QA: QA

**Civilian Radioactive Waste Management System**
**Management & Operating Contractor**

# GVP SOFTWARE ROUTINE REPORT

**STN: 10341-1.02-00**
**SDN: 10341-SRR-1.02-00**

**August 2000**

Prepared for:

U.S. Department of Energy
Yucca Mountain Site Characterization Office
P.O. Box 30307
North Las Vegas, Nevada 89036-0307

Prepared by:

TRW Environmental Safety Systems, Inc.
1261 Town Center Drive
Las Vegas, Nevada 89144-6352

Under Contract Number
DE-AC08-91RW00134

## DISCLAIMER

**Civilian Radioactive Waste Management System**
**Management & Operating Contractor**

# GVP SOFTWARE ROUTINE REPORT

**STN: 10341-1.02-00**
**SDN: 10341-SRR-1.02-00**

**August 2000**

Prepared by:

Bryan E. Bullard
Performance Assessment Department

August 24, 2000
Date

Verified by:

Darren M. Jolley
Performance Assessment Department

Aug 24, 2000
Date

Approved by:

Kevin Mon

Kevin G. Mon
Performance Assessment Department

Aug 24, 2000
Date

## CONTENTS

## FIGURES

**Page**



## TABLES

**Page**

# 1. SOFTWARE ROUTINE IDENTIFICATION

Name and Version Number: GVP (Gaussian Variance Partitioning), version 1.02

This routine was developed using Microsoft Developer Studio 97 with Visual Fortran 5.0, Standard Edition.

SRR Document Identification Number: 10341-SRR-1.02-00

SRR Media Number (if applicable): 10341-PC-1.02-00

# 2. DESCRIPTION AND TESTING

GVP is a routine that decomposes a cumulative distribution function (CDF) containing both uncertainty and variability and produces a distribution that characterizes variability. This provides for a better conceptual understanding of model sensitivity to the elements of uncertainty and variability. The outputs of GVP are:

- A text file containing a CDF table for the variability distribution, and
- An output argument which contains the median of the variability distribution.

## 2.1 DESCRIPTION OF SOFTWARE ROUTINE AND THE EXECUTION ENVIRONMENT

The GVP source code is a Fortran program 265 lines in length. It conforms to the Fortran 90 standard and is thus highly portable. The subroutine GVP was developed and tested in the Windows NT 4.0 operating system, and has been compiled with Visual Fortran 5.0, Standard Edition for Microsoft Windows 32 bit operating system environments. GVP compiled as a dynamic link library (GVP.DLL) may be coupled with GoldSim (Golder Associates 2000) though it's external element mechanism. Inserting data elements in the GoldSim environment allows input parameters to be specified. GVP directly links to and runs within GoldSim for modeling waste package failures. The outputs are used by GoldSim to generate distributions for waste package failures and consequent dose.

The CDF table file formats consists of a first line containing the number of rows in the CDF lookup table with the following lines containing two columns of numbers. The first column of numbers is the distribution values in increasing order. The second column contains the cumulative probability values.

Compilation of GVP requires two Fortran modules to be present from the WAPDEG library (CRWMS M&O 1999). These are modDefaultSize and modStandardNormal.

GVP computes the variability distribution by calculation of normal scores. The inputs are read as part of the argument list of GVP, as the elements of array in(*):

in(1) = The fraction of the variance belonging to uncertainty.

in(2) = The fractile value of the uncertainty distribution to place the median value of the variability distribution.
in(3) = Take logarithmic transform (positive value, yes; zero or negative value, no).
in(4) = file index for the input file (combined uncertainty and variability) CDF.
in(5) = file index for the output file (variability) CDF.

The last two inputs are indices (line numbers) within a reference list file (WD4DLL.WAP) for filenames used by several External Functions used by GoldSim for waste package simulation.

The output consists of the variability CDF written to the files indexed in(5), and the median of the variability distribution (written to out(1)).

The GVP DLL follows a project-coding standard that requires all DLL's to accept as input a method variable that controls the operation of the program (see Figure 1). If a DLL is called with the following values of method, the following will occur:

method = 0 Initialize (GVP requires no initialization, thus nothing happens).
method = 1 Calculate (for GVP, compute the variability CDF).
method = 2 Report the version number as out(1).
method = 3 Report the number of input and output arguments as out(1) and out(2), respectively (for GVP, this should yield the values 5 and 1, respectively).
method = 99 Clean up, close any open files.

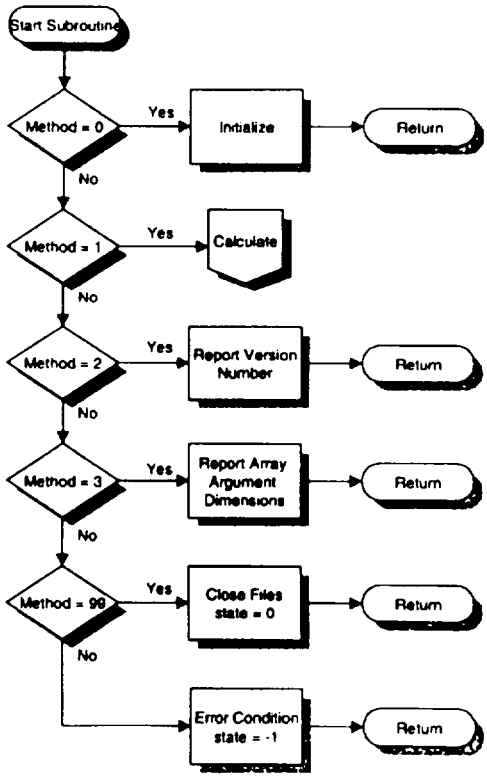


Figure 1. Method calling structure for DLL

## 2.2 DESCRIPTION OF THE ALGORITHM

### 2.2.1 Theoretical Background

Gaussian variance partitioning starts with a distribution that involves both uncertainty and variability and then works backward to obtain two separate distributions, one that characterizes variability and another that characterizes uncertainty. This is accomplished by assuming that uncertainty and variability are independent. If the mixed distribution is normally distributed, i.e. $N(\mu,\sigma_\mu^2+\sigma_v^2)$, then it can be represented as a random variable $\gamma$ having the form

$$\gamma = m + v$$

where $m$ is a normal random variable with mean $\mu$ and variance $\sigma_\mu^2$, and $v$ is a normal random variable with mean zero and variance $\sigma_v^2$. Thus, $\gamma$ is a random variable distributed around the mean $\mu$ with a total variance given by the sum of the variances due to uncertainty and variability. If uncertainty is defined as the uncertainty in the mean value and variability as the variance about that mean, then $\gamma$ can be alternatively parameterized as

$$\gamma \sim N(m,\sigma_v^2), \text{ where } m \sim N(\mu,\sigma_\mu^2)$$

The uncertain mean is represented by the random variable, $m$, which is normally distributed with mean, $\mu$ and variance, $\sigma_\mu^2$. The random variable, $\gamma$, is then the convolution of the distributions of the random variable given by $m$ and a random variable, $v$, which can be represented by the addition of two normal random variables as given above where

$$m \sim N(\mu,\sigma_\mu^2) \text{ and } v \sim N(0,\sigma_v^2)$$

Thus, given the distributions for $m$ and $v$, a variability distribution is realized by sampling a value from the parameter uncertainty distribution and adding it to the mean zero variability distribution.

This partitioning method can be extended to non-normal distributions by means of a score transform (Deutsch and Journel 1992, p.138) mapping the percentiles of the non-normal CDF to those of the standard normal by a lookup table. The normal score transforms works best if the non-normal CDF is as symmetric as possible. This may sometimes be accomplished by using the natural logarithms of CDF values. The natural logarithms of the CDF values are used to perform the normal score transformation and the transformed distribution is used to partition the total variance of the transformed distribution between uncertainty and variability. Finally the normal score transformation is applied in reverse to the resultant distributions to obtain a final distribution for variability.

The GVP subroutine was developed to effectively create variability distributions from randomly distributed input data consistent with the above approach.

### 2.2.2 Implementation

The procedure steps in the GVP code are graphically presented in Figure 2.

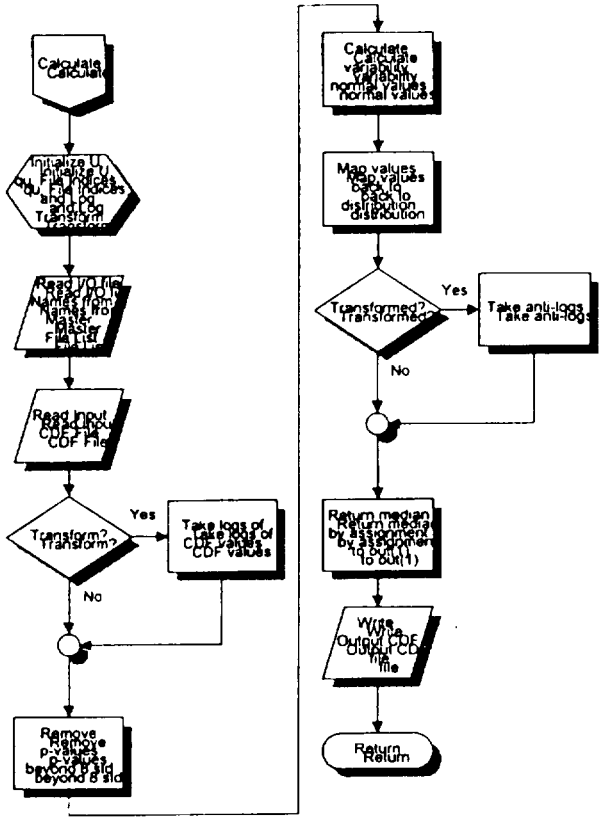


Figure 2. Flowchart of GVP Algoritm

The steps in the algorithm are outlined below:

1. The input CDF is read in. Probabilities are stored in a vector called *pvals* and values are stored in a vector called *vals*.

2. If in(3) > 0 then natural logarithms are taken of the CDF values (i.e., ln(*vals*)).

3. If any of the input CDF probabilities, *pvals*, are less than 1.0E-15 or greater than (1 – 1.0E-15) then they are copied to the output CDF unmodified.

4. The standard normal score value that corresponds with *qu* is given by

$$zu = \sqrt{U} \cdot G^{-1}(qu) \qquad \text{(Eq. 1)}$$

   where *G(x)* is the standard normal CDF and $G^{-1}(x)$ is the inverse normal CDF.

5. The vector *zv* is assigned the standard normal values with mean *zu* and variance *(1-U)* that correspond with the variability distribution i.e.,

$$zv = zu + \sqrt{1-U} \cdot G^{-1}(pvals) \qquad \text{(Eq. 2)}$$

6. The variability distribution CDF values are evaluated by linear interpolation of the probability values given by *pvals* and distribution values given by *vals* (or ln(*vals*), see step 2). The forward normal is taken of each *zv* value, and the corresponding variability CDF value is determined through linear interpolation

$$xv = L(G(zv)) . \qquad \text{(Eq. 3)}$$

where,

$$L(p) = vals_i + (vals_{i+1} - vals_i) \cdot \left[ \frac{p - pvals_i}{pvals_{i+1} - pvals_i} \right] \quad for \quad pvals_i \le p < pvals_{i+1} \qquad \text{(Eq. 4)}$$

7. If in(3) > 0 then exponentials are taken of the CDF values (i.e., exp(*xv*)).

8. Output CDF table of distribution values (*xv* or exp(*xv*)) and probabilities (*pvals*) for the variability distribution.

## 2.3 DESCRIPTION OF TEST CASE

The GVP routine performance was verified by running it in GoldSim and comparing its results to that of the Mathcad calculations. Two test cases were run to validate the routine. The first case tests the linear interpolation option; the second case tests the semi-log interpolation option.

Running the GVP routine as a DLL, the following values are inserted in the input stream:

**Test Case 1**

in(1) = 0.65 (the fraction of the variance belonging to uncertainty)
in(2) = 0.75 (the fractile value for uncertainty representing the median value for variability)
in(3) = -1.0 (do not take logarithmic transforms)
in(4) = 1 (file index for the input file CDF)
in(5) = 2 (file index for the output file (variability) CDF)

**Test Case 2**

in(1) = 0.65 (the fraction of the variance belonging to uncertainty)
in(2) = 0.75 (the fractile value for uncertainty representing the median value for variability)
in(3) = 1.0 (take logarithmic transforms)
in(4) = 1 (file index for the input file CDF)
in(5) = 3 (file index for the output file (variability) CDF)

Commercially available software used to verify the GVP routine are:

- Mathcad 2000 Professional. This commercially available software was used to perform hand calculation verification of GVP

- Excel 97 SR-2. This commercially available software was used to compare the outputs from MathCad and GVP.

The above software programs were executed on a workstation equipped with a Pentium II processor in the Windows NT operating system

The Mathcad calculation uses the built-in MathCad functions exp() (exponential), ln() (natural logarithm), the square root function, cnorm() (the standard normal CDF), qnorm() (the inverse normal CDF), and linterp() (a linear interpolator). The Mathcad calculation listing is included in Section 3.3.

The test case requires an input text file, WD4DLL.WAP, which is a list of filenames to be read by GVP. A listing of WD4DLL.WAP is provided in Section 3.4. Lines in the file contain the names of files used by GVP for the input and output CDFs.

## 2.4 DESCRIPTION OF TEST RESULTS

The outputs were imported to an Excel worksheet to compare and quantify the differences. The Excel results for each test case are presented in Tables 1 and 2. These tables present comparisons between the GVP outputs and MathCad calculation results using the same CDF. Results agree to the eleven digits produced by GVP. From the tables, it is concluded that the GVP routine is verified by hand calculation.

Table 1. Test Case1 (linear interpolation) Comparison

| GVP version1.02 | Mathcad | Difference |
|---|---|---|
| 20.0000000000 | 20.0000000000 | 0.0000000000 |
| 24.1512337640 | 24.1512337640 | 0.0000000000 |
| 25.1829752470 | 25.1829752470 | 0.0000000000 |
| 25.9233340590 | 25.9233340590 | 0.0000000000 |
| 26.5317583340 | 26.5317583340 | 0.0000000000 |
| 27.0670735600 | 27.0670735600 | 0.0000000000 |
| 27.5605640820 | 27.5605640820 | 0.0000000000 |
| 28.0345596840 | 28.0345596840 | 0.0000000000 |
| 28.5122480440 | 28.5122480440 | 0.0000000000 |
| 29.0353619260 | 29.0353619260 | 0.0000000000 |
| 30.0000000000 | 30.0000000000 | 0.0000000000 |

Table 2. Test Case 2 (semi-log interpolation) Comparison

| GVP version1.02 | Mathcad | Difference |
|---|---|---|
| 20.0000000000 | 20.0000000000 | 0.0000000000 |
| 24.1486262450 | 24.1486262450 | 0.0000000000 |
| 25.1800557970 | 25.1800557970 | 0.0000000000 |
| 25.9219382060 | 25.9219382060 | 0.0000000000 |
| 26.5270580870 | 26.5270580870 | 0.0000000000 |
| 27.0659416920 | 27.0659416920 | 0.0000000000 |
| 27.5560816580 | 27.5560816580 | 0.0000000000 |
| 28.0339774560 | 28.0339774560 | 0.0000000000 |
| 28.5078637460 | 28.5078637460 | 0.0000000000 |
| 29.0347867480 | 29.0347867480 | 0.0000000000 |
| 30.0000000000 | 30.0000000000 | 0.0000000000 |

## 2.5 RANGE OF INPUT PARAMETER VALUES OVER WHICH RESULTS WERE VERIFIED

As the method applied is invariant to transformation of the distribution values, these results verify the procedure over all possible ranges (within the limits of numerical precision of the interpolation step).

## 2.6 IDENTIFICATION OF LIMITATIONS ON SOFTWARE ROUTINE OR VALIDITY

GVP will execute properly if the following ranges and types of parameter values are met:

- The fraction of the variance belonging to uncertainty must be a fraction between zero and one inclusive.
- The fractile value of the uncertainty distribution to place the median value of the variability distribution must be a fraction between zero and one inclusive.
- If semi-log interpolation is used, the distribution values of the input CDF must all be positive.
- The input CDF must be properly formatted with distribution and probability values monotonically increasing.

# 3. SUPPORTING INFORMATION

## 3.1 DIRECTORY LISTING OF EXECUTABLES AND DATA FILES

```
Directory of DLLs-SRRs\GVP

Program files:
04/26/00  12:00p              271,872 gvp.dll
08/22/00  10:15a               42,099 gvp.gsm

Input files:
08/22/00  10:05a                   50 WD4DLL.wap
08/22/00  10:13a                  180 WDgvp-in.cdf

Output files:
08/22/00  10:14a                  759 WDgvp-outxv.cdf
08/22/00  10:15a                  759 WDgvp-outyv.cdf
```

## 3.2 COMPUTER LISTING OF SOURCE CODE

```
subroutine gvp(method, state, in, out)
!
!  Subroutine to perform Gaussian Variance Partitioning.
!
!  1. Read combined cdf from an input file, the uncertainty
!     variance share, and the uncertainty quantile level.
!  2. Find/print the variability cdf.
!  Note if log transform option is used the user is responsible
!  for values being in the proper range for the log function.
!
!DEC$ ATTRIBUTES dllexport,c :: gvp
!DEC$ ATTRIBUTES ALIAS : "GVP" :: GVP
!DEC$ ATTRIBUTES value       :: method
!DEC$ ATTRIBUTES reference   :: state
!DEC$ ATTRIBUTES reference   :: in
!DEC$ ATTRIBUTES reference   :: out
      USE ModDefaultSize
      USE ModStandardNormal
      IMPLICIT NONE
      integer(IKind) :: method   ! tells gvp what to do
      integer(IKind) :: state    ! returns, 0 = OK, -1 = FATAL
```

```
      real(RKind)     :: in(*)     ! input arguments
      real(RKind)     :: out(*)    ! output arguments
      real(RKind),     PARAMETER :: VERSION = 1.02
      integer(IKind), PARAMETER :: NUMIN = 5, NUMOUT = 1
      integer(IKind), PARAMETER :: INITIALIZE = 0
      integer(IKind), PARAMETER :: CALCULATE = 1
      integer(IKind), PARAMETER :: VERSN = 2
      integer(IKind), PARAMETER :: ARGMNTS = 3
      integer(IKind), PARAMETER :: CLEANUP = 99
      integer(IKind) :: cdfunit, filunit, errunit
      integer(IKind) :: i, n, n1, n2, idxinp, idxout
      real(RKind)    :: U, qu, lntrns, V, zu, medv, epsilon
      character(LEN = 80) :: filefile, inputcdf, outputcdf, line1
      real(RKind), ALLOCATABLE, DIMENSION(:) :: vals
      real(RKind), ALLOCATABLE, DIMENSION(:) :: pvals
      real(RKind), ALLOCATABLE, DIMENSION(:) :: zv
      real(RKind), ALLOCATABLE, DIMENSION(:) :: xv
      logical(LKind) :: OK
      static errmsg
      character(LEN = 80) errmsg
      integer(IKind) :istrloc
      real(RKind)    rstrloc
      equivalence (istrloc, rstrloc)
!
!*****************************************************************
!
      istrloc = loc(errmsg)
      state = 0
      select case (method)
      case (INITIALIZE)                ! Initialize
        continue
      case (VERSN)                     ! Report code version
        out(1) = VERSION
      case (ARGMNTS)                   ! Report number of arguments
        out(1) = NUMIN
        out(2) = NUMOUT
      case (CLEANUP)                   ! Cleanup
        close(unit = filunit)
        close(unit = cdfunit)
        close(unit = errunit)
      case default                     ! Error trap for unknown method
        errmsg = 'gvp crashed, unknown method'C
        out(1) = rstrloc
        state = -1
        errunit = nextfreeunit()
        open(unit = errunit, file = 'gvperror.log')
        write(errunit,*) 'gvp crashed method = ',method
        close(unit = errunit)
      case (CALCULATE)                 ! Perform calculations
        U  = in(1)
        qu = in(2)
        lntrns = in(3)
        idxinp = in(4)
        idxout = in(5)
!
!  Read I/O CDF-File names from master list file
!
        filunit = nextfreeunit()
        filefile = 'WDSCDF.MAP'
        inquire(file = filefile, exist = OK)
        if (.not. OK) then
          errmsg = 'gvp: file list not found'C
          out(1) = rstrloc
          state = -1
          errunit = nextfreeunit()
          open(unit = errunit, file = 'gvperror.log')
          write(errunit,*) 'Cannot find file list, ',filefile
          close(unit = errunit)
          return
        end if
        open(unit = filunit, file = filefile, err = 1750)
```

```
      n = max(idxinp, idxout)
      do i = 1, n
        read(filunit,*, err = 1750) line1
        if (i .eq. idxinp) inputcdf  = line1
        if (i .eq. idxout) outputcdf = line1
      end do
      close(unit = filunit, err = 1750)
!
!  Open Input CDF-File and read contents
!
      inquire(file = inputcdf, exist = OK)
      if (.not. OK) then
        errmsg = 'gvp: input cdf file not found'C
        out(1) = rstrloc
        state = -1
        errunit = nextfreeunit()
        open(unit = errunit, file = 'gvperror.log')
        write(errunit,*) 'input cdf file not found'
        close(unit = errunit)
        return
      end if
      cdfunit = nextfreeunit()
      open(unit = cdfunit, file = inputcdf, err = 1750)
      read(cdfunit, *, err = 1750) n
      ALLOCATE(vals(n))
      ALLOCATE(pvals(n))
      ALLOCATE(zv(n))
      ALLOCATE(xv(n))
      do i = 1, n
        read(cdfunit,*, err = 1750) vals(i), pvals(i)
      end do
      close(unit = cdfunit, err = 1750)
!
!  Perform Calculations
!  If log transformed (lntrns) then take logs
!
      if (lntrns .gt. 0.0) then
        do i = 1, n
          vals(i) = log(vals(i))
        end do
      endif
!
!  Check for limits of normal functions, remove p-values
!  beyond eight standard deviations
!
      n1 = 1
      n2 = n
      epsilon = 1.0D-15
      do while (pvals(n1) .le. epsilon)
        xv(n1) = vals(n1)
        n1 = n1 + 1
      end do
      do while (pvals(n2) .ge. (1.0 - epsilon))
        xv(n2) = vals(n2)
        n2 = n2-1
      end do
!
!  calculate normal values for variability and map back
!  to distribution
!
      V = 1-U
      zu = sqrt(U)*InvNor(qu)
      medv = linterpl(n,pvals,vals,FwdNorm(zu))
      do i = n1, n2
        zv(i) = zu + sqrt(V)*InvNor(pvals(i))
        xv(i) = linterpl(n,pvals,vals,FwdNorm(zv(i)))
      end do
!
!  If log transformed then take antilogs
!
      if (lntrns .gt. 0.0) then
```

```
          medv = exp(medv)
          do i = 1, n
            xv(i) = exp(xv(i))
          end do
        endif
!
!  Output results and clean up
!
        out(1) = medv
        cdfunit = nextfreeunit()
        open(unit = cdfunit, file = outputcdf, err = 1750)
        write(cdfunit,*, err = 1750) n
        do i = 1, n
          write(cdfunit,3332, err = 1750) xv(i), pvals(i)
 3332     format(1x,1pe17.10,2x,e22.15)
        end do
        write(cdfunit,*, err = 1750)
        write(cdfunit,3330, err = 1750) VERSION
        write(cdfunit,3331, err = 1750) U, qu
        write(cdfunit,3338, err = 1750) ( i, in(i), i = 1, NUMIN )
        write(cdfunit,*, err = 1750)
 3330   format('! Output from gvp version ',f4.2)
 3331   format('! Sampled random variables U =',f9.5,',qu =',f9.5)
 3338   format('! argument in(',I2,') = ',f12.5)
        close(unit = cdfunit, err = 1750)
        DEALLOCATE(vals, pvals, zv, xv)
      end select
      return
 1750 continue ! I/O error exit
        errmsg = 'gvp crashed, unknown I/O error'
        out(1) = rstrloc
        state = -1
        errunit = nextfreeunit()
        open(unit = errunit, file = 'gvperror.log')
        write(errunit,*) 'gvp crashed, unknown I/O error'
        close(unit = errunit)
      return
      CONTAINS    ! linterpl, nextfreeunit
!
!*********************************************************************
!
      real(RKind) FUNCTION linterpl(n, x, y, xval)
!
!    linear interpolation routine from a lookup table.
!    Input : n, x, y, xval
!    Output: (function value)
!    Local : i, ii
!
!    Arguments
!
      integer(IKind) :: n
      real(RKind)    :: x(*), y(*), xval
!
!    Local variable
!
      integer(IKind) :: i, ii
!
      if (xval .le. x(1)) then
        linterpl = y(1)
      else if (xval .ge. x(n)) then
        linterpl = y(n)
      else
        ii = 2
        do while (xval .gt. x(ii))
          ii = ii+1
        end do
        i = ii-1
        linterpl = y(i) + (y(ii)-y(i))*(xval - x(i))/(x(ii)-x(i))
      end if
      RETURN
      END FUNCTION linterpl
```

```
!
!*****************************************************************
!
      integer(IKind) FUNCTION nextfreeunit()
!
!  Find the smallest unit number not currently attached and in use.
!  Avoid units 5 and 6.
!  Input : (none)
!  Output: (function value)
!  Local : i, InUse
!
!  Local variables
!
      integer(IKind) :: i
      logical InUse
!
      InUse = .true.
      i = 0
      do while (InUse)
        i = i + 1
        if(i .ne. 5 .and. i .ne. 6) then
          inquire(i, opened = InUse)
        end if
      end do
      nextfreeunit = i
      RETURN
      END FUNCTION nextfreeunit
!
!*****************************************************************
!
      END subroutine gvp
```

## 3.3 MATHCAD CALCULATION

**GVP Mathcad Calculation** $N := 10$ $i := 0..N$

Input: $U := 0.65$ $qu := 0.75$ $F_i := \frac{i}{N}$ $x_i := 20 + i$

$$qnorm1(x) := \begin{cases} \infty & \text{if } x \geq 1 - 1\cdot10^{-15} \\ -\infty & \text{if } x \leq 1\cdot10^{-15} \\ qnorm(x,0,1) & \text{otherwise} \end{cases}$$

qnorm is modified to except the probability zero and one end points Note that these end points are treated differently for the Mathcad calculation than in the program (See algorithm description, step 3).

Variablity Median Values (out1a and out2a are linear and semi-log interpolation respectively)

$out1a := linterp(F, x, cnorm(\sqrt{U}\cdot qnorm1(qu)))$ $out1a = 27.0670735599$

$out1b := exp(linterp(F, ln(x), cnorm(\sqrt{U}\cdot qnorm1(qu))))$ $out1b = 27.065941692$

Variability Distributions (xv and yv are linear and semi-log interpolation respectively)

$xv_i := linterp(F, x, cnorm(\sqrt{U}\cdot qnorm1(qu) + \sqrt{1-U}\cdot qnorm1(F_i)))$

$yv_i := exp(linterp(F, ln(x), cnorm(\sqrt{U}\cdot qnorm1(qu) + \sqrt{1-U}\cdot qnorm1(F_i))))$

$$x = \begin{pmatrix} 20 \\ 21 \\ 22 \\ 23 \\ 24 \\ 25 \\ 26 \\ 27 \\ 28 \\ 29 \\ 30 \end{pmatrix} \quad F = \begin{pmatrix} 0 \\ 0.1 \\ 0.2 \\ 0.3 \\ 0.4 \\ 0.5 \\ 0.6 \\ 0.7 \\ 0.8 \\ 0.9 \\ 1 \end{pmatrix} \quad xv = \begin{pmatrix} 20 \\ 24.151233764 \\ 25.182975247 \\ 25.923334059 \\ 26.531758334 \\ 27.06707356 \\ 27.560564082 \\ 28.034559684 \\ 28.512248044 \\ 29.035361926 \\ 30 \end{pmatrix} \quad yv = \begin{pmatrix} 20 \\ 24.148626245 \\ 25.180055797 \\ 25.921938206 \\ 26.527058087 \\ 27.065941692 \\ 27.556081658 \\ 28.033977456 \\ 28.507863746 \\ 29.034786748 \\ 30 \end{pmatrix}$$

## 3.4 COMPUTER LISTING OF TEST DATA INPUT AND OUTPUT

Input master file (WD4DLL.wap).

```
WDgvp-in.cdf
WDgvp-outxv.cdf
WDgvp-outyv.cdf
```

Input CDF for test cases (file: WDgvp-in.cdf).

```
11
20.0      0.0
21.0      0.1
22.0      0.2
23.0      0.3
24.0      0.4
25.0      0.5
26.0      0.6
27.0      0.7
28.0      0.8
29.0      0.9
30.0      1.0
```

GVP output CDF for the first (linear interpolation) test case (file: WDgvp-outxv.cdf).

```
          11
  2.0000000000E+01   0.0000000000000000E+00
  2.4151233764E+01   1.0000000000000000E-01
  2.5182975247E+01   2.0000000000000000E-01
  2.5923334059E+01   3.0000000000000000E-01
  2.6531758334E+01   4.0000000000000000E-01
  2.7067073560E+01   5.0000000000000000E-01
  2.7560564082E+01   6.0000000000000000E-01
  2.8034559684E+01   7.0000000000000000E-01
  2.8512248044E+01   8.0000000000000000E-01
  2.9035361926E+01   9.0000000000000000E-01
  3.0000000000E+01   1.0000000000000000E+00

! Output from gvp version 1.02
! Sampled random variables U =  0.65000,qu =  0.75000
! argument in( 1) =      0.65000
! argument in( 2) =      0.75000
! argument in( 3) =     -1.00000
! argument in( 4) =      1.00000
! argument in( 5) =      2.00000
```

GVP output CDF for the second (semi-log interpolation) test case (file: WDgvp-outyv.cdf).

```
          11
 2.0000000000E-01   0.000000000000000E+00
 2.4146626245E-01   1.000000000000000E-01
 2.5180055797E+01   2.000000000000000E-01
 2.5921938206E+01   3.000000000000000E-01
 2.6527058087E+01   4.000000000000000E-01
 2.7036941692E+01   5.000000000000000E-01
 2.7556381658E+01   6.000000000000000E-01
 2.8033977456E+01   7.000000000000000E-01
 2.8507863746E+01   8.000000000000000E-01
 2.9034786748E+01   9.000000000000000E-01
 3.0000000000E+01   1.000000000000000E+00

! Output from gsp version 1.02
! Sampled random variables U =  0.65000,qu =  0.75000
! argument in( 1) =       0.65000
! argument in( 2) =       0.75000
! argument in( 3) =       1.00000
! argument in( 4) =       1.00000
! argument in( 5) =       3.00000
```

## 4. REFERENCES


CRWMS M&O 1999. *Testing of Software Routine to Determine Deviate and Cumulative Probability: ModStandardNormal Version 1.0.* CAL-EBS-MD-000004 REV 00. Las Vegas, Nevada: CRWMS M&O. ACC: MOL.19991018.0213.

Deutsch, C.V. and Journel, A.G. 1992. *GSLIB Geostatistical Software Library and User's Guide.* New York, New York: Oxford University Press. TIC: 224174.

Golder Associates 2000. *User's Guide, GoldSim, Graphical Simulation Environment.* Version 6.02. Manual Draft #4 (March 17, 2000). Redmond, Washington: Systems Simulation Group Golder Associates Inc. TIC: 247347.