Computing Confidence Intervals for Floods I

Robert Whitley (1) and T.V. Hromadka II (2)

(1) Professor of Mathematics, Department of Mathematics, University of California at Irvine, Irvine, CA 92717, USA
(2) Research Associate, Department of Civil Engineering, Princeton University, Princeton, N.J., 08544, USA

I. ABSTRACT

The estimation of the size of the T-year flood is a basic problem in hydrology. One important source of uncertainty in this estimate is that caused by the uncertain estimation of the parameters in the log Pearson III distribution which describes the occurance of the maximum annual discharge. We describe here a program which, by means of a simulation, allows the computation of confidence intervals for the magnitude of the T-year flood as well as determining the distribution of the estimates for this flood.

II. KEY WORDS

T-year Flood, Confidence Interval, Log Pearson III Distribution, Simulation

III. INTRODUCTION

It has been known for a long time, and is given in bulletins 17A and 17B of the Water Resources Council [1,2], that in the case of a flood distribution whose logarithm is normally distributed, confidence intervals for the T-year flood can be obtained by the use of the non-central student's t-distribution.

The more general case, following the guidelines of bulletins 17A and 17B, is when the logarithm of the flood distribution has a Pearson III distribution with a non-zero skew parameter. This case of non-zero skew is more complicated than the case of a lognormally distributed flood which is the case of zero skew [3,4,5,6,7,8,9]. In an important paper [9] Stedinger shows that the confidence intervals, for quantiles, which are given in the U.S. Water Resources Council guidelines [1,2] are not satisfactory. He uses a formula due to Kite [7] and derives an expression for confidence intervals for the quantiles which he shows is satisfactory in several simulations.

In [6] Hardison simulated a random variable which we will call the "non-central gamma distribution"; note that in statistics this name is also used for a different random variable. He uses this simultation to assess the accuracy of an approximate formula for confidence limits for flood-frequency curves given in Bulletin 17A. (This is one of the formulas which Stedinger in [9] found to be unsatisfactory.) Hardison's simulation involved T values of 10, 50, 100, and 500 years, sample sizes 10,20 and 40, skew coefficients of -1,0 and 1, levels of significance .01,.05,.10,.90,.95, and .99, and were each determined by means of 1000 point simulations.

To obtain more accuracy and to cover a more extensive range of T, sample sizes, skews, and levels of significance than [6], a program was developed to simulate a set of values of the non-central gamma random variable. For the purpose of comparing choices of probabilities and the size of the associated one-sided confidence intervals it is useful to have a table of the percentiles of the non-central gamma distribution; our simulation gives percentiles for the range 5%(5%)95%. For applications of this see [10]. From this table an empiricial distribution function for the value Q_T of the T-year flood is easily found and this distribution is essential for other hydrological calculations; for an example see [11].

IV. MATHEMATICAL DISCUSSION

A. Zero skew. The case of zero skew is a model for the case of non-zero skew. So we first consider the case where X, the logarithm of the maximum annual discharge, has a normal distribution. For the T-year flood, take p=1-1/T, and let y_p be the p-th quantile of X; i.e.

$$P(X \le y_p) = p \qquad (1)$$

It is y_p that we want to estimate. If the mean μ and standard deviation σ of X were known, then since $(X-\mu)/\sigma$ has a N(0,1) distribution, i.e. a normal distribution with mean 0 and standard deviation 1,

$$(y_p-\mu)/\sigma = z_p \qquad (2)$$

where z_p is the p-th quantile for an N(0,1) distribution. Of course μ and σ are not actually known; we only have estimates for them, $\hat{\mu}$ and $\hat{\sigma}$, based on m data points. It is natural to use the estimate

$$\hat{y}_p = \hat{\mu} + \hat{\sigma} z_p, \qquad (3)$$

but it is important to note that the estimator $\hat{y}_p$ is now itself a random variable depending on $\hat{\mu}$ and $\hat{\sigma}$.

0266-9463/86/030138 - 13 $2.00
© COMPUTATIONAL MECHANICS PUBLICATIONS 1986

Consider

$$(y_p-\hat{\mu})/\hat{\sigma}=((\mu-\hat{\mu})/\sigma)+z_p/(\hat{\sigma}/\sigma). \qquad (4)$$

Equation (4) can be written , see [12], as

$$(1/\sqrt{m})((Z+z_p\sqrt{m})/\sqrt{W}). \qquad (5)$$

The random variable in brackets in (5) has a non-central t-distribution, with non-centrality parameter $\delta=z_p\sqrt{m}$; the special case $\delta=0$ is the student's t-distribution.

Since the distribution of (4) can be written in terms of the known non-central t-distribution, confidence intervals for y_p can be given. Because of the way in which the flood design value is used, the confidence interval of greatest interest is a one-sided interval which gives an upper bound, but of course other confidence intervals can also be found by using the appropriate percentiles of the non-central t-distribution. And, as was mention above, the empirical distribution function is also needed for other problems. *The choice of the value T, for the T-year flood, and the number m of data points determine the non-centrality parameter δ.* The other quantity which must be specified is a probability p' for the one-sided confidence interval. If $t_{p'}$ is the p'-th quantile of the non-central t-distribution, then

$$P(y_p\le\hat{\mu}+\hat{\sigma}(t_{p'}/\sqrt{m}))=p' \qquad (6)$$

and we have found the one-sided 100p' percent confidence interval for y_p. The choice of the set of *values p'= .05(.05).95 gives an empirical distribution* function.

B. Non-zero skew. In this case the logarithm X of the yearly peak discharge is assumed to have a Pearson type III distribution with density function

$$f(x)=(1/|a|\Gamma(b))((x-c)/a)^{b-1}\exp-((x-c)/a) \qquad (7)$$

where, in the case of positive a, the density is given by (7) for x>c and is zero for x<c, while in the case of negative a the density is given by (7) for x<c and is zero for x>c. Computing the mean μ, standard deviation σ, and skew γ from (7) shows that (see, for example [13])

$$\begin{aligned}\sigma^2&=a^2b\\ \gamma^2&=4/b,\\ \mu&=c+ab\end{aligned} \qquad (8)$$

where a has the same sign as γ. In following the guidelines of bulletins 17A and 17B [1,2], the skew coefficient γ is estimated either from a map of regional skews or from a large pool of data from that region. *Consequently the error in estimating γ is of* an entirely different kind than that which arises in estimating μ and σ by means of m data points for the specific area for which the T-year flood is being estimated. What is usually done to simplify this complicated situation, and what was done for the simulation, is to suppose that γ is given exactly. This focuses attention on that part of the variability in the estimate of the T-year flood which arises from the uncertainty in the estimation of μ and σ, and ignores the variability which comes from the uncertainty in the *estimate of γ.*

The form of the density (7) shows that the random variable

$$Z=(X-c)/a \qquad (9)$$

has the one parameter density

$$g(x)=(1/\Gamma(b))x^{b-1}e^{-x} \qquad (10)$$

for x>0 and 0 for x<0; i.e. Z has a gamma distribution with shape parameter b and scale parameter 1.

In [10] it is shown that if t_q is the 100q-th percentile for the gamma distribution Z, $\hat{\theta}_Z$ and $\hat{\theta}_Z^2$ are the usual estimators for the mean of Z and the variance of Z using m sample points, then

$$\begin{aligned}(y_p-\hat{\mu})/\hat{\sigma}&=(t_p-\hat{\mu}_Z)/\hat{\theta}_Z \text{ for } a>0\\ (y_p-\hat{\mu})/\hat{\sigma}&=(\hat{\mu}_Z-t_{1-p})/\hat{\theta}_Z \text{ for } a<0.\end{aligned} \qquad (11)$$

So confidence intervals for y_p can be obtained if, for Z having the gamma distribution (10), we can find the distribution of

$$(\delta-\hat{\mu}_Z)/\hat{\theta}_Z. \qquad (12)$$

It is this distribution which we have called the noncentral gamma distribution [10].

v. PROGRAMMING DISCUSSION

In order to simulate the distribution of the random variable of equations (11) we first must find the percentile t_p or t_{1-p} of the gamma distribution Z. In our program this is obtained by first finding the corresponding percentile of the normal N(0,1) distribution, and to do this Hasting's rational approximation [14, page 932] for the normal distribution function F is used to find F(z) and interval halving is used to find the value of z_p with $F(z_p)=p$ or the z_{1-p} with $F(z_{1-p})=1-p$: see the function *"NormalF" and the function "InverseF" in the program* listing. Then the Wilson-Hilferty transformation [15,16] is applied to this percentile for the normal to get the percentile for the gamma distribution: see the function "Wilson-Hilferty" in the program.

Next we must generate a sequence of random values whose distribution is that of the given gamma distribution Z. To do this, we used the ingenious Box-Muller method [17,18] for generating normally distributed random variables and then apply the Wilson-Hilferty transformation to these variates to get the desired *sequence: see the procedure "RandomNormal" in the* program.

Now for a given m, m independent gamma distributed random variables can be generated, the sample mean and standard deviation calculated, and, together with the gamma percentile which we have discussed calculating above, one value of the random variable given by equations (11) can be calculated. This is done in procedure "mands" and function "statistic". The purpose of the function "statistic" is to separate out the three cases of skew equal zero, *the case of a normal distribution, skew greater than* zero, the case of the first of the two equations (11), and skew less than zero, the case of the second of the two equations (11).

In order to use a sequence of numbers calculated as discussed above to get an approximate probability distribution for the random variable we are simulating, a histogram must be constructed for this sequence. To this end, a small sequence of size 100 is calculated in order to be able to scale the numbers so that they will almost all lie in (-2,2) and *also so that they will be spread out enough in (-2,2)* to obtain good accuracy for the distribution *function:see procedure "scale". Once this scaling is*

found, the interval [-2,2] is divided up with a step size of .01 and in the procedure "tally" a count is kept of the number of time a random variable value falls in this subinterval of length .01.

The procedure "percentiles" uses the results of "tally" and linear interpolation to give the percentiles .05(.05).95.

The user inputs the number of m, of data points to be used in the sample mean and sample standard deviation, the skew coefficient, and the value T for the T-year flood. The number of blocks to be used in the simulation, each of size 1000, can also be choosen by the user; for example, for m = 30 data points and a choice of 10 blocks, the program will 10,000 times sample 30 points thereby sending 10,000 values of the statistic, given by the function "statistic, to the procedure "tally" and in the process of doing this it will generate 300,000 gamma distributed random variables.

For output, the user first sees the percentiles for the distribution; see the procedure "percentiles" and the sample output below.

Concerning the accuracy of a simulation, we quote [18, page 81]:"Apparently, however, the testing of generators of random vectors would best be carried out by constructing a simulation problem to which the solution is known, and which requires essentially the same characteristics as the problem which is ultimately to be solved". This advice is implemented in the procedure "percentiles". Although the main purpose of the procedure "tally" is to tally up the histogram for the statistic of interest, it also keeps a tally for the the sample mean; the distribution of the sample mean of a gamma distribution is known exactly and as one check on the accuracy of the simulation the difference between the sample mean percentile and the simulated sample mean percentile is printed out along with the percentiles of the statistic. Another check on the simulation accuracy is obtained by running the case of zero skew. The resulting distribution is the noncentral t-distribution and the simulation percentiles can be checked against published tables [19] or by generating the noncentral t-distribution itself for which see, for example, [20].

The user can also choose to see the entire histogram from which the percentiles have been interpolated: see the procedure "histogram"; and the user can choose to input estimates for the mean and standard deviation at a specific site so as to see the transformed simulation percentiles which give the percentiles in terms of the discharge values:see the procedure "floodpercentiles".

The program "confidence", described above and listed below, was written in Turbo-Pascal, with 8087 support, on the IBM PC. With the Intel 8087 numerical data processor chip, a simulation like the example given below, which involves the generation of several hundred thousand gamma distributed random numbers, takes roughly a half-hour. The program would be an order of magnitude slower without the 8087.

VI. REFERENCES

1. Guidelines for Determining Flood-Flow Frequency, Water Resources Council Hydrology Committee, Bulletin #17A, Washington, D.C., 1977.
2. Guidelines for Determing Flood-Flow Frequency, Water Resources Council Hydrology Committee, Bulletin #17B, Washington, D.C., 1981.
3. Bobee, B., Comment on "The log Pearson type 3 distribution: The T-year event and its asymptotic standard error" by R. Condie, Water Res. Research 15(1979)189-190.
4. Bobee, B., Sample error of T-year events computed by fitting a Pearson type 3 distribution, Water Res. Research 9(1973)1264-1270.
5. Condie, R., The log Pearson type 3 distribution: The T-year event and its asymptotic standard error by maximum likelihood theory, Water Res. Research 13(1977)987-991.
6. Hardison, C., Confidence limits for flood frequency curves computed from samples from *Pearson type III populations*, J. of Research U.S.G.S. 4(1976)545-547.
7. Kite, G.W.,Confidence limits for design events, Water Res. Research 11(1975)48-53.
8. Kite, G.W.,Frequency and Risk Analysis in Hydrology,Water Resources Pub.,Fort Collins, Colorado, 1977.
9. Stedinger, J. R., Confidence intervals for design events, J. of Hydraulic Engineering 109(1983)13-27.
10. Whitley,R. and Hromadka II, T., Estimating 100 year flood confidence intervals, submitted to Advances in Water Research, 1986.
11. Hromadka II, T. and Whitley, R., Calibrating a design storm method, submitted to Advances in Water Research, 1986.
12. Breiman, L., Statistics: With a View Towards Applications, Houghton-Mifflin, Boston, 1973.
13. Hall, M. J., Urban Hydrology, Elsevier, London, 1984.
14. Abramowitz, M. and Stegun, I., Handbook of Mathematical Functions, National Bureau of Standards, Washington,D.C., 1965.
15. Wilson, E.B. and Hilferty, M. M., The distribution of chi-square, *Proc. Nat. Acad. Sci., U.S.A. 17(1931)684-8.*
16. Mathur, R.K., A note on the Wilson-Hilferty transformation of chi-squared, Bull.Calcutta Stat. *Assoc. 10(1961)103-105.*
17. Box, G. and Muller, M., A note on the generation of random normal deviates, Ann. Math. Stat. 29 (1958) 610-611.
18. Newman, T. and Odell, P, The Generation of Random Variates, Griffin, London, 1971.
19. Resnikoff, G.J. and Lieberman, G.J., Tables of the Non-Central t-Distribution, Standford Univ. Press, Calif., 1957.
20. Whitley, R. and Hromadka II, T., Computing confidence limits for floods II, Microsoftware for Engineers.

VII. SAMPLE OUTPUT

PERCENTILES

(linearly interpolated from histogram)
m=sample mean, s sample s.d.
with 30 points in the sample
from a gamma random variable with
gamma = 0.2500
b = 64.0000
Kp = 2.50907
tp = 84.07257
This is the Kp and tp for the 100.0 year flood.

Y=[(tp-m)/s]

error in mean = m - true sample mean

Number of points in the simulation:
10 blocks of size 1000

Percentile	Y	Y-Kp	error in mean
5.00	1.950	-0.559	-0.004
10.00	2.071	-0.438	-0.010
15.00	2.154	-0.355	0.012
20.00	2.225	-0.284	0.013
25.00	2.285	-0.224	0.025
30.00	2.338	-0.171	0.011
35.00	2.390	-0.119	0.012
40.00	2.440	-0.069	0.017
45.00	2.490	-0.019	0.007
50.00	2.541	0.032	0.017
55.00	2.599	0.090	0.016
60.00	2.650	0.141	0.008
65.00	2.706	0.197	0.009
70.00	2.774	0.265	0.008
75.00	2.844	0.335	0.014
80.00	2.923	0.414	0.019
85.00	3.016	0.507	0.011
90.00	3.147	0.638	-0.002
95.00	3.349	0.840	-0.032

FLOOD PERCENTILES

10 to the power m+s*Yp
Yp the percentile of Y
given in the percentile table
m = 3.40000
s = 0.20000

skew = 0.2500
Number of data points = 30
For the T year flood, T = 100.0

Percentile	Flood
0.05	6166
0.10	6518
0.15	6774
0.20	6998
0.25	7193
0.30	7372
0.35	7550
0.40	7728
0.45	7908
0.50	8095
0.55	8313
0.60	8512
0.65	8735
0.70	9012
0.75	9308
0.80	9652
0.85	10073
0.90	10701
0.95	11745

VIII. PROGRAM LISTING

```
program confidence;
{Ted Hromadka, Williamson and Schmid, 17782 Sky Park Blvd. Irvine, CA 92714
 Robert Whitley, Dept. of Math.,Univ. of Ca., Irvine, CA, 92717
 1986
}
{This program computes confidence percentiles, 5%(5%)95%, for sampling
 from a Pearson type III distribution with a given skew parameter.
 Input consists of this skew parameter, T for the T year flood, and the
 number of datapoints used in the sample mean and sample standard deviation.
 The user can choose the number of blocks of size 1000 in the simulation
 to be between 1 and 30 and can print out the total histogram of the
 simulation if desired. Given estimates for the mean and the standard
 deviation of the log (base 10) of specific data, percentiles for this
 data can be printed.
}
    const
        block = 1000;
        maxdata = 50;
        h = 0.01; {step size in the tally}
        L = 2; {simulation tallied in [-L,L]}
        maxj = 399; {=(2*L/h)-1}
        maxjplusone = 400;
        scaletest = 100; {sample size used to get scalings}
        leastint= -32767; {(almost) smallest integer in turbo pascal}
        greatestint = 32767; {largest integer in turbo pascal}
    type
        maxvector = array[1..maxdata] of real;
        tvector = array[0..maxj] of real;
        factor = array[0..18] of real;
    var
        NoBlocks:integer;      {# of blocks used in simulation}
        NoDataPoints:integer;{# of points used in the sample estimators for
                               the distribution parameters}
        dvector:maxvector;     {vector of nodatapoints r.v. simulating a
                               sequence of datapoints}
        tally1:tvector;        {histogram tallies of the mean as a check}
        tally2:tvector;        {histogram tallies of the statistic Y of
                               interest: see procedure "statistic"}
        pfactor:factor;        {pfactor[j] is the 0.05+j*(0.05) percentile
                               of Y for j=0(1)18}
        gamma:real;            {skew coefficient}
        b:real;                {sqr(gamma)=4/b, gamma = zero and b infinite
                               correspond to the case of a normal distribution}
        tp:real;               {F(tp)=1-1/T, F N(0,1) d.f.if gamma is zero, else
                               F is the gamma d.f. with parameter b and
                               F(tp)=1-p for gamma positive while F(tp)=p
                               for gamma negative.}
        Kp:real;               {point estimate for the p-th percentile of a
                               Pearson III is mean + s.d. * Kp}
        T:real,                {the T-year flood}
        p:real;                {p=1-(1/T)}
        scale1:real;           {(mean-mid1)/scale1 scales the mean to [-2,2]}
        mid1:real;
        scale2:real;           {(Y-mid2)/scale2 scales the statistic Y
                               to the range [-2,2]}
        mid2:real;
        GammaNotZero:boolean;{false in the non-normal case of finite b}
        GammaPositive:boolean;{true if gamma>0}
        ans1:char;             {n if do not print histogram}
        ans2:char;             {n if don't want percentiles for specific data}
        x,y:real;k,j:integer;


function NormalF(x:real):real;
{see Abramowitz and Stegun, Handbook of Mathematical Functions, page 932.
 NormalF is the distribution function for a normal N(0,1)
```

```
            }
               const
                 p = 0.2316419;
                 b1 =0.319381530;
                 b2 = -0.356563782;
                 b3 = 1.781477937;
                 b4 = -1.821255978;
                 b5 = 1.330274429;
               var
                 temp,temp2,t: real;
             begin {function NormalF}
               t:=1/(1+p*x);
               temp:=b4+b5*t;
               temp:=b3+temp*t;
               temp:=b2+temp*t;
               temp:=b1+temp*t;
               temp:=temp*t;
               temp2:=exp(-x*x/2)/sqrt(2*pi);
               NormalF:=1-temp2*temp;
             end; {function  NormalF}

           function InverseF(p:real):real;
           {finds x such that F(x)=p by interval halving
           }
               const
                 small = 10E-8;
               var
                 x1,x2,x3:real;
                 F1,F2,F3:real;
             begin {function InverseF}
                 x1:=0;
                 F1:=NormalF(x1);
                 x2:=1;
                 F2:=NormalF(x2);
                   while p > F2 do begin
                   x1:=x2;
                   F1:=F2;
                   x2:=x2+1;
                   F2:=NormalF(x2);
                   end;{of while}
                 repeat
                   x3:=(x1+x2)/2;
                   F3:=NormalF(x3);
                   if F3 > p then
                     begin
                     x2:=x3;
                     F2:=F3;
                     end{of if}
                   else
                       begin
                       x1:=x3;
                       F1:=F3;
                       end;


    until F2-F1 < small;
  InverseF:=(x1+x2)/2;
end;{function InverseF}

  procedure RandomNormal(var rn1,rn2:real);
       {a call to RandomNormal(x,y) returns two independent
        normally distributed r.v. as x and y. See Newman and Odell,
        The Generation of Random Variates, page 10, or Abramowitz
        and Stegun, Handbook of Mathematical Functions, page 953
       }
        var
          t1,t2,t3:real;
        begin
          t1:=random;
          t2:=random;
          t3:=sqrt(-2*ln(t1));
          rn1:=t3*sin(2*pi*t2);
          rn2:=t3*cos(2*pi*t2);
        end;{procedure randomnormal}
```

```
function WilsonHilferty(t,shape:real):real;
        {A call to WilsonHilferty(t,shape), with t normally distributed,
         returns a variable which has (approximately) a gamma
         distribution with a (shape) parameter "shape".
         See Kendall and Stuart, The Advanced Theory of Statistics,
         Vol.I,page 400, and see Mathur, A Note on Wilson-Hilferty
         Transformation of Chi-squared, Bull. of the Calcutta Stat.
         Assoc.10(1961)103-105 for a discussion of the error in this
         approximation. It is very good as long as the number of d.f,
         which is 2b when you convert gamma, with shape parameter b,
         to chi-squared,is >14.
        }
     var
        temp:real;
     begin
        temp:=1-(1/(9*shape))+t*sqrt(1/(9*shape));
        temp:=exp(3*ln(temp));
        WilsonHilferty:=shape*temp;
     end;{function WilsonHilferty}

procedure RandomGamma(var rn1,rn2:real);
       {A call to randomgamma(x,y) returns two independent gamma
        distributed r.v. as x and y. The approximation used is
        discussed in the function WilsonHilferty. Note that if
        X has a Pearson III distribution with parameters a,b,and c,
        then (X-c)/a has a gamma distribution with parameter b.
        This procedure assumes a global shape parameter b for the
        gamma functions.
       }
        var
          t1,t2:real;
        begin
          RandomNormal(t1,t2);
          rn1:=WilsonHilferty(t1,b);
          rn2:=WilsonHilferty(t2,b);
        end;{procedure RandomGamma}

 procedure datavector;
 {This procedure generates a vector of dimension nodatapoints
  whose components are independent with the same gamma distribution
  with skew parameter gamma; in the case of a zero skew parameter,
  they are normally distributed.
 }
      var
         j:integer;
         a,b,u:real;
      begin
           j:=1;
           u:=int(nodatapoints/2)+0.5;
              repeat
              if gammanotzero then randomgamma(a,b) else randomnormal(a,b);
              dvector[2*j-1]:=a;
              dvector[2*j]:=b;
              j:=j+1;
              until j>u;
         if odd(nodatapoints) then begin
            if gammanotzero then randomgamma(a,b) else randomnormal(a,b);
            dvector[nodatapoints]:=a;
         end;{if}
       end;{procedure datavector}

 procedure MandS(var mean,sd:real);
 {Returns the mean and standard deviation of the data in the data vector.
 }
      var
         j:integer;
      begin
         mean:=0;
         sd:=0;
         for j:=1 to nodatapoints do begin
             mean:=mean + dvector[j];
             sd:=sd + sqr(dvector[j]);
```

```
        end;{of for}
        mean:=mean/nodatapoints;
        sd:=(sd-nodatapoints*sqr(mean))/(nodatapoints-1);{unbiased}
        sd:=sqrt(sd);
     end;{of procedure mands}

function statistic(x,y:real):real;
      {Given  mean x and standard deviation y, this computes the
       statistic which is being simulated.
      }
      begin
         if gammanotzero then
            begin
              if gammapositive then statistic:=((tp-x)/y)
              else statistic:=-((tp-x)/y);
            end{gammnotzero then}
         else statistic:=((tp-x)/y);{then gamma is zero}
      end;{function statistic}

procedure scale(var scale1,mid1,scale2,mid2:real);
   {procedure scale samples the mean and statistic scaletest times
    and keeps track of the max and min and sets the scale factors
    so that (x-mid1)/scale1 is exactly scaled to [-1,1] for the mean
    and (x-mid2)/scale2 is exactly scaled to [-1,1] for the statistic;
    for the definition of the statistic see the function statistic.
    The procedure tally then counts the  occurrences of the scaled
    numbers in [-2,2] with an interval size of h.
   }
   var
   j:integer;
   temp,x,y,max1,min1,max2,min2:real;
   begin
     datavector;
     mands(x,y);
     min1:=x;
     max1:=x;
     min2:=statistic(x,y);
     max2:=statistic(x,y);
     for j:=1 to scaletest do begin
       datavector;
       mands(x,y);
       if min1>x then min1:=x;
       if max1<x then max1:=x;
       temp:=statistic(x,y);
       if min2>temp then min2:=temp;
       if max2<temp then max2:=temp;
     end;{j for}
     mid1:=(max1+min1)/2;
     scale1:=(max1-min1)/2;
     mid2:=(max2+min2)/2;
     scale2:=(max2-min2)/2;
   end;{procedure scale}

procedure tally(var mean,sd:real);
{Counts the number of times the simulated statistic falls in each interval
 [L+(j-.5)*h,L+(j+.5)*h) and does the same for the sample mean. The
 distribution of the sample mean is known, which gives a check on the
 accuracy of the simulation.
 If more than 32 blocks of 1000 are desired, the sum variable must be
 changed from integer to real to avoid overflow.
}
     var
         j1,j2:integer;
         c1,c2:real;
     begin
          c1:=mean;
          c1:=(c1-mid1)/scale1;
          if (c1>maxj*h) then tally1[maxj]:=tally1[maxj]+1
          else if (c1<-maxj*h) then tally1[0]:=tally1[0]+1
          else        {the first two ifs are to avoid integer overflow}
          begin
            j1:=round((c1+L)/h);
               case j1 of
                        leastint..-1: tally1[0]:=tally1[0]+1;
                             0..maxj: tally1[j1]:=tally1[j1]+1;
```

```
                maxjplusone..greatestint: tally1[maxj]:=tally1[maxj]+1;
                  end;{case}
              end;{third else}
              c2:=statistic(mean,sd);
              c2:=(c2-mid2)/scale2;
              if (c2>maxj*h) then tally2[maxj]:=tally2[maxj]+1
              else if (c2<-maxj*h) then tally2[0]:=tally2[0]+1
              else
              begin
                j2:=round((c2+L)/h);
                  case j2 of
                          leastint..-1: tally2[0]:=tally2[0]+1;
                               0..maxj: tally2[j2]:=tally2[j2]+1;
                   maxjplusone..greatestint: tally2[maxj]:=tally2[maxj]+1;
                  end;{case}
              end;{third else}
          end;{procedure tally}

   procedure histogram;
   {If requested, prints outs the histogram outcome of tally
   }
       var
        s2:real;
        j:integer;
       begin
           writeln(lst,'              HISTOGRAM');
           writeln(lst);

         if GammaNotZero then
          begin
if gammapositive then writeln(lst,'    of the simulation of Y=[(tp-m)/s]')
else  writeln(lst,'    of the simulation of Y=-[(tp-m)/s]');
           writeln(lst,'    with gamma = ',gamma:2:3);
          end {of if}
          else
           writeln(lst,'    of the simulation of Y=[(t-m)/s]');

           writeln(lst);
           writeln(lst,'    m=sample mean, s sample s.d.');
           writeln(lst,'    with ',NoDataPoints,' points in the sample');


           writeln(lst,'    Number of points in the simulation:');
           writeln(lst,'        ',NoBlocks,' blocks of size ',block);
           writeln(lst);
           writeln(lst,'    For each point p listed, the tally is the ');
           writeln(lst,'    number of points in the interval [p-K/2,p+K/2)');
           writeln(lst,'    K = ', h*scale2);
           writeln(lst,'    no listing if tally is zero. ');
           writeln(lst,'    Sum is a running subtotal.');
           writeln(lst);
           writeln(lst,'    Scale factors:');
           writeln(lst,'        scale1 = ', scale1);
           writeln(lst,'        mid1   = ', mid1);
           writeln(lst,'        scale2 = ', scale2);
           writeln(lst,'        mid2   = ', mid2);
           writeln(lst);
           writeln(lst);
           writeln(lst,'     p              Y        sum ');
           writeln(lst,'-------------------------------------');
           s2:=0;
         for j:= 0 to maxj do begin
           if (tally2[j]>0) then begin
             write(lst,((j*h-L)*scale2+mid2):10:5);
             write(lst,'     ',tally2[j]:5:0);
             s2:=s2+tally2[j];
             writeln(lst,'      ',s2:5:0);
           end;{of if}
         end;{of j for}
           writeln(lst);
           writeln(lst);
       end;{of procedure histogram}
```

```
function truemean(normalp:real):real;
   {This function returns the true percentage point of the sample mean
    corresponding to the percentage point normalp of  N(Ø,1).
   }
   var
     temp:real;
   begin
     temp:=normalp,
        if GammaNotZero then {non-normal distribution}
           begin
             temp:=WilsonHilferty(temp,b*nodatapoints);
             truemean:=temp/nodatapoints;
           end {of if}
         else  {normal distribution, b infinite}
           begin
             truemean:=temp/sqrt(NoDataPoints);
           end; {of else}
   end;{function truemean}

procedure percentiles;
{Prints the percentiles 5%(5%)95% of the simulated statistic which
 are found by linear interpolation in the histogram of procedure
 tally.
}
   var
   j,j1,j2,k:integer;
   percent,tmean,error,s,NoSimPoints,pN,interpolation:real;

begin
   writeln(lst);writeln(lst);
   writeln(lst,'                        PERCENTILES');
   writeln(lst);
   writeln(lst,'       (linearly interpolated from histogram)');
   writeln(lst,'       m=sample mean, s sample s.d.');
   writeln(lst,'       with ',NoDataPoints,' points in the sample');
  if GammaNotZero then
   begin
   writeln(lst,'       from a gamma random variable with');
   writeln(lst,'                gamma = ',gamma:6:4);
   writeln(lst,'                b = ', b:8:4);
   writeln(lst,'       Kp = ',Kp:8:5);
   writeln(lst,'       tp = ',tp:8:5);
   writeln(lst,'       This is the Kp and tp for the ',T:5:1,' year flood.');
   writeln(lst);
     if gammapositive then writeln(lst,'       Y=[(tp-m)/s]')
     else  writeln(lst,'       Y=-[(tp-m)/s]');
   end {of if}
  else
   begin
   writeln(lst,'       from a normal N(Ø,1) random variable with');
   writeln(lst,'       t= ',tp:8:5);
   writeln(lst,'       This is the t for the ',T:5:1,' year flood.');
   writeln(lst);
   writeln(lst,'       Y=[(t-m)/s]');
   end; {of else}
   writeln(lst);
   writeln(lst,'       error in mean = m - true sample mean');
   writeln(lst);
   writeln(lst,'       Number of points in the simulation:');
   writeln(lst,'          ',NoBlocks,' blocks of size ',block);
   writeln(lst);
   writeln(lst);

   write(lst,'Percentile             Y                ');
   if gammanotzero then write(lst,'Y-Kp') else write(lst,'Y-t');
   writeln(lst,'          error in mean     ');
   writeln(lst,'_____________________________________________________________________');
   writeln(lst);
   NoSimPoints:=block*NoBlocks;
     write(lst,' ');
     for j:=Ø to 18 do begin
     percent:=(5+j*5);
     write(lst,percent:4:2);
     pN:=(Ø.Ø5+(Ø.Ø5*j))*NoSimPoints;
     if tally2[Ø]>=pN then interpolation:=((-L)*scale2+mid2)
```

```
 else begin
     k:=0;
     s:=tally2[0];
      repeat
        k:=k+1;
        s:=s+tally2[k];
      until s>pN;
      j2:=k;
      if k=maxj then interpolation:=L*scale2+mid2
         else begin
           repeat
           k:=k-1;
           until tally2[k]>0;
           j1:=k;
         end;{else with repeat}
  end; {else executed if tally2[0]<pN}
     interpolation:=((pN-(s-tally2[j2]))/tally2[j2])*(j2-j1)*h+(j1*h-L)+h/2;
     interpolation:=interpolation*scale2+mid2;
     write(lst,'           ',interpolation:9:3);

     pfactor[j]:=interpolation;

     if gammanotzero then write(lst,'          ',interpolation-Kp:9:3)
     else write(lst,'          ',interpolation-tp:9:3);

   if tally1[0]>=pN then  interpolation:=((-L)*scale1+mid1)
   else begin
        k:=0;
        s:=tally1[k];
          repeat
            k:=k+1;
            s:=s+tally1[k];
          until s>pN;
         j2:=k;
         if k=maxj then interpolation:=L*scale1+mid1
            else begin
               repeat
               k:=k-1;
               until tally1[k]>0;
               j1:=k;
            end; {else with repeat}
   end; {else executed if tally1[0]<pN}
      interpolation:=((pN-(s-tally1[j2]))/tally1[j2])*(j2-j1)*h+(j1*h-L)+h/2;
      interpolation:=interpolation*scale1+mid1;
             case j of
                  {the argument of truemean is the 0.05+j*(0.05) percentage
                   point for a N(0,1) random variable}
                0: tmean:=truemean(-1.64485); {.05}
                1: tmean:=truemean(-1.28155); {.10}
                2: tmean:=truemean(-1.03643); {.15}
                3: tmean:=truemean(-0.84162); {.20}
                4: tmean:=truemean(-0.67449); {.25}
                5: tmean:=truemean(-0.52440); {.30}
                6: tmean:=truemean(-0.38532); {.35}
                7: tmean:=truemean(-0.25335); {.40}
                8: tmean:=truemean(-0.12566); {.45}
                9: tmean:=truemean(0.00000);  {.50}
               10: tmean:=truemean(0.12566);  {.55}
               11: tmean:=truemean(0.25335);  {.60}
               12: tmean:=truemean(0.38532);  {.65}
               13: tmean:=truemean(0.52440);  {.70}
               14: tmean:=truemean(0.67449);  {.75}
               15: tmean:=truemean(0.84162);  {.80}
               16: tmean:=truemean(1.03643);  {.85}
               17: tmean:=truemean(1.28155);  {.90}
               18: tmean:=truemean(1.64485);  {.95}
             end;{case}
          error:=interpolation - tmean;
          writeln(lst,'           ',error:7:3);
       end;{j for}
       writeln(lst);
       writeln(lst);
       writeln(lst);
       writeln(lst);
end;{of procedure percentiles}
```

```
procedure floodpercentiles;
     {Uses the global variable pfactor[0..18] and prints a table
      of 10 raised to the power (imputmean + inputsd*pfactor[j]).
     }
      var
        j:integer;
        inputm, inputs:real;

      begin
      writeln;
      writeln('Input estimates for mean and s.d.');
      writeln('mean = ?');
      read(inputm);
      writeln;
      writeln('standard deviation = ?');
      read(inputs);
      writeln;
        writeln(lst);
        writeln(lst);
        writeln(lst);
        writeln(lst,'            FLOOD PERCENTILES');
        writeln(lst,'    10 to the power m+s*Yp');
        writeln(lst,'       Yp the percentile of Y');
        writeln(lst,'       given in the percentile table');
        writeln(lst,'    m = ',inputm:8:5);
        writeln(lst,'    s = ',inputs:8:5);
        writeln(lst);
        writeln(lst,'    skew = ',gamma:6:4);
        writeln(lst,'    Number of data points = ',nodatapoints:4);
        writeln(lst,'    For the T year flood, T = ',T:5:1);
        writeln(lst);
        writeln(lst,'Percentile          Flood');
        writeln(lst,'-----------------------------------');
        writeln(lst,' ');
        for j:=0 to 18 do begin
             write(lst,0.05+j*(0.05):4:2,'               ');
             writeln(lst,exp(ln(10)*(inputm+pfactor[j]*inputs)):6:0);
        end; {j for}
     end; {procedure floodpercentiles}

begin {program confidence}
    writeln('input the number of data vectors');
    read(nodatapoints);
    writeln;
    writeln('input skew coefficient gamma');
    read(gamma);
    writeln;
      if abs(gamma)<0.05 then
         GammaNotZero:=false
       else
           begin
           if (gamma>0) then gammapositive:=true else gammapositive:=false;
           b:=4/sqr(gamma);
           GammaNotZero:=true;
       end; {of else}
     writeln('Computation will be for the T-year flood. Input T.');
     read(T);
     writeln;
     p:=1-(1/T);
     tp:=inverseF(p);    {F N(0,1) d.f.}
     if GammaNotZero then
       begin
         if gammapositive then tp:=WilsonHilferty(tp,b)
         else tp:=WilsonHilferty(-tp,b);
         if gammapositive then Kp:=(tp-b)/sqrt(b)
         else Kp:=-(tp-b)/sqrt(b);
       end; {of gammanotzero then}
    scale(scale1,mid1,scale2,mid2);
    writeln('How many blocks (1-30) do you want to do?');
    read(NoBlocks);
```

```
    writeln;
    for k:=0 to maxj do begin
      tally1[k]:=0;
      tally2[k]:=0;
    end;{of for}
    randomize;
    for k:=1 to NoBlocks do begin
      write('  block #',k:2);
      for j:=1 to Block do begin
        datavector;
        marids(x,y);
        tally(x,y);
      end;{of j for}
    end;{of k for}
      percentiles;
      writeln;
      writeln('Do you want to print the histogram(y/n)?');
      read(ans1);
      if (ans1 = 'y') then histogram;
      writeln;
      writeln('Do you want percentiles for specific data(y/n)?');
      read(ans2);
      if (ans2 ='y') then  floodpercentiles;
end.{of confidence}
```