exception.Create('error in CaclDist.betai');
  if (x<0) or (x>1) then
    exception.Create('error in CaclDist.betai');
  if (x=0) or (x=1) then
    bt :=0
  else
    bt := exp(lngamma(a+b)-lngamma(a)-lngamma(b) + a * ln(x) + b*ln(1-x));
  if ( x < ((a+1)/(a+b+2)) ) then
    betai := bt*betacf(a,b,x)/a
  else
    betai:=1-bt*betacf(b,a,1-x)/b
end;
{end of Numerical Recipes Code}
{-------------------------------------------------------------------------}

{GENERALIZED FUNCTIONS}

function icdf(dtype:integer;y,parm1,parm2,parm3:double):double;

begin
   Case Dtype of
       dGamma       : icdf:=parm1+parm3*icdfGamma(y,Parm2);
       dBeta        : icdf:=icdfbeta(y,parm1,Parm2,Parm3);
       dNormal      : icdf:=icdfNormal(y,parm1,Parm2);
       dLogNormal   : icdf:=icdfLogNormal(y,parm1,Parm2);
       dTriangular  : icdf:=icdfTriangular(y,parm1,Parm2,Parm3);
       dPoisson     : icdf:=icdfPoisson(y,parm1);
       dBinomial    : icdf:=icdfBinomial(y,parm1,round(Parm2));
       dUniform     : icdf:=icdfUniform(y,parm1,Parm2);
       dExponential : icdf:=icdfExponential(y,parm1);
       dGeometric   : icdf:=icdfGeometric(y,parm1);
       dWeibull     : icdf:=icdfWeibull(y,parm1,Parm2);
       dLogistic    : icdf:=icdfLogistic(y,parm1,Parm2);
       dCauchy      : icdf:=icdfCauchy(y,parm1,Parm2);
       dPareto      : icdf:=icdfPareto(y,parm1,Parm2);
       dHML         : icdf:=icdfHML(y,parm1,Parm2, Parm3);
       dLogUniform  : icdf:=icdfLoguniform(y,parm1,Parm2);
       dnone        : icdf:=1;
       end; {case}
end; {ICDF}
```

```
function cdf(dtype:integer;x,parm1,parm2,parm3:double):double;
begin
   Case Dtype of
       dGamma       : cdf:=cdfGamma(x/parm3-parm1,Parm2);
       dBeta        : cdf:=cdfbeta(x,parm1,Parm2,Parm3);
       dNormal      : cdf:=cdfNormal(x,parm1,Parm2);
       dLogNormal   : cdf:=cdfLogNormal(x,parm1,Parm2);
       dTriangular  : cdf:=cdfTriangular(x,parm1,Parm2,Parm3);
       dPoisson     : cdf:=cdfPoisson(x,parm1);
       dBinomial    : cdf:=cdfBinomial(x,parm1,round(Parm2));
       dUniform     : cdf:=cdfUniform(x,parm1,Parm2);
       dExponential : cdf:=cdfExponential(x,parm1);
       dGeometric   : cdf:=cdfGeometric(x,parm1);
       dWeibull     : cdf:=cdfWeibull(x,parm1,Parm2);
       dLogistic    : cdf:=cdfLogistic(x,parm1,Parm2);
       dCauchy      : cdf:=cdfCauchy(x,parm1,Parm2);
       dPareto      : cdf:=cdfPareto(x,parm1,Parm2);
       dHML         : cdf:=cdfHML(x,Parm1,Parm2,Parm3);
       dLogUniform  : cdf:=cdfLoguniform(x,parm1,Parm2);
       dnone        : cdf:=x;
       end; {case}
end; {cdf}

{-----------------------------------------------------------------------}

FUNCTION rbeta(aa,bb,scale:double): Double;

  {  Returns a single random deviate from the beta distribution with
     parameters A and B.  The density of the beta is
     x^(a-1) * (1-x)^(b-1) / B(a,b) for 0 < x < 1}

  { R. C. H. Cheng
    Generating Beta variatew with Nonintegral Shape Parameters
    Communications of the ACM, 21:317-322  (1978)
    Translated from fortran }

  { Preliminary Eyeballing of distributions of the output from this
    function seem to indicate that it's working fine.}

Const  expmax=89;
       infnty=1.0e38;

var  a,alpha,b,beta,gamma,r,s,t,u1,u2,v,w,z: double;

begin
  if ((aa=1) and (bb=1)) then begin
    rbeta:=random*scale;
    exit;
  end;
  if aa<bb then begin
```

```
    a:=aa;
    b:=bb;
  end
  else begin
    b:=aa;
    a:=bb;
  end;
  alpha := a + b;
  beta := sqrt((alpha-2.0) / (2.0*a*b-alpha));
  gamma := a + 1.0/beta;

  repeat
    u1 := random;
    u2 := random;
    v := beta*ln(u1/ (1.0-u1));
    if v>expmax then w:=infnty else w:=a*exp(v);
    z := (u1*u1)*u2;
    r := gamma*v - 1.3862944;
    s := a + r - w;
    t := ln(z)
  until (((r+alpha*ln(alpha/ (b+w)))>=t) or ((s+2.609438) >= (5.0*z)) or (s>t));

  if  aa=a then
    rbeta := scale*(w/(b+w))
  else
    rbeta := scale*(b/(b+w));
end;


function rNormal(Mean, StandardDeviation : double) : double;
var r, v1, v2, v3, dev: double;
begin
  repeat
    v1 := 2*Random-1;
    v2 := 2*Random-1;
    r := sqr(v1) + sqr(v2);
  until (r<1.0);
  if Random(2) = 1 then
    v3:=v2
  else
    v3:=v1;
  dev := v3 * SQRT(( - 2 * LN(r)) / r);
  rNormal := mean + dev * StandardDeviation;
end;


function rLogNormal(GM, GSD : double) : double;
begin
  rLogNormal:=exp(rNormal(ln(GM),ln(GSD)));
end;
```

```
(* Poisson routine taken from numerical recipes *)
(* gammln used in Poisson *)
function gammln(xx:double) : double;
const
   stp = 2.50662827465;
   half= 0.5;
   fpf = 5.5;
   cof: array[1..6] of double = (76.18009173,-86.50532033,24.01409822,
                                 -1.231739516,0.120858003e-2, -0.536382e-5);
var
   x,tmp,ser:double;
   j:integer;
begin
   x:= xx-1;
   tmp := x+fpf;
   tmp := (x+0.5)*ln(tmp)-tmp;
   ser := 1.0;
   for j := 1 to 6 do begin
     x:= x+1;
     ser := ser+cof[j]/x;
   end;
   gammln := tmp+ln(stp*ser);
end;


function rPoisson(Rate : double) : double;
var n,t:double;    (* times, ln of waiting time (1st part) *)
    limit,glsq,gllnrate,glg,y:double;

begin
   if (Rate<12) then begin
     n := -1;
     t := 1.0;
     limit := exp(-Rate);
     repeat
       n := n+1;
             t := t * random;
     until (t< limit);
   end else begin
     glsq := sqrt(2.0*Rate);
     gllnRate := ln(Rate);
     glg := Rate*gllnRate-gammln(Rate+1.0);
     repeat
       repeat
               y:= PI * Random;
         y:= sin(y)/cos(y);
         n:=glsq*y+Rate;
       until (n >= 0.0);
       n := trunc(n);
```

```
       t := 0.9*(1.0+sqr(y))*exp(n*gllnRate-gammln(n+1.0)-glg);
     until (t >= Random);
   end;
   rPoisson:=n;
end;

function rExponential(Rate : double) : double;
begin
  rExponential:= - ln(Random)/Rate;
end;


function rTriangular(Minimum, Maximum, MostLikely : double) : double;
var temp, base, part: double;
begin
  base := Maximum - Minimum;
  part := MostLikely - Minimum;
  Temp:=Random;
  if (Temp < (part/base)) then
     Temp := Minimum + Sqrt(base*part*Temp)
  else
     Temp := Maximum - Sqrt(base*(Maximum-Mostlikely)*(1-Temp));
  rTriangular := Temp;
end;

function rBinomial(Probability, Trials : double) : double;
{Probability must be in (0,1) for this function to work}

var mean,em,en,g,angle,oldg,p,pc,bnl: double;
    pclog,plog,sq,t,y:double;
    j: integer;
    n     : longint;
begin
   if (Probability<=0) or (Probability>=1) then
     raise exception.Create('error in CaclDist.rBinomial');
   n:=Trunc(Trials);
   if (Probability<=0.5) then p:=Probability else p:=1.0-Probability;
   mean := n*p;
   if (n<25) then begin
      bnl := 0.0;
      for j:=1 to n do begin
        if (Random<p) then bnl := bnl+1.0;
      end
   end else
   if (mean<1.0) then begin
     g := exp(-mean);
     t := 1.0;
     j := 0;
     repeat
        t:=t*Random;
```

```
        j:=j+1;
      until ((t<g) OR (j=n));
      bnl := j;
    end else begin
      en := n; oldg := gammln (en+1.0);
      pc := 1.0-p; plog := ln(p); pclog := ln(pc);
      sq := sqrt(2.0*mean*pc);
      repeat
        repeat
                angle := pi * Random;
          y := tan(angle);
          em := sq*y+mean;
        until ((em >=0.0) and (em<en+1.0));
        em := trunc(em);
        t := 1.2*sq*(1.0+sqr(y))   * exp(oldg-gammln(em+1.0) -
             gammln(en-em+1.0)+ em * plog+(en-em)*pclog);
      until (Random<=t);
      bnl := em;
    end;
    if(p<>Probability) then bnl := n-bnl;
    rBinomial := bnl;
end;


function rUniform(Minimum, Maximum : double) : double;
begin
  rUniform:=Minimum + (Maximum-Minimum)*Random;
end;

function rLogUniform(Minimum, Maximum : double) : double;
begin
  Result := icdfLogUniform(Random,Minimum,Maximum)
end;


function rGeometric(Probability : double) : double;
{Probability must be in (0,1) for this to work.}
begin
  if (probability<=0) or (probability >=1) then
    raise exception.Create('error in CaclDist.rGeometric');
  rGeometric:=trunc(ln(1-Random)/ln(1-Probability))+1;
end;


function rWeibull(scale,shape: double) : double;
{Inversion of CDF given in @Risk book
 Changed functional form 7/1/96
 removed scale=(1/scale)^(1/shape) - BMF}
begin
  rWeibull := scale * power( (ln(1/random)) , (1/shape) );
```

```
end;


(* simple inversion of cumulative distribution function *)
function rLogistic(Mean, Scale : double) : double;
begin
  rLogistic:=Mean-Scale*ln(1/Random - 1);
end;

(* compared to Erlang, this gets n-th time something happens  *)
(* distribution lambda^n * x^(n-1) * exp(-lambda*x)/Gamma(n)  *)
(* Gamma is just controlling routine, calling either          *)
(* gamma_calc1a (when a>=1) or gamma_calc2 (when 0<a<1).      *)
(* gamma_calc1 is an older routine that I found first;        *)
(* gamma_calc1a is apparently much faster.                    *)
(* gamma_calc1, gamma_calc1a are for parameters a>=1          *)
(* gamma_calc2 is for parameters 0<a<1                        *)
(* controlling routine must call right one                    *)
(* this function can be called only when alpha is >1
   there is no error checking, but the results may not be good
   Based on an algorithm presented in :
   Minh, Do Le.  ACM Trans. on Math. Software,14/3
   September 1988                                             *)

function gamma_calc1a(alpha:double):double;
label top,deliver;
var a,D,Dl,x,x1,x2,x4,x5,x_el,xp,xr,u,v,w:double;
    f1,f2,f4,f5,p1,p2,p3,p4:double;
    result1:double;
begin
   a := alpha -1.0;
   D := sqrt(a);
   if (alpha <= 2) then begin
      Dl := a/2;
      x1 := 0;
      x2 := Dl;
      x_el := -1.0;
      f1 := 0;
   end else begin
      Dl := D - 0.5;
      x2 := a- Dl;
      x1 := x2 - Dl;
      x_el := 1.0 - a/x1;
      f1 := exp(a*ln(x1/a) + 2*Dl);
   end;
   f2 := exp(a*ln(x2/a)+Dl);
   x4 := a+D;
   x5 := x4 + D;
   xr := 1.0 - a/x5;
   f4 := exp(a*ln(x4/a)-D);
```

```
   f5 := exp(a*ln(x5/a) - D - D);
   p1 := 2* f4 * D;
   p2 := 2 * f2 * Dl + p1;
   p3 := f5/xr + p2;
   p4 := -f1/x_el + p3;
top:                                          { 5 in Minh }
   repeat
      u := p4 * Random;
      if (u>p1) then begin
         if (u>p2) then begin
            w := Random;
            if (u>=p3) then begin             { 10 in Minh }
               u := (p4-u)/(p4-p3);
               x := x1 - ln(u)/x_el;
               if (x<0) then goto top
               else
               if (w<=(x_el*(x1-x)+1)/u) then begin
                  result1 := x;
                  goto deliver;
               end else w := w*f1*u;
            end else begin                                {9 in Minh }
               u := (p3-u)/(p3-p2);
               x := x5 - ln(u)/xr;
               if (w<=(xr*(x5-x)+1)/u) then begin
                  result1 := x;
                  goto deliver;
               end;
               w := w*f5*u;
            end;
         end else begin                                    { 7 in Minh }
            w := (u-p1)/Dl - f2;
            if (w<=0) then begin
               result1 := a-(u-p1)/f2;
               goto deliver;
            end;
            if (x<=f1) then begin
               result1 := x1+(w*Dl)/f1;
               goto deliver;
            end;
            v := Random;                                { 8 in Minh }
            x := x1 + v*Dl;
            xp := 2*x2-x;
            if (w>=f2+(f2-1)*(x-x2)/(x2-a)) then begin
               result1 := xp;
               goto deliver;
            end;
            if (w<=f2*(x-x1)/Dl) then begin
               result1 := x;
               goto deliver;
            end;
```

```
            if ((w>2*f2-1) AND (w>2*f2 - exp(a*ln(xp/a)+a-xp))) then begin
               result1 := xp;
               goto deliver;
            end;
         end;
      end else begin                                          { 5,6 in Minh }
         w :=  u/D - f4;
         if (w<=0) then begin
            result1 := a + u/f4;
            goto deliver;
         end;
         if (w<=f5) then begin
            result1 := x4 + (w*D)/f5;
            goto deliver;
         end;
         v := Random;
         x := x4 + v*D;
         xp := 2*x4-x;
         if (w>=f4+(f4-1)*(x-x4)/(x4-a)) then begin
            result1 := xp;
            goto deliver;
         end;
         if (w<=f4+(a/x4-1)*f4*(x-x4)) then begin
            result1 := x;
            goto deliver;
         end;
         if ((w>=2*f4-1) AND (w>=2*f4-exp(a*ln(xp/a)+a-xp))) then begin
            result1 := xp;
            goto deliver;
         end;
      end;
   until(ln(w)<=a*ln(x/a)+a-x);                               (* 11 in Minh *)
   result1 := x;
deliver:  gamma_calc1a := result1;
end;


{ the routine for when 0<a<1 }
function gamma_calc2(a:double) : double;
const e_hat_minus_one = 0.3678794;
var b,p,check,sgamma:double;
begin
  b := 1.0 + a * e_hat_minus_one;
  repeat
    p := b * Random;
    if (p<1.0) then begin
      sgamma := exp(ln(p)/a);
      check :=  sgamma;
    end else begin
      sgamma := - ln((b-p)/a);
```

```
      check := (1.0-a) * ln(sgamma);
    end;
  until (check <= rExponential(a));
  gamma_calc2 := sgamma;
end;


function rGamma(a,Lamda : double) : double;
begin
  if (a>=1) then rGamma := Lamda*gamma_calc1a(a)
  else rGamma := Lamda*gamma_calc2(a);
end;


function rCauchy(a,b:double): double;

{This function usually requires at least one parameter.  In the
function we received from NRAND1, the alpha (centering) parameter
is assumed as 0, beta as 1.

Therefore, the output is extrapolated to the correct parametrization by
running cdf of the output of this function with parameters of 0 and 1 and
then running icdf of that output with the parameter that is desired.  }

var t,u:double;
begin
  u:=random;
  if (u=0) then begin
     repeat; u:=random; until (u>0);
  end;  {range of borland random is 0<=random<1 }
  t:=(u-0.5)*pi;
  rCauchy:=cdfcauchy(cdfCauchy((sin(t)/cos(t)),0,1),a,b);
end;

function rPareto(x0,theta: double):double;
{function created by inserting a random number (0-1) into the
inverse of the cumulative density function. Feb 13, 1995 }
{theta's too low cause FP overflow}

begin
  if (x0<=0) or (theta<0.02) then
    raise exception.Create('error in CaclDist.rPareto');
  rpareto:= x0+x0*(power((1-random),(-1/theta))-1);
end;

function rHML(High, Medium, Low : double):double;
begin
  case Random(2) of
    0 : rHML:=High;
    1 : rHML:=Medium;
```

```
    2 : rHML:=Low;
  end;
end;

(*
function cdfCustom(X : double; Stats : TDescriptiveStatistics) : single;
var t : integer;
    prob : double;
begin
  t:=-1;
  if X=Stats.Quantiles[0] then
    prob:=0
  else
  if X=Stats.Quantiles[100] then
    prob:=1
  else begin
    repeat
      t:=t+1;
    until (X>=Stats.Quantiles[t]) and (X<=Stats.Quantiles[t+1]) and (t<100);
    if  (Stats.Quantiles[t+1]=Stats.Quantiles[t]) then
      prob:=t*0.05+0.025
    else

prob:=t*0.05+0.05*((X-Stats.Quantiles[t])/(Stats.Quantiles[t+1]-Stats.Quantiles[t]))
;
  end;
  cdfCustom:=prob;
end;


function icdfCustom(Prob : double; Stats : TDescriptiveStatistics) : double;
var t : longint;
    x : single;
begin
  t:=trunc(Prob*100) DIV 5;
  if t<100 then begin
    if (Stats.Quantiles[t+1]-Stats.Quantiles[t])>0 then

x:=Stats.Quantiles[t]+((Prob-t*0.05)/(0.05))*(Stats.Quantiles[t+1]-Stats.Quantiles[t
])
    else
      x:=Stats.Quantiles[t];
  end else
    x:=Stats.Quantiles[100];
  icdfCustom:=x;
end;
*)


function cdfNormal(y,mean,dev:double) : single;
```

```
const
        p  : extended =  0.231641900;
        b1 : extended =  0.319381530;
        b2 : extended = -0.356563782;
        b3 : extended =  1.781477937;
        b4 : extended = -1.821255978;
        b5 : extended =  1.330274429;
var
   sd,x,y1,y2,y3,y4,y5    : extended;
   t,z,r                  : extended;
begin
  sd := (y-mean)/dev;
{ check to see if the standard deviation ( sd ) is in range }
  {if sd < 0 then exit;}
  x := sd;
  r := EXP(-(x*x)/2)/2.5066282746;
{  r = frequency }
  z  := x;
  y1 := 1/(1+(p*ABS(x)));
  y2 := y1 * y1;
  y3 := y2 * y1;
  y4 := y3 * y1;
  y5 := y4 * y1;
  t  := 1 - r*(b1*y1+ b2*y2 + b3*y3 + b4*y4 + b5*y5);
  if z > 0 then
    cdfNormal := t
  else
    cdfNormal := 1 - t;
end;

function icdfNormal(prob,mean,dev:double) : double;
const
        c0 : extended = 2.515517;
        c1 : extended = 0.802853;
        c2 : extended = 0.010328;
        d1 : extended = 1.432788;
        d2 : extended = 0.189269;
        d3 : extended = 0.001308;
var
        dn,up,xp : EXTendED;
        t1,t2,t3 : EXTendED;
begin
  if prob=1 then icdfnormal:=mean+5*dev;
  if prob=0 then icdfnormal:=mean-5*dev;
  if (prob <= 0) OR (prob >= 1) then exit;
  if prob > 0.5 then
     xp := 1 - prob
  else
     xp := prob;
  t1 := SQRT(LN(1/SQR(xp)));
```

```
  t2 := t1 * t1;
  t3 := t2 * t1;
  up := c0  + c1*t1 + c2*t2;
  dn := 1 + d1*t1 + d2*t2 + d3*t3;
  xp := t1 - (up/dn);
  if prob <= 0.5 then
    icdfNormal := mean-xp*dev
  else
    icdfNormal := mean+xp*dev;
end;

function cdfLogNormal(x,GM,GSD: double): single;
var u,std: double;
begin
  if x=0 then cdflognormal:=1;
  if (GM<=1) or (GSD<=1) or (x<=0) then exit;
  u:=ln(GM); std:=ln(GSD);
  cdflognormal:=cdfnormal((ln(x)-u)/std,0,1);
end;

function icdfLogNormal(Prob,GM,GSD:double) : double;
var n,u,std: double;
begin
  if (GM<=1) or (GSD<=1) then exit;
  if Prob=0 then icdflognormal:=0;
  if Prob=1 then icdflognormal:=1e200;
  if (prob<=0) or (prob>=1) then exit;
  u:=ln(GM); std:=ln(GSD);
  n:=icdfnormal(prob,0,1);
  if u+std*n>1e4 then icdflognormal:=1e200 else
                      icdflognormal:=exp(u+std*n);
end;

function  icdfTriangular(y,A,B,ML:double): single;
var maba : double;
begin
  icdftriangular:=0;
  if b=a then exit;
  maba:=(ML-A)/(B-A);
  if (y<=maba) then
    icdftriangular:=a + sqrt(y * (ML-a) * (B-A))
  else
    icdftriangular:=b - sqrt((1-y) * (B-ML) * (B-A));
end;

function cdfTriangular(X,A,B,ML : double) : single;
begin
  cdftriangular:=0;
  if b=a then exit;
  if x<A then exit;
```

```
  if x>B then begin cdftriangular:=1; exit; end;
  if x<ML then cdfTriangular:=((ML-A)/(B-A)) * sqr((x-a)/(ml-a))
          else cdfTriangular:=1-((B-ML)/(B-A)) * sqr((b-x)/(b-ml));
end;


function cdfUniform(X,A,B : double) : single;
begin
   cdfUniform := (x-a)/(b-a);
end;
function icdfUniform(Prob,A,B : double) : double;
begin
   icdfUniform:=Prob*(b-a)+a;
end;

function cdfLogUniform(x,A,B :double):single;
begin
  if x<a then Result := 0 else
  if x>=b then Result := 1 else
  Result := ln(x/a)/ln(b/a);
end;
function icdfLogUniform(prob,A,B:double):double;
begin
  Result := exp(ln(b/a)*prob)*a;
end;

function cdfExponential(x,lamda : double): single;
begin
   cdfExponential:=1.0-1/exp(lamda*x);
end;
function icdfExponential(Prob,lamda : double): double;
begin
   icdfExponential:=ln(1/(1-Prob))/Lamda;
end;

function cdfGamma(X,A : double) : single;
begin
  cdfGamma:=igammap(A,X);
end;

function icdfgamma(p,a: double):double;
{RETURNS NEGATIVE ONE (-1.0) if FUNCTION NON-EVALUABLE}

{From fortran code written by A.H. Morris Jr}

Label 30,1,2,3,110,120,130,140,150,170,180,190,200,210,4,230,240,250,260;

CONST NUM_ITERATIONS=50;

var  x,q,x0,a0,a1,a2,a3,am1,amax,ap1,ap2,ap3,apn,b,b1,b2,b3,
```

```
     b4,c,c1,c2,c3,c4,c5,d,e,e2,eps,g,h,ln10,pn,qg,qn,
     r,rta,s,s2,sum,t,tol,u,w,xmax,xmin,xn,y,z: double;
     ierr,iop: integer;
     amin,bmin,dmin,emin,eps0:  array [0..2] of double;

begin
  ln10:=2.302585;
  c:=0.577215664901533 ;
  a0:=3.31125922108741 ;a1:=11.6616720288968 ;
  a2:=4.28342155967104 ;a3:=0.213623493715853 ;
  b1:=6.61053765625462 ;b2:=6.40691597760039 ;
  b3:=1.27364489782223 ;b4:=0.036117081018842 ;
  eps0[1]:=1E-10; eps0[2]:=1E-08;
  amin[1]:=500.0 ; amin[2]:=100.0 ;
  bmin[1]:=1E-28; bmin[2]:=1E-13;
  dmin[1]:=1E-06; dmin[2]:=1E-04;
  emin[1]:=2E-03; emin[2]:=6E-03;
  tol:=1E-5;
  ierr:=0;
  x0:=0;
{  E, xmin, and xmax are machine dependent constants.
    e is the smallest number for which 1.0 + e > 1.0.
    xmin is the smallest positive number and xmax is the
    largest positive number.}

  e := 1e-21;
  xmin := 1e-150;
  xmax := 9.9e150;

  x := 0.0;
  if (a<0.001) then begin icdfgamma:=-1; exit; end;
  q:=1-p;
  if (p=0.0) then begin icdfgamma:=x; exit; end;
  if (q=0.0) then begin icdfgamma := xmax; exit; end;
  if (a=1.0) then
    if (q>=0.9) then begin
      icdfgamma :=  -ln(1-p); exit;
    end else begin
      icdfgamma := -ln(q); exit;
    end;

  e2 := 2.0*e;
  amax := 0.4E-10/(sqr(e));
  iop := 1;
  if (e>1E-10)  then iop := 2;
  eps := eps0[iop];
  xn := x0;

{A<1}
```

```
  if (a>1.0) then GOTO 2;
  g := calcgamma(a+1.0);
  qg := q*g;
  if (qg=0.0) then begin
    icdfgamma:=xmax;
    exit;
  end;
  b := qg/a;
  if (qg>0.6*a) then GOTO 3;
  if not ((a>=0.30) OR (b<0.35)) then begin
    t := exp(-(b+c));
    u := t*exp(t);
    xn := t*exp(u);
    GOTO 1;
        end;
  if (b>=0.45) then GOTO 3;
  if (b=0.0) then begin
    icdfgamma:=xmax;
    exit;
  end;
  y := -ln(b);
  s := 0.5 + (0.5-a);
  z := ln(y);
  t := y - s*z;
  if (b>=0.15) then begin
    xn := y - s*ln(t) - ln(1+s/(t+1));
    GOTO 4;
  end;

  if (b>0.01) then begin
    u := ((t+2*(3-a))*t+(2-a)* (3-a))/
          ((t+(5-a))*t+2);
    xn := y - s*ln(t) - ln(u);
    GOTO 4;
  end;

30:
  c1 := -s*z;
  c2 := -s* (1+c1);
  c3 := s* ((0.5*c1+ (2.0-a))*c1+ (2.5-1.5*a));
  c4 := -s*(((c1/3+ (2.5-1.5*a))*c1+ ((a-6)*a+7))*
            c1+ ((11*a-46)*a+47)/6);
  c5 := -s* ((((-c1/4+ (11*a-17)/6)*c1+ ((-3.0*a+
       13.0)*a-13.0))*c1+0.5* (((2.0*a-25.0)*a+72.0)*a-
        61.0))*c1+ (((25.0*a-195.0)*a+477.0)*a-379.0)/
        12.0);
  xn := ((((c5/y+c4)/y+c3)/y+c2)/y+c1)+ y;
  if (a>1.0) or (b>bmin[iop]) then GOTO 4;
  icdfgamma := xn;
  EXIT;
```

```
3:    if (p>=0.9) and (a<0.01) then begin icdfgamma:=-1; exit; end;
      if (b*q<=1E-8) then xn := exp(-(q/a+c))
                           else if (p<=0.9) then xn := exp(ln(p*g)/a)
                                                 else xn :=
exp((ln(1-q)+gammln(a))/a);

      if (xn=0) then begin icdfgamma:=-1; exit; end;
      t := 0.5 + (0.5-xn/(a+1));
      xn := xn/t;
      if xn<0 then xn:=e;
      GOTO 1;

{   Selection of the initial approximation xn of x if a>1 }


2: if (q<=0.5) then w := ln(q) else w := ln(p);
   t := sqrt(-2*w);
   s := t - (((a3*t+a2)*t+a1)*t+a0)/((((b4*t+b3)*t+b2)*t+b1)*t+1);
   if (q>0.5) then s := -s;
   rta := sqrt(a);
   s2 := sqr(s);
   xn := a + s*rta + (s2-1)/3 + s* (s2-7)/ (36*rta) -
         ((3*s2+7)*s2-16)/ (810*a) +
         s* ((9*s2+256)*s2-433)/ (38880*a*rta);
   if xn<0 then xn:=0;
   if a<amin[iop] then GOTO 110;
   x := xn;
   d := 0.5 + (0.5-x/a);
   if (abs(d)<=dmin[iop]) then begin
     icdfgamma:=x; EXIT;
   end;

110:
   if (p<=0.5) then GOTO 130;
   if (xn<3.0*a) then GOTO 4;
   y := - (w+gammln(a));
   d := a* (a-1.0);
   if d<2 then d:=2;
   if (y<ln10*d) then GOTO 120;
   s := 1 - a;
   z := ln(y);
   GOTO 30;

120:
   t := a - 1.0;
   xn := y + t*ln(xn) - ln(1+ (-t/(xn+1)));
   xn := y + t*ln(xn) - ln(1+ (-t/(xn+1)));
   GOTO 4;
```

```
130:
   ap1 := a + 1;
   if (xn>0.70*ap1) then GOTO 170;
   w := w + gammln(ap1);
   if (xn>0.15*ap1) then GOTO 140;
   ap2 := a + 2;
   ap3 := a + 3;
   x := exp((w+x)/a);
   x := exp((w+x-ln(1+ (x/ap1)* (1+x/ap2)))/a);
   x := exp((w+x-ln(1+ (x/ap1)* (1+x/ap2)))/a);
   x := exp((w+x-ln(1+ (x/ap1)* (1+ (x/ap2)* (1+ x/ap3))))/a);
   xn := x;
   if (xn>1E-2*ap1) then GOTO 140;
   if (xn<=emin[iop]*ap1) then begin
     icdfgamma:=x; EXIT;
   end;
   GOTO 170;

140:
   apn := ap1;
   t := xn/apn;
   sum := 1 + t;

150:
   apn := apn + 1;
   t := t* (xn/apn);
   sum := sum + t;
   if (t>1E-4) then GOTO 150;
   t := w - ln(sum);
   xn := exp((xn+t)/a);
   xn := xn* (1- (a*ln(xn)-xn-t)/ (a-xn));
   GOTO 170;

{  SCHRODER ITERATION USING P   }

1: if (p>0.5) then GOTO 4;
170:
   if (p<=1E10*xmin) then begin
     ICDFGAMMA:=Xn;
     EXIT;
   end;
   am1 := (a-0.5) - 0.5;
180:
   if (a<=amax) then GOTO 190;
     d := 0.5 + (0.5-xn/a);
    if (abs(d)<=e2) then begin
      ICDFGAMMA:=Xn; EXIT;
    end;

190:
```

```
   if (ierr>=NUM_ITERATIONS) then begin
     icdfgamma:=x;
     exit;
   end;
   ierr := ierr + 1;
   qn:=gammq(a,xn);
   pn:=gammp(a,xn);
   if (pn=0.0) or (qn=0.0) then begin
     ICDFGAMMA:=xn;
     EXIT;
   end;
   r := exp(-xn)*power(xn,a)/calcgamma(a);
   if (r=0) then begin
     ICDFGAMMA:=xn;
     EXIT;
   end;
   t := (pn-p)/r;
   w := 0.5* (am1-xn);
   if (abs(t)<=0.1 ) and ( abs(w*t)<=0.1) then GOTO 200;
   x := xn* (1.0-t);
   if (x<=0.0) then begin
     ICDFGAMMA:=-1;
     EXIT;
   end;
   d := abs(t);
   GOTO 210;

200:
  h := t* (1.0+w*t);
  x := xn* (1.0-h);
  if (x<=0.0) then begin
    ICDFGAMMA:=-1;
    EXIT;
  end;
  if (abs(w)>=1.0 ) and ( abs(w)*t*t<=eps) then begin
    icdfgamma:=x;
    EXIT;
  end;
  d := abs(h);

210:
  xn := x;
  if (d>tol) then GOTO 180;
  if (d<=eps) or (abs(p-pn)<=tol*p) then begin
    icdfgamma:=x;
    EXIT;
  end;
  GOTO 180;

  {  SCHRODER ITERATION USING Q  }
```

```
4:
  if (q<=1E10*xmin) then begin
    ICDFGAMMA:=xn;
    EXIT;
  end;
  am1 := (a-0.5) - 0.5;
230:
  if (a<=amax) then GOTO 240;
  d := 0.5 + (0.5-xn/a);
  if (abs(d)<=e2) then begin
    ICDFGAMMA:=xn;
    EXIT;
  end;

240:
  if (ierr>=NUM_ITERATIONS) then begin
    ICDFGAMMA:=x;
    EXIT;
  end;
  ierr := ierr + 1;
  qn:=gammq(a,xn);
  pn:=gammp(a,xn);
  if (pn=0.0) OR  (qn=0.0) then begin
    ICDFGAMMA:=xn;
    EXIT;
  end;
  r := exp(-xn)*power(xn,a)/calcgamma(a);
  if (r=0.0) then begin
    ICDFGAMMA:=xn;
    EXIT;
  end;
  t := (q-qn)/r;
  w := 0.5* (am1-xn);
  if (abs(t)<=0.1 ) and ( abs(w*t)<=0.1) then GOTO 250;
  x := xn* (1.0-t);
  if (x<=0.0) then begin
    ICDFGAMMA:=-1;
    EXIT;
  end;
  d := abs(t);
  GOTO 260;

250:
  h := t* (1.0+w*t);
  x := xn* (1.0-h);
  if (x<=0.0) then begin
    ICDFGAMMA:=-1;
    EXIT;
  end;
```

```
  if (abs(w)>=1.0) AND (abs(w)*sqr(t)<=eps) then begin
    icdfgamma:=x;
    EXIT;
  end;
  d := abs(h);

260:
  xn := x;
  if (d>tol) then GOTO 230;
  if (d<=eps) then begin
    icdfgamma:=x;
    EXIT;
  end;
  if (abs(q-qn)<=tol*q) then begin
    icdfgamma:=x;
    EXIT;
  end;
  GOTO 230;
end;  {THIS ICDFGAMMA NIGHTMARE}


function cdfWeibull(XD,Scale,Shape : double) : single;
begin
  cdfWeibull:=1 - exp ( -1* power((XD/scale),shape));
end;

function icdfweibull(y,scale,shape: double): double;
begin
  icdfWeibull:=scale*power(-LN(1-y),(1/shape));
end;


function cdfLogistic(x,a,b:double): single;
begin
   cdfLogistic:=1/(1+exp(-(x-a)/b));
end;
function icdfLogistic(Prob,a,b:double): double;
begin
   icdfLogistic:=-b*ln((1/Prob)-1) + a;
end;

function icdfPoisson(y,lambda:double) : double;

const max_iterations = 25;
{ Iterative guess technique }
{ This acuracy requires a maximum of 25 iterations under normal circs. }

var err,yofx: double;
    xtop,xbot,xguess: integer;
```

```
begin
   Xtop:=round(lambda*2);  if lambda<70 then xtop:=round(lambda*3); if lambda<10
then xtop:=30;
   xbot:=0;
   xguess:=round((xtop-xbot)/2);
   repeat
     yofx:=gammq(xguess+1,lambda);
     if yofx < y then begin
       xbot:=xguess;
       xguess:=xguess+round((xtop-xguess)/2);
     end;
     if yofx > y then begin
       xtop:=xguess;
       xguess:=xguess-round((xguess-xbot)/2);
     end;
   until (xtop-xbot<=1) or (yofx=y);
   err:=abs(yofx-y);
   if abs(gammq(xguess,lambda)-y)<err then
     xguess:=xguess-1
   else if abs(gammq(xguess+2,lambda)-y)<err then
     xguess:=xguess+1;
   icdfpoisson:= xguess;
end;


function cdfPoisson(xd,lambda:double):single;

begin
  cdfPoisson:=GammQ(xd+1,lambda);   {numerical recipes}
end;


function cdfBinomial(xd,p:double;n:integer): single;
{XD = argument}
{p = Probability}
{n=  trials}

{See numerical recipes, p169. Faster, and it works. }

begin
  if xd>=n then cdfBinomial:=1
  else if xd<0 then cdfBinomial:=0
  else cdfBinomial := 1-betai(trunc(xd)+1,n-trunc(xd),p);
end;

function icdfBinomial(y,p : double; n: integer): double;

const max_iterations = 25;
{ Iterative guess technique }
{ This acuracy requires a maximum of 25 iterations under normal circs. }
```

```
var err,yofx: double;
    xguess,xtop,xbot: integer;

begin
  if (p<0) or (p>1) or (n<0) then begin
    icdfbinomial:=-1;
    exit;
  end;

  Xtop:=n;
  xbot:=0;
  xguess:=round((xtop-xbot)/2);
  repeat
    yofx:=cdfbinomial(xguess,p,n);
    if yofx < y then begin
      xbot:=xguess;
      xguess:=xguess+round((xtop-xguess)/2);
    end;
    if yofx > y then begin
      xtop:=xguess;
      xguess:=xguess-round((xguess-xbot)/2);
    end;
  until (xtop-xbot<=1) or (yofx=y);
  err:=abs(yofx-y);
  if abs(cdfbinomial(xguess,p,n)-y)<err then
    xguess:=xguess-1
  else if abs(cdfbinomial(xguess+2,p,n)-y)<err then
    xguess:=xguess+1;
  icdfbinomial:= xguess;
end;


function cdfgeometric(xd,p:double):single;

begin
  if (p<0) or (p>1) then begin
    cdfgeometric:=-1;
    exit;
  end;
  if xd>1e100 then
    cdfgeometric:=1
  else
    cdfgeometric:=1-power((1-p),(Trunc(xd)))
end;

function icdfGeometric(y,p : double): double;

begin
  if (p>1) or (p<0) then icdfgeometric:=-1 else
```

```
    if y=1 then icdfgeometric:=1e150 else
      if p=1 then icdfgeometric:=0 else
         icdfGeometric:=trunc(ln(1-y)/ln(1-p));
end;

function cdfBeta(x,a,b,scale:double):single;
begin
  if x>scale then x:=scale;
  cdfbeta:=betai(a,b,(x/scale));
end;

function icdfBeta(y,a,b,scale:double):double;
const acurate_to = 1e-8;
      max_iterations = 25;
{ Iterative guess technique }
{ This acuracy requires a maximum of 25 iterations under normal circs. }

var xguess,xtop,xbot,yofx: double;
    iterations: integer;

begin
  iterations:=0;
  xtop:=1;
  xbot:=0;
  xguess:=0.5;
  repeat
    iterations:=iterations+1;
    yofx:=betai(a,b,xguess);
    if yofx < y then begin
      xbot:=xguess;
      xguess:=xguess+(xtop-xguess)/2;
    end;
    if yofx > y then begin
      xtop:=xguess;
      xguess:=xguess-(xguess-xbot)/2
    end;
  until (abs(yofx-y)<acurate_to) or (iterations>max_iterations);
  icdfbeta:= xguess*scale;
end;


function cdfCauchy(x,a,b:double):single;
begin
  cdfCauchy:=0.5+(1/pi)*arctan((x-a)/b);
end;
function icdfCauchy(prob,a,b:double):double;

begin
  icdfCauchy:=b*(tan(pi*(prob-0.5)))+a
end;
```

```
function cdfpareto(x,x0,theta: double): single;
begin
   cdfpareto:=1-power(1+((x-x0)/x0),-theta);
end;

function icdfPareto(y,x0,theta:double):double;
begin
     icdfPareto:=x0+x0*(power(1-y,-1/theta)-1);
end;


function cdfHML(X,High, Medium, Low : double) : single;
begin
   if x>=High then   cdfHML:=1 else
   if x>=Medium then cdfHML:=0.6666666 else
   if x>=Low then    cdfHML:=0.3333333 else
                     cdfHML:=0;
end;
function icdfHML(Prob,High,Medium,Low : double) : double;
begin
   if Prob>0.6666666 then icdfHML:=High else
   if Prob>0.3333333 then icdfHML:=Medium else
                          icdfHML:=Low;
end;

begin
end.
```