*Issue No. 15*

*1985 March*

# The GENSTAT Newsletter

NAG
NUMERICAL ALGORITHMS GROUP

Editors

R W Payne
Rothamsted Experimental Station
Harpenden
Hertfordshire
AL5 2JQ

M G Richardson
NAG Central Office
Mayfield House
Oxford
OX2 7DE

Printed and produced by the Numerical Algorithms Group

# GENSTAT NEWSLETTER
## Issue No. 15

NP1050 1985 March

## Contents



Published Twice Yearly by
Rothamsted Experimental Station Statistics Department
and the Numerical Algorithms Group Limited

Printed 1985 October

# Editorial

The number of Genstat sites continues to grow more or less linearly with time – there are now well over 300 sites outside of the ARC. With this growth has come concomitant growth in NAG staff in the statistical packages area: over a twelve month period these will have increased from one to four.

All first-line support of Genstat for non-ARC sites now comes from NAG and these sites should not normally contact the package developers at Rothamsted or the various machine range implementors, except through NAG. The developers and implementors are, of course, still available to provide back-up to NAG when necessary.

This issue of the Newsletter contains articles from a geographically wide range of authors, most of whom submitted material in machine-readable form, for which we are extremely grateful. Our need for submissions continues to be as great as ever: very few articles describing Genstat *applications* have been submitted recently and we urge readers to consider whether their own application might not be of general interest.

When this Newsletter appears, the Genstat Conference will probably have come and gone. We hope that many of our readers will have been able to attend and will have enjoyed the Conference. As usual, many of the papers presented will also appear in future issues of the Newsletter.

# Corrigenda

## 14.6 Dendrograms and Ziggurats

References to 'the previous article', 'the previous program', etc. should be to 'the following article', etc.

On p. 14, line 4, $2^{n+1}$ should read $2^{n-1}$.

## 14.7 Drawing Pretty Dendrograms

The `reference` statement for the program on p. 20 should read

```
'refe' dendrograms_on_the_sigma
```

In some copies of Newsletter 14 the punctuation marks in the programs are very faint. Readers needing clarification of the programs may obtain listings from NAG Central Office.

# Letter to the Editor

*B. L. Shea*
*NAG Central Office*
*Mayfield House*
*256 Banbury Road*
*Oxford OX2 7DE*
*United Kingdom*

## The Role of a Subroutine Library for Statisticians

Anyone who has carried out a statistical analysis of some kind will almost certainly have used one of the many widely available statistical packages. Indeed, there are a great many such packages designed to relieve the user of unnecessary programming effort whilst enabling him to perform an analysis of some sort. Whilst there can be little doubt that data analysis is most easily done using such packages, there are perhaps occasions when having access to a subroutine library would be advantageous. Users in academic institutions generally have easy access to packages such as Genstat. Hence the only data analysis routines in a subroutine library which would be of interest to them are those which carry out non-standard analyses not generally available in package form, such as robust statistics. Obviously, doing statistical analysis using library routines is much more time consuming and seems a rather alien approach to adopt.

One should not perhaps be over-concerned with the academic statistician who is so richly endowed with packages. Consider instead the average statistician working for a small company which subscribes to a subroutine library for mathematicians and scientists. If statistical packages are deemed too expensive the statistician may not be too badly handicapped if statistical routines are available in the library.

Many packages may not present the results to the user's satisfaction. There may be an advantage in being able to write in-house programs which call library routines and interact with local graph plotting facilities. The user then has complete control over output and can produce high quality graphics.

Clearly there would be advantages in having arguments to a library routine call which allowed the user the option of whether or not to display (on the line printer) some or all of the results.

As new methodologies become available these might be more speedily passed on to users if they are distributed in subroutine rather than package form.

A subroutine library obviously has a key role to play in providing the basic linear algebra routines such as matrix inversion and function minimization.

What statistical routines should be in a subroutine library and who will the users be? I would be grateful to hear from anyone who has views on this.

# Fitting Exponential or Weibull Distributions to Survival Data

*A.W.A. Murray*
*Statistics Department*
*Rothamsted Experimental Station*
*Harpenden*
*Hertfordshire AL5 2JQ*
*United Kingdom*

Survival data consist of observed times of death (failure in the case of inanimate objects) or times of loss of an individual from the study for other causes; these latter are called censored observations. We may wish to fit an appropriate distribution to these data and estimate its parameters. The exponential and Weibull distributions are often suitable probability models for survival data. This article describes a simple diagnostic procedure to aid in the choice of distribution and shows how these distributions, or binary mixtures of them, may be fitted.

The diagnostic procedures make use of the empirical product-limit or Kaplan-Meier estimate of the survivor function. This is essentially a 1-sample cumulative distribution function but allows the inclusion of censored data. The formula is given by

$$\hat{S}(t) = \prod_{i|t_i<t} \frac{(n_i - d_i)}{n_i} \qquad (1)$$

where $\hat{S}$ is the product-limit or Kaplan-Meier estimate of the survivor function, $n_i$ is the number of individuals at risk at time $t_i$ and $d_i$ is the number of observed deaths at time $t_i$. The survivor function, $S$, for exponentially distributed data, is given by

$$S(t) = exp[-\lambda t] \qquad (2)$$

for some parameter $\lambda$. The diagnostic procedure is to plot $ln(\hat{S}(t))$ against $t$, which should give a straight line of slope $\lambda$ if the data are distributed in this fashion. The survivor function for a Weibull distribution is given by

$$S(t) = exp[-(\lambda t)^w] \qquad (3)$$

for some parameters $\lambda$,*w2F3*. The identity

$$ln[-ln(S(t))] = w(ln(t)+ln(\lambda)) \qquad (4)$$

shows that a plot of $ln[-ln(\hat{S}(t))]$ against $ln(t)$ should give a straight line of slope $w$ and intercept $wln(\lambda)$.

Program 1 calculates and displays the Kaplan-Meier survivor function, together with 95% confidence limits according to Greenwood's formula as modified by Kalbfleisch and Prentice (1980). Diagnostic plots for exponential and Weibull distributions are performed. There is an option to select plotter output for the graphs, which will provide very much better detail than the line-printer style of output. The necessary data, to be provided on a secondary input stream, are the times, `T`, of death or censorship and an indicator variable, `C`, which is `0` where an observation is censored and `1` if a death has been observed. An example data set is shown below. A heading, `HDNG`, must be provided to identify the data

```
'' Example data set ''

'' random times from an exponential distribution with theoretical mean 100 ''
62 184 23 151 50 4 143 162 14 88 126 20 27 52 124 203 100 18 25 26
21 250 130 164 135 56 16 46 79 59 1 157 6 28 3 283 29 79 29 230
7 127 60 210 1 146 112 69 98 124 68 27 26 82 59 32 42 19 223 164
```

```
90 32 55 18 75 276 117 37 44 256 28 281 45 127 97 105 64 53 218 41
32 62 88 36 156 76 85 24 145 176 5 93 116 94 60 63 85 17 19 134 'EOD' times
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1
0 1 1 0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1
'EOD' 10% censored at random
```

If plotter output is desired then the line

```
    'SET'  PLOTTER = YES
```

should be included in the program and a secondary output file made available when Genstat is run.

```
'REFE' PROGRAM (1)
''
Analysis of survival data (lifetimes in days, including censored
observations)
.
calculation of Kaplan-Meier (product-limit) estimate of survivor function
.
also gives upper and lower 95% confidence bounds using Greenwood's
formula as modified by Kalbfleisch  J D and Prentice R L (1980),
The Statistical Analysis of Failure Time Data. New York: Wiley
.
diagnostic test for fit of Exponential distribution
.
diagnostic test for fit of Weibull distribution
''
'SET' PLOTTER=YES
'HEAD' HDNG
'INPUT' 2
'READ' HDNG
'PRINT' HDNG
'READ/S,NUN=V' T,C
'INPUT' 1
'RUN'
'INTE' YES=1  : NO=2
'NAME' YESNO=yes,no
'FACTOR' REMOVED$YESNO,T
'GROUP' REMOVED=RANK(C)
'TABLE/M' HOW_MANY$REMOVED
'TABU' T; ASSCT=HOW_MANY
'SCAL' NT,T_NTP1,N0,FIRST,TAIL_TIME,RH_TAIL
'EQUA' N0=HOW_MANY$2X,1
'VARI' TXC$T
'CALC' TXC=T*C/C
:      TXC=REPMV(MIN(TXC))
'GROUP' DTIMES=INTPT(TXC)
'CALC' NT=NLEV(DTIMES)
:      T_NTP1=2*NT+1
'RUN'
```

```
'UNIT'$NT
'TABLE' TAB_DTIME$DTIMES
'TABU' TXC; MEAN=TAB_DTIME
'EQUA' TIME=TAB_DTIME
'CALC' LOG_TIME=LOG(TIME)
'GROUP' TIMES=LIMITS(T;TIME)
'RUN'
'RESTR' T$REMOVED=NO
'TABLE' TAB_TIME$TIMES
'TABU' T;ASSCT=TAB_TIME
'EQUA' DEATHS=TAB_TIME
: LAGDEATHS=0,DEATHS
'RESTR' T$REMOVED=YES
'TABU' T;ASSCT=TAB_TIME
'EQUA' CENSORED=TAB_TIME
'CALC' NJ_DJ=N0-CUM(CENSORED)-CUM(DEATHS)
:      NJ=N0-CUM(CENSORED)-CUM(LAGDEATHS)
: TERM=NJ_DJ/NJ
''log Kaplan-Meier estimate of survivor fn''
: LOG_S_T=CUM(LOG(TERM+0.00001*(TERM.EQ.0)))
: S_T=EXP(LOG_S_T) ''Kaplan-Meier (product-limit) estimate of survivor fn''
''corrected formula for variance function, Kalbfleisch & Prentice p.15''
:      SE_S_T=SQRT(CUM(DEATHS/(NJ*(NJ_DJ+(NJ_DJ.EQ.0))))/(LOG_S_T**2))
:      LO_LIM=S_T**(EXP(1.96*SE_S_T))  '' lower confidence bound ''
:      HI_LIM=S_T**(EXP(-1.96*SE_S_T))  '' upper confidence bound ''
:      ELEM(LO_LIM,HI_LIM;NT)=0.000
:      W_TEST=LOG(-LOG_S_T)
:      OBS_FREQ=1.00-S_T
:      FIRST=ELEM(OBS_FREQ;1)
:      OBS_FREQ=DIFF(OBS_FREQ;1)
:      ELEM(OBS_FREQ;1)=FIRST
:      TAIL_TIME=ELEM(TIME;NT)
:      RH_TAIL=1-SUM(OBS_FREQ)
:      FIRST=ELEM(LO_LIM;1)
'HEAD' HTAIL= ''   >''
:      HY= '' Value of survivor function''
:      HY(1)= '' log Kaplan Meier survivor function''
:      HY(2)= ''log (-log Kaplan Meier survivor function)''
:      HX(1)= '' time (days)''
:      HX(2)= ''  log(time)''
:      HT= '' Graph of empirical (Kaplan-Meier) survivor function''
:      HT(1)= '' Graph to test possible fit of Exponential distribution ''
:      HT(2)= '' Graph to test possible fit of Weibull distribution ''
:      L=''L''
:      LPP=''LPP''
:      DASH=''-''
:      LLL=''L2L1L2''
```

```
'LINE' 5
'PRINT' HDNG
'PRINT/MARG=TOTAL' HOW_MANY$10
'PRIN/P' TIME,DEATHS,CENSORED,OBS_FREQ,S_T,LO_LIM,HI_LIM$ 3(10),4(10.3)
'JUMP' NO_TAIL *(RH_TAIL.EQ.0)
'PRINT/C,LABR=1,VAR=1' HTAIL,TAIL_TIME,RH_TAIL$ 2X,1,4,30.3
'LABEL' NO_TAIL
'VARI' GTIME,G_S_T,G_LO,G_HI$T_NTP1
:      YXSCALE=0,1.0,*,*
'EQUA' GTIME,GTIME$1,(1,X)NT,2X,(1,X)NT=0,TIME,TIME
:      G_S_T,G_S_T$(1,X)NT,(1,X),(1,X)NT=1,S_T,1,S_T
:      G_HI,G_HI$(1,X)NT,(1,X),(1,X)NT=1,HI_LIM,1,HI_LIM
:      G_LO,G_LO$(1,X)NT,(1,X),(1,X)NT=FIRST,LO_LIM,FIRST,LO_LIM
'GRAPH/HY,HX(1),TITL=HT,BV=YXSCALE' G_S_T,LO_LIM,HI_LIM;GTIME,TIME,TIME
$LPP; *,DASH,DASH
'VARI' ZERO=0.00002
'FACTOR' EXCLUDE$2,NT
'GROUP' EXCLUDE=LIMITS(TERM;ZERO)
'RESTRICT' LOG_S_T,TIMEW TEST,LOG_TIME$ EXCLUDE=NO
'GRAPH/HY(1),HX(1),TITL=HT(1)' LOG_S_T;TIME$ L
'GRAPH/HY(2),HX(2),TITL=HT(2)' W_TEST;LOG_TIME$ L
'JUMP' NO_PLOT *PLOTTER.ISNT.YES
'OUTPUT'2
'GRAPH/HY,HX(1),TITL=HT,BV=YXSCALE,BUFF=N,DEVICE=1'
     G_LO,G_S_T,G_HI;GTIME$ LLL
'GRAPH/HY(1),HX(1),TITL=HT(1),BUFF=N,DEVICE=1' LOG_S_T;TIME$ L
'GRAPH/HY(2),HX(2),TITL=HT(2),BUFF=N,DEVICE=1' W_TEST;LOG_TIME$ L
'OUTP/RECL=132'1
'LABEL' NO_PLOT
'RUN'
'CLOSE'
'STOP'
```

The diagnostic plot produced by program 1 may suggest that it might be worth fitting either an exponential or a Weibull survivor function to the data. Program 2 will fit an exponential distribution and estimate the parameter $\lambda$ (L in the program) by means of direct minimization of the negative log-likelihood function using 'OPTIMISE'. The likelihood, $L$, for $n$ observed deaths at times $t_1, t_2, \ldots, t_n$ and $m$ censored observations at times $u_1, u_2, \ldots, u_m$ is given by

$$L(t_1, t_2, \ldots, t_m, u_1, u_2, \ldots, u_m; \lambda) = \prod_{i=1}^{n} f(t_i) \prod_{i=1}^{m} S(u_i) \qquad (5)$$

where $S$ is given by equation (2) and $f$, the probability density function, by

$$f(t) = -\frac{dS(t)}{dt} = \lambda exp[-\lambda t] \qquad (6)$$

The initial value of $\lambda$ is found by taking the reciprocal of the mean of observed times of death. Steplengths are set to 2% of the initial value. The mean of the fitted distribution is printed and the program could easily be expanded to provide graphical output of the fitted survivor function for comparison with the Kaplan-Meier plot.

```
'REFE/NUNN=200,PRIN=IPF' PROGRAM_(2)
''
Analysis of survival data (lifetimes in days, including censored
observations)
.
fit of exponential survivor function
.
direct minimization of negative loglikelihood function
.
''
'HEAD' HDNG
'INPUT' 2
'READ' HDNG
'PRINT' HDNG
'READ/S,NUN=V' T,C
'INPUT' 1
'RUN'
'INTE' NO=2
'NAME' YESNO=yes,no
'FACTOR' REMOVED$YESNO,T
'GROUP' REMOVED=RANK(C)
'SCAL' MEAN_,L,LOG_LIKELIHOOD
'VARI' EXP_$T
'VARI' STEPL$1
'RESTRICT' T$ REMOVED= NO
'CALC' L=1/MEAN(T)
'RESTRICT' T
'CALC' STEPL=L*0.02
'MODEL' EXPON$ EXP_=EXP(-L*T)
            $LOG_LIKELIHOOD=-(SUM(C*LOG(L*EXP_))+SUM((1-C)*LOG(EXP_)))
'CAPT' ''
***** Fit of exponential distribution. *****''
'OPTI/PRIN=PSM,LIK=1,NPAR=1' EXPON; FMIN=LOG_LIKELIHOOD; PARAM=L;
STEPS=STEPL
'CALC' MEAN_=1/L
'CAPT' ''
***** Mean of fitted exponential distribution *****''
'PRINT/LABR=1' MEAN_$25.1
'RUN'
'CLOSE'
'STOP'
```

Program 3 will fit a Weibull distribution to the survival times. It is necessary to guess an initial value for the parameter $w$(`W` in the program). If the mortality rate, or hazard, is thought to decrease with time then $w$ should be in the range 0 to 1 and a guess of 0.5 should be satisfactory. If the hazard is increasing with time then $w > 1.0$ and a suitable first-try initial value might be 1.5; this could be increased if there is no convergence to a solution. The program calculates the mean of the uncensored data and uses this to find an initial value for $\lambda$ (`L` in the program). The mean, median and mode of the fitted distribution are printed.

```
'REFE/NUNN=200,PRIN=IPF' PROGRAM_(3)
''
Analysis of survival data (lifetimes in days, including censored
observations)
.
fit of Weibull survivor function
.
direct minimization of negative loglikelihood function
.
''
'MACRO' LGAM$
''
Macro to calculate log(complete gamma function)
''
'LOCAL' XX
'SCAL' XX
'CALC' XX=X+3*(X.LT.3)
'CALC'
LOGGAMMA=(XX-0.5)*LOG(XX)-XX+0.9189385+(0.0833333-0.00277778/(XX*XX))/XX
:       LOGGAMMA=LOGGAMMA-(X.LT.3)*LOG(X*(X+1)*(X+2))
'ENDMACRO/LOCAL=DESTROY'
'HEAD' HDNG
'INPUT' 2
'READ' HDNG
'PRINT' HDNG
'READ/S,NUN=V' T,C
'INPUT' 1
'RUN'
'INTE' NO=2
'NAME'              YESNO=yes,no
'FACTOR' REMOVED$YESNO,T
'GROUP' REMOVED=RANK(C)
'SCAL' X,MEAN_,MEDIAN_,MODE_,L,W,LOGGAMMA,LOG_LIKELIHOOD
'VARI' WEIB_$T
'VARI' STEPLW$2
''
Set initial value of parameter W
.
suggest 1.5 if hazard rate thought to increase with age
.
suggest 0.5 if hazard rate thought to decrease with age
''
'CALC' W=1.5
:       X=(W+1)/W
'USE/R' LGAM$
'RESTRICT' T$ REMOVED= NO
'CALC' L=EXP(LOGGAMMA)/MEAN(T)
'RESTRICT' T
'CALC' ELEM(STEPLW;1)=L*0.02
```

```
:       ELEM(STEPLW;2)=W*0.02
'MODEL' WEIBULL$ WEIB_=EXP(-((L*T)**W))
                 $ LOG_LIKELIHOOD=-(SUM(C*LOG(L*W*((L*T)**(W-1))*WEIB_))
                                   +SUM((1-C)*LOG(WEIB_)))
'CAPT' ''
***** Fit of Weibull distribution *****''
'OPTI/PRIN=PSM,LIK=1,NPAR=2' WEIBULL; FMIN=LOG_LIKELIHOOD; PARAM=L,W;
                                      STEPS=STEPLW
'LINE'2
'CAPT' ''
***** Mean of fitted Weibull distribution *****''
'CALC' X=(W+1)/W
'USE/R' LGAM$
'CALC' MEAN_=EXP(LOGGAMMA)/L
:       MEDIAN_=(LOG(2)**(1/W))/L
:       MODE_=(((W-1)/W)**W)/L
'PRINT/LABR=1' MEAN_$20.1
'CAPT' ''
***** Median of fitted Weibull distribution *****''
'PRINT/LABR=1' MEDIAN_$ 20.1
'CAPT' ''
***** Mode of fitted Weibull distribution *****''
'PRINT/LABR=1' MODE_$ 20.1
'RUN'
'CLOSE'
'STOP'
```

Some data may require a more complex probability model. If the diagnostic plot has the appearance of two intersecting straight lines, like a 'dog-leg' or 'broken stick', then a model involving mixtures of distributions could be appropriate. This could be a mixture of two exponential components, two Weibull components or an exponential and a Weibull. We can write this situation as

$$S(t)=pS_1(t)+(1-p)S_2(t) \tag{7}$$

where $p$ is a mixture parameter $0 < p < 1$, and $S_1$ and $S_2$ are the component survivor functions. For a mixture of two exponential distributions we would have

$$S(t) = p\,exp[-\lambda_1 t] + (1-p)\,exp[-\lambda_2 t] \tag{8}$$

In order to obtain initial parameter values for optimization, we assume that in the time up to the first death $S_2(t) = 1$ so that

$$S(t) = p\,exp[-\lambda_1 t] + (1-p) \tag{9}$$

The value for $\lambda_1$ is given by

$$\lambda_1 = \frac{(ln[\hat{S}(t_1) - 1+p] - 1n[p])}{t_1} \tag{10}$$

where $\hat{S}(t_1)$ is the Kaplan-Meier estimate at time $t_1$ calculated by equation (1) and we guess the value of $p$ either from prior knowledge or as 0.5. We also assume that, for large values of $S$,

$$S_1(t) = 0$$

so that

$$S(t) = (1-p)exp[-\lambda_2 t] \qquad (11)$$

Then the value for $\lambda_2$ is given by

$$\lambda_2 = \frac{ln[\hat{S}(t_j)] - ln[1-p]}{t_j}$$

where $\hat{S}(t_j)$ is the Kaplan-Meier estimate for some large value of time, $t_j$. It may be better to take an average of three values from the latter part of the sample distribution. The values of $\hat{S}$ can be obtained by running program 1. It is possible to make a reasonable guess at a suitable value of $p$ by studying the Kaplan-Meier plot and diagnostic plots. Initial values for $\lambda_1$ and $\lambda_2$ are calculated as above and edited into the program. Program 4 fits a double exponential model to survival data.

```
'REFE/NUNN=200,PRIN=IPF' PROGRAM (4)
''
Analysis of survival data (lifetimes in days, including censored
observations)
.
fit of binary mixture of exponential distributions
.
direct minimization of negative loglikelihood function
''
'HEAD' HDNG
'INPUT' 2
'READ' HDNG
'PRINT' HDNG
'READ/S,NUN=V' T,C
'INPUT' 1
'RUN'
''
set initial values for parameters
''
'SCAL' P=0.25  '' guess value for mixture parameter ''
:      L(1)=0.0492   '' L(1)=-(LOG(S_T+1+P)-LOG(P))/t1 for time t1 ''
:      L(2)=0.00584  '' L(2)=-(LOG(S_T)-LOG(1-P))/tj for some large time tj ''
:      LOG_LIKELIHOOD,MEAN_
'VARI' STEPPLL$3
:      UP_LIMS=1.0,1.0,1.0
:      LO_LIMS=0.0001,0.0001,0.0001
.      EXP_1,EXP_2,F_T $ T
'CALC' ELEM(STEPPLL;1)=P*0.02
:      ELEM(STEPPLL;2)=L(1)*0.02
:      ELEM(STEPPLL;3)=L(2)*0.02
'MODEL' DEXPON$ EXP_1=P*EXP(-L(1)*T)
              $ EXP_2=(1-P)*EXP(-L(2)*T)
              $ F_T=L(1)*EXP 1+L(2)*EXP_2
              $ LOG_LIKELIHOOD=-(SUM(C*LOG(F_T))
                              +SUM((1-C)*LOG(EXP_1+EXP_2)))
'CAPT' ''
*****  Fit of double exponential distribution  *****''
```

```
'OPTI/PRIN=PSM,LIK=1,NPAR=3' DEXPON; FMIN=LOG_LIKELIHOOD;
PARAM=P,L(1,2); STEPS=STEPPLL; UPPER=UP_LIMS; LOWER=LO_LIMS
'CALC' MEAN_=P/L(1)+(1-P)/L(2)
'CAPT' ''
***** Mean of fitted double exponential distribution *****''
'PRINT/LABR=1' MEAN_$25.1
'RUN'
'CLOSE'
'STOP'
```

A double Weibull or mixed exponential and Weibull could be fitted in a similar way. Elandt-Johnson and Johnson (1980), chapter 7 provides a good discussion of fitting mixtures.

Grids of likelihood can be obtained using the GRID option of 'OPTIMISE', and these can be plotted in one dimension by 'GRAPH' or two dimensions by 'CONTOUR'. In addition to the parameter estimates, Genstat provides the square root of the second derivatives and a matrix of scaled second derivatives of the log-likelihood function, with respect to the parameters, at the optimum. These are, asymptotically, the standard errors and correlations among the parameter estimates but it may be necessary to have a fairly large sample, perhaps some hundreds of observations, before the second derivatives can be expected to behave well as standard errors and correlations.

In this article I have attempted to sketch some of the theory involved in fitting exponential and Weibull distributions to survival data. The reader is referred to Kalbfleisch and Prentice (1980) and Elandt-Johnson and Johnson (1980) for a full discussion of the relevant theory. The programs should provide at least a basic 'tool-kit' for analysis of survival data.

## References

[1] Elandt-Johnson, R.C. and Johnson, N.L.
*Survival Models and Data Analysis.*
John Wiley & Sons Inc., New York, 1980.

[2] Kalbfleish, J.D. and Prentice, R.L.
*The Statistical Analysis of Failure Time Data.*
John Wiley & Sons Inc., New York, 1980.

# Genstat by Post

*J. Riley*
*Overseas Development Administration Biometrician*
*Statistics Department*
*Rothamsted Experimental Station*
*Harpenden*
*Hertfordshire AL5 2JQ*
*United Kingdom*

I am one of the members of the Overseas Unit of Rothamsted Statistics Department. Here I describe some of the difficulties of collaboration by post and how they affect efficient programming techniques. I also describe some typical analyses, peculiar to overseas work, which can be programmed in Genstat.

Much of the data which I receive for analysis arrives in a form far from the ideal one that I constantly suggest to my clients. The reasons for this are several: (1) I have often been introduced to an overseas research team long after the data were collected; (2) the fast turn-over of contract staff in overseas teams means that a team will need continual training in data collection – training which is difficult to arrange over many thousands of miles; (3) because of limited numbers of trained staff in the area, a research team may often depend upon untrained local people to help at harvest time; and (4) as experiments from a great variety of countries can rarely follow any typical layout, it is difficult to influence a researcher thousands of miles away before he begins recording data from a trial.

To render the data into a form suitable for analysis may take more time than the analysis itself. Here, I give a few examples of the sort of data organisation required and one or two typical features of analyses that I have carried out, using Genstat, for such distant research teams.

## Organisation of Data

Untidy, and often illegible, data sheets are returned to their origins if they really are so bad that data processor and statistician find them too difficult to work with. On one occasion, however, I knew that the very untidy data sheets were the only ones available, the originals having been destroyed through no fault of the researchers. The experiment involved different age-groups containing about 50 sheep, which were weighed and scored for condition seven times a year, while fleece weights and lamb data were recorded once. Unfortunately, the data, collected long before a statistician arrived on the scene, were recorded by a different person and on different sheets of paper at each date. The seven weights and condition scores for any one group of sheep could not be aligned sufficiently well to permit a parallel 'READ' statement for all the variates and so the data had to be input one date at a time and a program written to form a new data file holding all the variates in parallel for each group of sheep. This less cumbersome file was much easier to handle at the analysis stage and a print-out of the file enabled the researcher to follow the progress of any one sheep throughout the whole year.

The next aim was to present the lamb variates in parallel with the sheep variates. This should have involved the simple step of reading the lamb data and printing them in parallel with the parent sheep data. However, the researchers had recorded the data in the order in which the lambs had arrived for weighing, which was not the same order as their parents' data. Furthermore, a lamb's number had no connection with its parent's number. Thus, for example, sheep number 12 had lamb number 123, while sheep number 13 had lamb number 105. Hence it was necessary to read in an extra variate of sheep numbers, with the lamb numbers, to allocate the correct offspring to their parents. The ORDER function could then be used to sort all the lamb data according to the sheep numbering so that the lamb data could be printed in parallel with the parents' weights and condition scores. Thus

the randomised sheep numbers (SHEEP) could be read in parallel with the lamb numbers (LAMB)and lamb variates (V(1,2,3)) to link up a lamb with its mother as follows:

```
'Read'   LAMB,   V(1),   V(2),   V(3),   SHEEP   'Run'
         101     8       0.1     10      5
         100     2       0.6     11      2
         105     3       0.9     12      6
         103     9       0.2     13      1
         102     6       0.4     14      4
         104     4       0.5     15      3
'EOD'
```

Then a few Genstat statements of the form:

```
'SET'  LAMBVARS = LAMB, V(1, 2, 3)
'CALC' LAMBVARS = ORDER (LAMBVARS; SHEEP)
    :  SHEEP = ORDER (SHEEP)


'PRIN/P' SHEEP, LAMBVARS $ 3(10), 10.1, 10
```

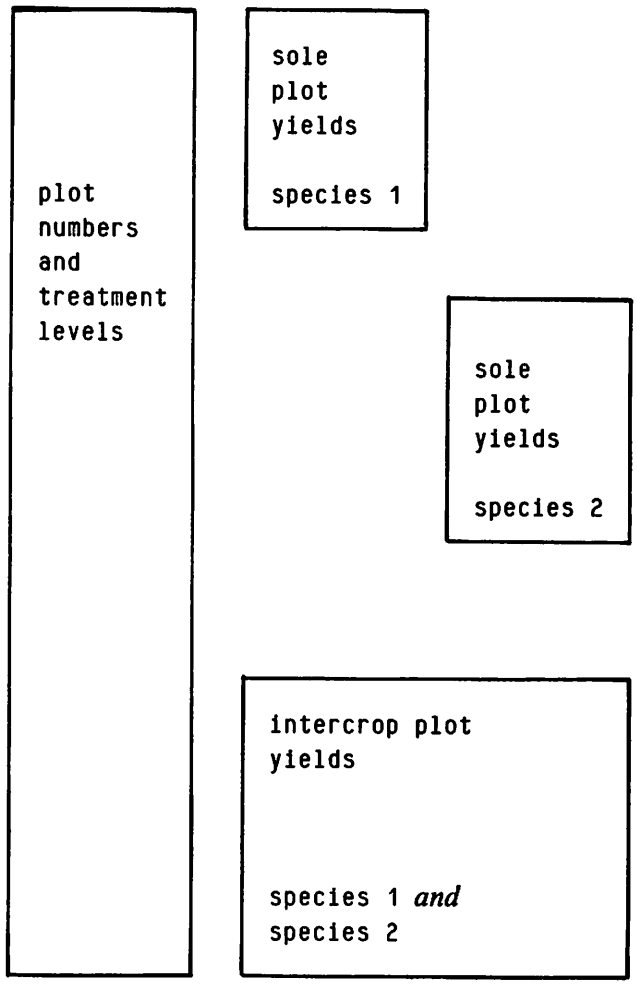
would produce the lamb data in the same order as the ordered sheep data:

| SHEEP | LAMB | V(1) | V(2) | V(3) |
|---|---|---|---|---|
| 1 | 103 | 9 | 0.2 | 13 |
| 2 | 100 | 2 | 0.6 | 11 |
| 3 | 104 | 4 | 0.5 | 15 |
| 4 | 102 | 6 | 0.4 | 14 |
| 5 | 101 | 8 | 0.1 | 10 |
| 6 | 105 | 3 | 0.9 | 12 |

Thus, short and simple programming is all that is required to deal with what would otherwise be a formidable task of data organisation.

## Intercropping Experiments

Experiments in tropical countries are often performed to investigate intercrops, that is, a mixture of two completely different species. A typical intercropping experiment consists of a number of intercropped plots, on which species 1 and species 2 are grown together, each plot receiving one of the treatments under study. To provide a comparison between the yields of species 1 and 2 when intercropped and the yields that could be obtained when the species are grown sole, the experiment may also contain plots of species 1 alone and plots of species 2 alone. These sole-cropped plots may be positioned next to the intercropped plots or they may be randomised amongst the intercropped plots. When analysing data from such a trial, it is important to distinguish between the sole and intercropped plots and to deal with them separately, since variation within the data for the two individual systems of planting can be expected to be quite different. The ideal way to present such data, then, is to give all the yields of the sole plots of species 1, separate from the yields of the sole plots of species 2, with both of these blocks of data followed by all the yields from the intercropped plots. A suitable layout may be of the following form:




More often than not, the sole plots have been randomised amongst the intercrop plots and the data, recorded in plot order, consist of variates whose length is the total number of plots in the experiment. Thus, for a sesame/sorghum intercropping trial, the plant heights in centimetres were recorded as follows for one of the four replicates.

| *Plot* | *Treatment* | *Sorghum height* | *Sesame height* |
|---|---|---|---|
| 1 | 1 | 159 | 189 |
| 2 | *sole sesame* | * | 176 |
| 3 | 2 | 162 | 184 |
| 4 | 9 | 181 | 171 |
| 5 | 3 | 166 | 170 |
| 6 | 7 | 162 | 165 |
| 7 | 8 | 182 | 170 |
| 8 | 5 | 161 | 183 |
| 9 | *sole sesame* | * | 180 |
| 10 | *sole sorghum* | 163 | * |
| 11 | 4 | 167 | 187 |
| 12 | *sole sesame* | * | 174 |
| 13 | 6 | 158 | 156 |

The setting-up of a factor REST to distinguish between the sole and mixed plots and the use of 'RESTRICT' permit calculations to be done on the mixed plots as follows:

```
'FACTOR' REST $ 2 = 2, 1, 6(2), 2(1), 2, 1, 2
'RESTRICT' SORGHUM, SESAME $ REST = 2
```

However, the accurate use of 'EQUATE' is preferred to produce the shorter variates for the separate cropping systems so that the usual range of analyses for such data can be applied effortlessly and efficiently. Thus,

```
'VARIATE' SOLESES $ 3
        : SORGHT, SESHT $ 9

'EQUATE' SOLESES = SESAME $ 1X, 1, 6X, 1, 2X, 1, 1X
        : SORGHT, SESHT = SORGHUM, SESAME $ 1, 1X, 6, 2X, 1, 1X, 1
```

The variate SOLESES will then hold the three sole sesame values and the variates SORGHT and SESHT will hold the nine sorghum and sesame heights respectively: these can then be used for the usual range of analyses for intercropping data.

One of the recognised analyses for two correlated variates $V1$ and $V2$ from intercropped plots is the Bivariate method (Pearce and Gilliver, 1978). The error variances of the yields $x_1$ and $x_2$ of the two crops are $V_{11}$ and $V_{22}$ and their error covariance is $V_{12}$. After each variate has been adjusted for the other, as in analysis of covariance, the variances become $V'_{11}$ and $V'_{22}$ where

$$V'_{11} = V_{11} - V_{12}^2/V_{22}$$

and

$$V'_{22} = V_{22} - V_{12}^2/V_{11}$$

Two new variates, $y_1$ and $y_2$, whose means can be plotted with rectangular axes, are defined as

$$y_1 = x_1/\sqrt{V'_{11}}$$

and

$$y_2 = (x_2 - V_{12}x_1/V_{11})/\sqrt{V'_{22}}$$

having error variances equal to 1 and error covariance equal to 0, i.e. $y_1$ and $y_2$ are independent. Noting that the covariance of $x_1$ and $x_2$ is one quarter of the variance of $(V1+V2)$ minus one quarter of the variance of $(V1-V2)$, to program this method is very easy using the 'EXTRACT' directive in Genstat. For an experiment involving four millet genotypes (MILLET) and four sorghum genotypes (SORGHUM) grown in all sixteen combinations in four replicate blocks of sixteen whole plots (WP), the Genstat instructions would be:

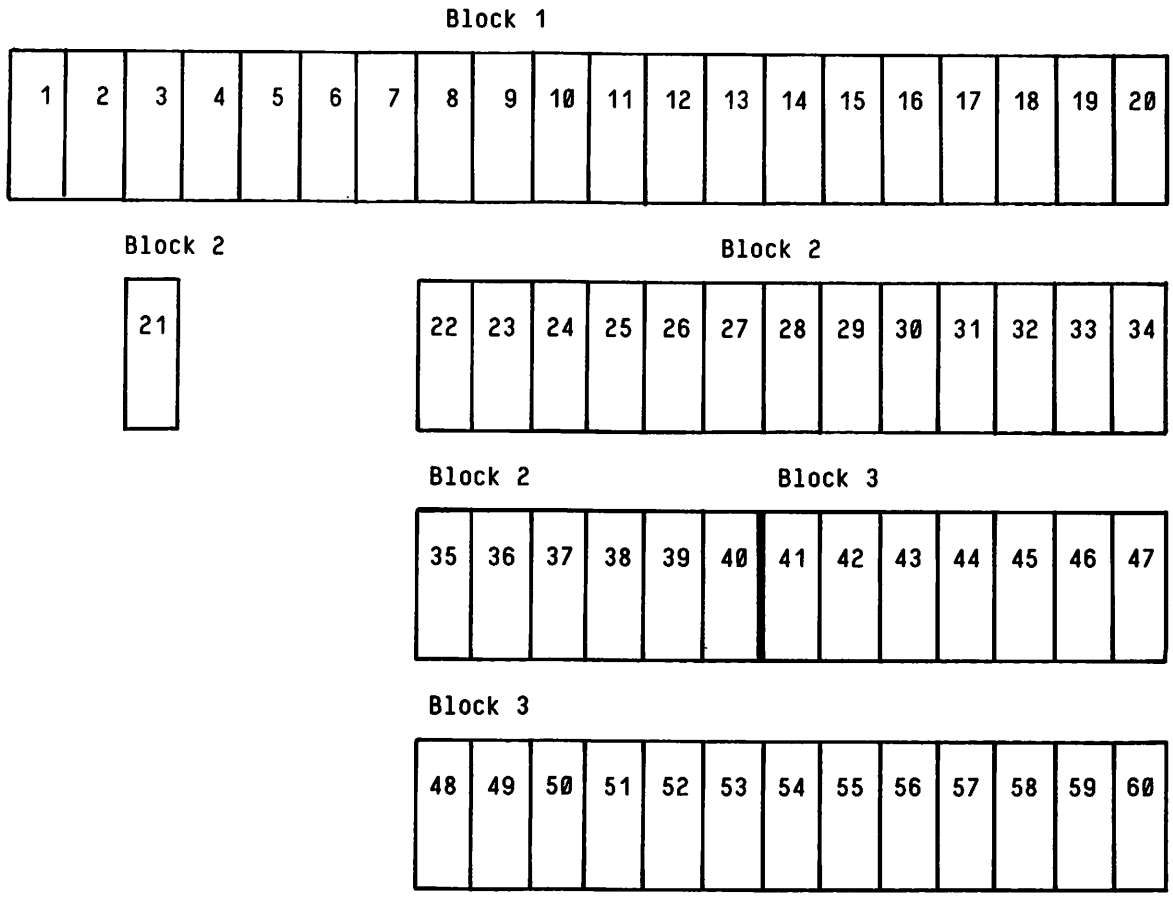
```
'CALC' TOTAL = X1 + X2
     : DIF = X1 - X2

'BLOCK' BLOCK/WP
'TREA' MILLET * SORGHUM

'ANOV' X1; OUT = MOUT
     : X2; OUT = SOUT
     : TOTAL; OUT = TOUT
```

```
      : DIF; OUT = DOUT

'EXTRACT' MOUT; MILLET * SORGHUM $ VAR = V11
        : SOUT; MILLET * SORGHUM $ VAR = V22
        : TOUT; MILLET * SORGHUM $ VAR = VT2
        : DOUT; MILLET * SORGHUM $ VAR = VD2

'SCALAR' V12, SV11, TV22, SV22

'CALC' V12 = (VT2 - VD2)/4
     : TV11 = V11 - (V12 * V12/V22)
     : TV22 = V22 - (V12 * V12/V11)
     : SV11, SV22 = SQRT(TV11, TV22)
     : Y1 = X1/SV11
     : Y2 = (X2 - V12 * X1/V11)/SV22

'ANOV' Y1 $ OUT = Y1OUT
     : Y2 $ OUT = Y2OUT

'EXTRACT' Y1OUT; MILLET * SORGHUM $ MEAN = MM1, MS1, MY1
        : Y2OUT; MILLET * SORGHUM $ MEAN = MM2, MS2, MY2

'VARI' VMY1, VMY2 $ 16

'EQUATE' VMY1 = MY1
       : VMY2 = MY2

'GRAPH' VMY2; VMY1
```

Thus the mean values for the sixteen genotype combinations can be calculated and plotted for the new independent variates, $Y_1$ and $Y_2$.

## Residuals in Field Layout

Experiments done in developing countries are often on land which is being cultivated for the first time; little is known of its past history and the experimental area is likely to be very variable in quality. Accurate positioning of the plots is thus necessary to ensure homogeneity within a block and this can lead to very irregularly-shaped experiments. The printing in field layout of plot residuals from an analysis is a useful step that can often indicate trends and extreme values which, in turn, may suggest that a further, modified, analysis is necessary. A table with factors whose numbers of levels equals the number of rows and columns of plots in the design can be used to hold the residuals; printing the table, with row and column labels suppressed, will provide a presentation of the residuals in field layout. For regularly laid-out experiments with contiguous or parallel blocks, this procedure is very simple. For the irregularly laid-out trials with which I am often concerned, the setting-up of the table is not difficult but requires a little more care to ensure that the output given truly represents the field layout. One such set of data which I received from a trial on different grass species consisted of three blocks of twenty plots arranged in the following way:




with plots 1 to 20 in Block 1, 21 to 40 in Block 2 and 41 to 60 in Block 3. The Genstat instructions to form a table to hold the residuals and then to print them out are as follows:

```
'FACT' X $ 4 = 20(1), 14(2), 13(3, 4)
     :Y $ 20 = 1 ... 20, 3, (8, 9, ... 20)3
'TABL' TAB $ X, Y
'ANOV' DATA; RES = RR
'TABU' RR; TAB
'PRIN/LABR = 1, LABC = 1' TAB $ 6.1
```

One point which should be made about the presentation of results of analyses to clients overseas is that results need to be very clear so that parts of an analysis can be referred to easily in a covering letter. To achieve this, I use plenty of headings and comments in the program; to make a point more clearly I may also use graphs and histograms. These may illustrate something fairly obvious in the data but, by doing so, may make the results clearer to a client who is not very comfortable with statistical analysis or jargon. I hope that with careful use of the Genstat output facilities I get my message across; if not, then I simply write some more on the computer output!

## Acknowledgement

The author is funded by the U.K. Overseas Development Administration.

## Reference

[1] Pearce, S.C. and Gilliver, B.
The statistical analysis of data from intercropping experiments.
*J. Agric. Sci.*, Camb., **91**, 625-632, 1978.

# Understanding Common Error Messages in Genstat

*R. Gough*
*East Malling Research Station*
*East Malling*
*Maidstone*
*Kent ME19 6BJ*
*United Kingdom*

'The error messages generated by the system provide users who are learning it with many problems. What they need is a fuller explanation with more examples', said Helen Talbot of Edinburgh Regional Computing Centre, at the 1983 Genstat Conference in Paris. A J Weeks, of the University of York, was of a similar opinion and saw the beginner's difficulty as '...deciding exactly what has caused a message such as:

```
Line 10   Statement 0   Fault VA 4
```

Both were expressing the feelings of many new and experienced Genstat users who know the problem of finding their program errors.

The users at East Malling were no exception and I realised that illustrations were needed of situations where errors can occur. As a result, a simple method of collating these illustrations was introduced some five years ago, recording examples of error messages in a loose-leaf folder – one error message per page. The next step was, for several months, to advertise, within the research station, for users to supply me with their error messages. The response was good, despite the fact that not everyone likes to admit to having an error. Soon I had collected examples of likely situations for the occurrence of 43 of the more common Genstat errors. Reference to these examples has saved much time in the search for the causes of errors.

In 1984 we had a new challenge – a VAX 11/750 running Genstat 4.04. The immediate problem of discovering the meaning of a fault such as

```
VA 4  ,  SX 6
```

had now been overcome by a one-line explanation following the fault message in the output file. However, there is still a need to refer to Chapter 11 of the Genstat Manual Part I for a fuller explanation, or to the loose-leaf folder of examples.

The next obvious step was to transfer the information from the loose-leaf folder to individual VAX computer files, one file per error message. This enabled the users to display, on the vdu screen, not only the one-line message but also the fuller version, followed by further examples – where available – of likely situations. (Of the possible 192 errors, 125 have now been entered in computer files.)

A print-out of the 43 more common error messages will be available at the 1985 Genstat Conference at York. If any reader who is unable to attend the conference would like a copy, I should be pleased to supply one.

I should also be pleased to receive examples of situations causing Genstat errors, so that I may add these to my collection. To illustrate these common error messages I have listed three below, namely `IO 11`, `SP 1` and `SX 10`.

## IO 11 Errors in data values

This comment introduces the detailed list of error reports on individual items.

**Examples**

(1) The following has been extracted from an output file where fault IO 11 had been reported.

```
******** *   LINE   20  STATEMENT  1  FAULT     IO 11

UNIT   V/SN     NCH
   4     11       0      FAULT   SX  8
```

In this example the error was in the 11th variate in a parallel READ statement and the 4th unit of the 11th variate contained a letter O instead of a zero.

(2) `'READ/P'  V(1...30) $ S`  IO 11 reported

solution either :- `'READ/P' V(1...30) $ S ,30`

or :- `'READ/P' V(1...30)`  (omit S altogether)

## SP 1 Directory full

Too many named or unnamed structures – change REFE options.

The number of named or unnamed structures has exceeded the limits set (or implied) by the REFERENCE statement [Genstat manual, Pt II:2.1].

The appropriate limit should be increased, judiciously, or DEVALUE used to remove unnamed structures no longer needed in the current block.

Default:- Number of IDentifiers NID = 100
Number of UNknowN identifiers NUNN = 50

To find how many NIDs or NUNNs have been used, look in the output file for the string:-

```
*** STRUCTURES
```

or

```
NAMED
```

The area you need appears as follows:-

```
*** STRUCTURES
          NAMED UNNAMED
NO. USED     99       5
 MAXIMUM    100      50
```

In the above example, the default number of NIDs is about to be exceeded, but only 5 out of 50 NUNNs have been used. Therefore, before the job is run again, the REFE line should be amended to:-

```
'REFE/NID=200' PROG
```

## SX 10 Incompatible adjacent elements

This very common message has a variety of causes. For example:

(a) two identifiers not separated by a comma
(b) identifier beginning with a number
(c) first number missing from a progression
(d) directive name missing.

**Examples**

(1) Check factor names; you cannot have parentheses ( ) or dashes - in a factor name.

(2) `'NAME'  NL= -,*,'` (factor levels to define symbols for plotting)

A prime cannot be used in a name list.

(3) `'EQUA' V4(1...3)=V(4) $ ((7,1_4X)3,7X)3`

Space within an item (the integer 14) is not allowed.

(4) Check that you are not using a reserved word, e.g. `MAX`, `DIFF`. See the Genstat manual, Pt II Appendix 4, for a complete list of reserved words.

(5) Identifiers must begin with an 'alphabetic' character (`A` to `Z`, `%`, `_`); e.g. `'REFE' 1985BUDS` is not allowed, but `BUDS1985` is permissible.

(6) `'SET/F' F(1)=4,28X`

```
'EQUA'    V(1...7) = DATA $ F(1)
```

`F`, `S` and `X` cannot be used as identifiers in a `SET/F` directive, nor can any subscripted version be used, e.g. `F(1)`, `S(44)`, `X(2)`.

## Drawing Bar Charts

*N.G. Alvey*
*23 Tuffnells Way*
*Harpenden*
*Hertfordshire AL5 3HJ*
*United Kingdom*

### Outline of the Problem

Up to release 4.04 of Genstat, it was possible to use the `GRAPH` directive to draw histograms as bar charts. As it is now possible to draw histograms using the `HISTOGRAM` directive, the additional code for drawing histograms using `GRAPH` was omitted in release 4.04. This macro has been written for those still wishing to produce bar charts.

### Method of Use

The user must create a factor `F` with boundary values in a variate `L`. This could be done using the `LIMITS` function of the `GROUP` directive. He must also set up 3 scalars:

`NL` = number of levels of `F` which is the number of values of `L` plus 1.
(This may be `CALCULATE`d using the function `NLEV`.)
`NL1` = `NL-1`
`NL2` = `NL-2`

Two headings `YTITLE` and `XTITLE` must be declared as titles for the *Y* and *X* axes respectively. After a `RUN` statement the statement

```
'USE' BARCHART $
```

must appear in a separate block of instructions.

**Global Identifiers**

| | | | | |
|---|---|---|---|---|
| FACTOR | F | input | : | factor identifier |
| VARIATE | L | input | : | variate of group limits |
| SCALAR | NL | input | : | no. of levels of F |
| SCALAR | NL1 | input | : | NL-1 |
| SCALAR | NL2 | input | : | NL-2 |
| HEADING | YTITLE | input | : | Y-axis title |
| HEADING | XTITLE | input | : | X-axis title |

The macro uses another macro called APPEND at run-time.

## Print of BARCHART and APPEND

```
'MACRO'     BARCHART $
'LOCAL'     S(1...NL),T(1...NL),R(1...NL),I,J,DUMR,DUMS,DUMT,IREST,BVAL,
            X,Y,XAPP,YAPP,STEP,MAXY
'START'
'SCALAR'    S(1...NL) : T(1...NL) : R(1...NL) : STEP,MAXY
'CALC'      S(1) = ELEM(L;1) - (ELEM(L;2) - ELEM(L;1)) / 2
'FOR'       DUMS = S(2...NL1) ; I = 2...NL1 ; J = 1...NL2
'CALC'      DUMS = (ELEM(L;I) + ELEM(L;J)) / 2
'REPEAT'
'CALC'      S(NL) = ELEM(L;NL1) + (ELEM(L;NL1) - ELEM(L;NL2)) / 2
'FOR'       I = 1...NL ; DUMT = T(1...NL)
'RESTRICT'  F $ F=I ; IREST
'CALC'      DUMT = NVAL(F)
'REPEAT'
'RUN'
'START'
'VARIATE'   T =T(1...NL)
'CALCULATE' MAXY = MAX(T)
  :           STEP = MAXY/40
'RUN'
'VARIATE'   Y = 0,STEP...T(1)
'CALC'      R(1) = NVAL(Y)
'RUN'
'VARIATE'   X = (S(1))R(1)
'FOR'       DUMS = S(2...NL) ; DUMT = T(2...NL) ; DUMR = R(2...NL)
'USE/R'     APPEND $
'REPEAT'
'VARI'      BVAL=0,*,*,*
'GRAPH/ATY=YTITLE,ATX=XTITLE,BV=BVAL,NRF=41'    Y ; X
'ENDMACRO'
```

```
'MACRO'    APPEND $
'START'
'VARIATE'  YAPP = 0,STEP...DUMT
'CALC'     DUMR = NVAL(YAPP)
'RUN'
'VARIATE'  XAPP = (DUMS)DUMR
'VARIATE'  Y=Y,YAPP : X=X,XAPP
'ENDMACRO'
```

**Test Program**

```
'REFE/NUNN=30'     HISTO
'HEAD'    YTITLE=''Y AXIS TITLE'' : XTITLE=''X AXIS TITLE''
'SCALAR'  NL,NL1,NL2
'VARIATE' V=(1...5)10,8(6...10),(11...19)10,5(16...20) : L=4,6...18
'GROUPS'  F=LIMITS(V;L)
'CALC'    NL=NLEV(F) : NL1=NL-1 : NL2=NL-2
'RUN'
'USE'     BARCHART $
'RUN'
'CLOSE'
'STOP'
```

# Linking Fortran Subprograms into Genstat

*P.W. Lane*
*P.G.N. Digby*
*Statistics Department*
*Rothamsted Experimental Station*
*Harpenden*
*Hertfordshire AL5 2JQ*
*United Kingdom*

This article is intended to help people who want to link specialised programs written in Fortran into Genstat release 4.04. It provides the details about the internal workings of Genstat necessary for straightforward applications. For more complicated tasks, it may be necessary to refer to the Genstat Implementor's Manual, but this is designed primarily for the people responsible for maintaining the original Genstat source code. The Manual can be obtained by writing to the Genstat Secretary, at the above address.

We assume here that you are familiar with writing programs in Fortran and with the Genstat command language. You will also need to know how to link Fortran programs, using the operating system on your computer: the process is illustrated here for VAX/VMS only.

## A Directive for Special Tasks

A directive called `'OWN'` is provided in Genstat to allow users to call their own Fortran subprograms. In the distributed form of Genstat, an `'OWN'` command will cause no action, since its use invokes a call to a null subprogram:

```
SUBROUTINE OWN
RETURN
END
```

You can replace this subroutine with one which calls your own subprograms; the Genstat system must then be relinked, including all the subprograms you need. You can use the resulting modified Genstat system in the same way as the standard system, except that whenever you give it an `'OWN'` command, it will do whatever operations are specified by your `OWN` subroutine.

### The Syntax of the 'OWN' Directive

In Genstat 4.04, the `'OWN'` directive has no options and you may specify only one list. (In the article by Bouvier in this Newsletter, the Genstat compiler has been modified to provide options for the directive `'OWN'`.) The list may include non-negative numbers, identifiers and missing values.

**Examples**

```
'OWN'
'OWN' 2
'OWN' Series, Result
'OWN' 4, Series, *, Out(1...3)
```

Do not include negative numbers, as these will be confused with structure references. All numbers are rounded to integers, so use identifiers of scalar structures if you want individual real numbers or negative integers.

You can arrange for the `'OWN'` directive to be able to do many different things if you want. The easiest way is to reserve the first value in the list as a switch, used internally to control which subprograms are invoked. Then

`'OWN' 1, ...` causes one type of action,

'OWN' 2, ... causes another, and so on.

You can put 'OWN' statements anywhere in a Genstat program, but remember that the OWN subroutine will not be called until execution time, i.e. all statements after an 'OWN' statement but before the next 'RUN' statement will be compiled before the 'OWN' statement is executed.

**Relinking Genstat**

The method of relinking Genstat depends on the operating system but the following example shows what is involved. We illustrate the process for the VMS operating system for VAX computers, in which the linking instructions are particularly simple.

The Genstat system as distributed includes the object code (produced by the Fortran compiler) in an object library. Say this library has the filename:

```
[GENSTAT]GNLIB.OLB
```

The main module of Genstat and the block data module will be separate from this library; say they have filenames:

```
[GENSTAT] MAIN.OBJ and BD.OBJ
```

Assume that your version of subroutine OWN calls a single subroutine called WORK and the compiled versions (object modules) of these subroutines have filenames:

```
[MYFILES] OWN.OBJ and WORK.OBJ
```

Then, working in the directory MYFILES, you can form a new version of Genstat, called MYGENSTAT.EXE say, by giving the following DCL command:

```
$ LINK/EXEC=MYGENSTAT  OWN, WORK, [GENSTAT]MAIN, BD, GNLIB/LIBRARY
```

You should run the new system by setting up a 'foreign' command. Assuming that the directory MYFILES is on device DRA01, a command MYGEN can be defined as follows:

```
$ MYGEN :== '$DRA01:[MYFILES]MYGENSTAT.EXE'
```

You can then use the command MYGEN in precisely the same way as the usual command GENSTAT.

## Storage in the Genstat System

Before you can implement the 'OWN' directive you will probably need to know some details of the internal workings of Genstat. Genstat release 4.04 is written in Fortran 66, but is generally compiled by Fortran 77 compilers. Some extensions to the ISO standard are used: refer to the Implementor's Manual, Section 1.1, if you need details.

The major area for storage is in blank common and consists of six arrays which are equivalenced to each other:

```
DATA, CDATA, RDATA, IDATA, ISDATA, TABLE
```

They are used for storing long reals, words (i.e. 8 characters stored in a long real), reals, integers, short integers, and again short integers, respectively. These arrays are usually given the dummy dimension of 1: the full dimension is set in the main routine of Genstat.

```
      COMMON          DATA(1)
      DIMENSION       CDATA(1),RDATA(1),IDATA(1),ISDATA(1),TABLE(1)
      DOUBLE PRECISION  DATA,CDATA
      REAL            RDATA
      INTEGER         IDATA
      INTEGER *2      ISDATA,TABLE
C
```

```
      EQUIVALENCE   (DATA(1),CDATA(1),RDATA(1),IDATA(1),
     1               ISDATA(1),TABLE(1))
```

On some computers, short integers are not supported and so the arrays ISDATA and TABLE should be declared as INTEGER, like IDATA; also, it may not be allowed to use a dummy dimension. You can copy the correct version of the common from the Genstat code, module MAIN.

Communication between different parts of the Genstat system is done via a series of named common blocks. Two of these are /DATAC/, for data accessing, and /SYSCON/, which holds system constants, e.g. the value used in Genstat to represent missing values in variates. You will almost certainly need to use these commons so you must include them in the OWN subroutine. You can copy these two commons from the block data module called BD, but they are also listed below.

```
      COMMON/DATAC/ IDENT(3),ATTOR(3),TYPE(3),VALOR(3),NVAL(3),MODE(3),
     1   VECNO(3),DESC(3),MVPTR(3),ACC(3),SET(3),UNIVEC(3),STAVEC(3),
     2   SUBCLS(3),NIND(3),NUN(3),NSV(3),SPEC1(3),SPEC2(3),SPEC3(3),
     3   IDSUF(3),ENDDAT,LATT,LSTDAT,MAXDAT,PCI
      DOUBLE PRECISION  IDENT
      INTEGER       ACC,ADDPTR(3),ATT(57),ATTOR,BMINW(3),DESC,DIAVEC(3),
     1   ENDDAT,FSET(3),IDIV(3),IDSUF,KROOTS(3),LATT,LNVEC(3),LSTDAT,
     2   MARG(3),MATVEC(3),MAXDAT,MODE,MVPTR,NCOL(3),NIND,NLEV(3),
     3   NRC(3),NROW(3),NSV,NUMEX(3),NUN,NVAL,NVAR(3),PCI,SCALNO(3),
     4   SDMVEC(3),SET,SETC(3),SETR(3),SPEC1,SPEC2,SPEC3,STAVEC,SUBCLS,
     5   TABVEC(3),TYPE,UNIVEC,USET(3),VALOR,VECNO,VSET(3),VTYPE(3),
     6   WTVAR(3)
      COMMON/SYSCON/DBLANK,DEPS,DMV,DTOL,EPS,MRSI,RMV,TOL,
     1              IMV,MASK,NBB,
     2              ISMV,ISZERO,MSI,NBI,NBV(5),NVDR(5),NIR,NSII,NSIR
      DOUBLE PRECISION DBLANK,DEPS,DMV,DTOL
      REAL          BLANK,EPS,MRSI,RMV,TOL
      INTEGER       IBLANK,IMV,MASK,NBB
      INTEGER *2    ISBLNK,ISMV,ISZERO,MSI,NBI,NIR,NSII,NSIR,
     1              NBV,NVDR,NIDR,NRDR,NSIDR,NSWDR
C
      EQUIVALENCE   (ATT(1),ATTOR(1)),
     1   (SET(1),LNVEC(1),NUMEX(1),SDMVEC(1),SETR(1),VTYPE(1),WTVAR(1)),
     2   (UNIVEC(1),NLEV(1),NVAR(1),SETC(1),TABVEC(1)),
     3   (STAVEC(1),IDIV(1),KROOTS(1),NRC(1)),
     4   (SUBCLS(1),ADDPTR(1),BMINW(1),MARG(1),NROW(1),VSET(1)),
     5   (NIND(1),FSET(1),MATVEC(1),NCOL(1),USET(1)),
     6   (NUN(1),DIAVEC(1)),(NSV(1),SCALNO(1))
C
      EQUIVALENCE   (DBLANK,BLANK,IBLANK,ISBLNK),
     1              (NSWDR,NSIDR,NVDR(4)),(NIDR,NVDR(3)),(NRDR,NVDR(2))
```

You may also need to access some of the other system commons, e.g. /DIAGPK/ which holds information about Genstat diagnostics, and /WSP/ which contains details of the allocation of workspace. These are mentioned below, under **Some Extra Details.**

On some computers, the Fortran compiler insists that all common blocks are described before any equivalences. If that is the case on your computer, you will have to put all the COMMON statements first and then their respective EQUIVALENCE statements; this has been done with /DATAC/ and /SYSCON/, above.

## Writing an Interpreter for 'OWN'

As well as including the standard common blocks described above, your OWN subroutine must be able to interpret the compiled form of the 'OWN' statement. This may well involve accessing and storing values in Genstat structures, such as variates: it is probably easiest if you put all the accessing etc. in the OWN subroutine.

### Compiled Form of an 'OWN' Statement

The Genstat interpreter compiles an 'OWN' statement into a coded form. This is stored in the array ISDATA, in blank common, and the first value of the coded form is in ISDATA(PCI), where PCI is a variable in common /DATAC/. This first value is the number of options: it is therefore always 0. The second value is the number of lists: it is therefore 1. The third value is the length of the list, i.e. the number of items in the 'OWN' statement. Following the length, if it is not zero, are the coded forms of each item in the list: numbers are replaced by the nearest integer, * is replaced by a large negative value, which is equal to variable ISMV in common /SYSCON/, and identifiers are replaced by negative integers which are reference numbers in the directory of identifiers.

### Examples

```
                                        ISDATA(PCI)
'OWN'                                      0,1,0
'OWN' 2                                    0,1,1,2
'OWN' Series, Result                       0,1,2,-2,-7
'OWN' 4, Series, *, Out(1...3)             0,1,6,4,-2,ISMV,-8,-9,-10
```

### Accessing Data Structures

If you want to include identifiers in an 'OWN' directive, you must be able to access the structures they identify and pass the relevant information on to your Fortran subprograms. To access all the information about a structure with reference number N (negative), give the following statement in the OWN subroutine:

```
I = GETATT(1, N)
```

GETATT is a Fortran short integer function, included in the Genstat source code. You must declare the type of this function in your OWN subroutine (leave out the *2 if your computer does not have short integers):

```
INTEGER *2 GETATT
```

The function GETATT returns the value 0 unless there is an invalid value in one of the arguments, e.g. N is not a current reference number. Therefore, you should test the value of I after the call above and take some action if it is non-zero, e.g.

```
IF (I .NE. 0) RETURN
```

In the standard Genstat code, this operation is usually done in one step:

```
IF (GETATT(1, N) .NE. 0) RETURN
```

Instead of just exiting from subroutine OWN after such a mistake, you may want to print a diagnostic message: see below for more details.

GETATT has two arguments: the first must be 1, 2 or 3 and specifies in which 'bank' GETATT is to store the attributes of that structure whose reference number is given in the second argument. The banks are a series of arrays in common /DATAC/; each array is of length 3 and holds a particular attribute of each of three structures. You can visualise the 'banks' as an array of information as pictured below:

**Example**

| Bank | Structure | TYPE | VALOR | NVAL | MODE | NLEV | NROW | NCOL |
|---|---|---|---|---|---|---|---|---|
| 1 | variate with 15 values | 4 | 1026 | 15 | 2 | 0 | 0 | 0 |
| 2 | undeclared identifier | ISMV | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 4×3 matrix with no values | 11 | 0 | 12 | 2 | 0 | 4 | 3 |

Thus you can, if you want to, invoke GETATT three times, increasing the first parameter from 1 to 3, and thus make available the attributes of three structures at the same time. If you then want to refer to a fourth structure, you will have to lose the information on one of the previous ones.

The attributes and corresponding arrays in common /DATAC/ which you will need most are as follows:

TYPE The coded type of the structure. A list of codes is given in the Genstat Manual, Part II 6.2.3, but the commonest are:

| | | | |
|---|---|---|---|
| -4 | Scalar | 7 | Symmat |
| 1 | Integer | 11 | Matrix |
| 4 | Variate | 12 | Diagmat |
| | | 13 | Factor |

If the structure has not been declared when GETATT is called, it will automatically be set up as a variate, i.e. its TYPE will be set to 4. This automatic declaration can be suppressed (see below), in which case the TYPE wil be missing, equal to ISMV.

VALOR The origin of values in the appropriate data array in blank common. If there are no values yet, then VALOR is zero.

NVAL The number of values. This is zero if values have not been assigned and the length of the structure has not been declared. However, if there is a 'UNIT' statement in force in your Genstat program when 'OWN' is used, then when GETATT is called for a vector (e.g. a variate, integer or factor) whose length has not already been defined, it will automatically declare the structure to have as many values as defined by the 'UNIT' statement.

MODE The mode of storage of the values. If the mode is 2 and values are present, they are in RDATA(VALOR+1) up to RDATA(VALOR+NVAL), where RDATA is the array in blank common. Mode 2 corresponds to real numbers, so is used for scalars, variates and matrices. If the mode is 4, values are in ISDATA(VALOR+1) up to ISDATA(VALOR+NVAL). This mode corresponds to short integers and is used for integers and factors: factor values are stored in coded form as 0, 1 ,..., NLEV-1: thus level 3 is represented as 2, for example.

NLEV The number of levels of a factor, or number of rows (or columns) of a symmat.

MVPTR The number of missing values of a structure.

NROW The number of rows of a matrix.

NCOL The number of columns of a matrix.

There are many other attributes but most are set only for more complicated structures such as tables. All are listed in the Implementor's Manual, Chapter 4.

**Example**

Suppose you want to access one variate structure, whose values should be present, and you want to pass the array of values and its length to a prepared Fortran subroutine called WORK.

```
SUBROUTINE OWN
  (Commons)
INTEGER *2 GETATT
IF (ISDATA(PCI + 2) .NE. 1) RETURN
IVAR = ISDATA(PCI + 3)
IF (GETATT(1, IVAR) .NE. 0) RETURN
IF (TYPE(1).NE.4 .OR. VALOR(1).EQ.0) RETURN
IFIRST = VALOR(1) + 1
CALL WORK (RDATA(IFIRST), NVAL(1))
RETURN
END

SUBROUTINE WORK(X, N)
REAL X(N)
  (Statements)
RETURN
END
```

**Defining Data Structures**

As well as passing information from previous Genstat statements to your Fortran subprograms, you may well want to do the reverse. If so, you must include, in the 'OWN' statement, identifiers to be used to store the information. If you know the type and length of these structures, then you can declare them in your Genstat program before giving the 'OWN' statement; however if the type or length is determined by calculations within the OWN subroutine, you will have to define the structures in full in that subroutine.

When a structure intended to store results has no values on entry to subroutine OWN, you must set up a block of values for it. This is done by using the standard short integer function SVALOR, e.g.

```
IF (SVALOR(1) .NE. 0) RETURN
```

This sets up a values block for the structure currently in bank 1 and sets the origin of the block in VALOR(1); the values are all initialised to the missing value. The function SVALOR returns the value 0 unless the parameter is invalid or there is not enough workspace for the values block. However, the structure itself is not modified by SVALOR(1). To store the value origin in the directory of structures, you must call the standard function PUTATT, e.g.

```
IF= (PUTATT(1, N) .NE. 0) RETURN
```

This takes all the attributes currently in bank 1 and assigns them to the structure with reference number N. In the example above, only the value origin will be changed (from 0 to whatever value SVALOR assigned).

After your subprograms have assigned values to the structure, there may be some values which are still missing. If so, you must use the short integer functions CNMV and PUTATT to count the number of missing values and store the count in the attribute MVPTR:

```
IF (CNMV(1) .NE. 0) RETURN
IF (PUTATT(1, N) .NE. 0) RETURN
```

When a structure intended to store results has not been defined at all on entry to subroutine OWN, you must define all the necessary attributes and then invoke PUTATT to store them all.

**Example**

Suppose you want to set up one variate, whose length and identifier are supplied by OWN, to store results from a prepared Fortran subroutine.

```
      SUBROUTINE OWN
        (Commons)
      INTEGER *2 CNMV, GETATT, SVALOR, PUTATT
      IF (ISDATA(PCI + 2) .NE. 2) RETURN
      IVAR = ISDATA(PC1 + 3)
      LVAR = ISDATA(PC1 + 4)
      IF (GETATT(1, IVAR) .NE. 0) RETURN
C         Check that the structure is a variate, or was not defined
C         before GETATT was called
      IF (TYPE(1) .NE. 4) RETURN
C         Set the length of the variate
      NVAL(1) = LVAR
      IF (SVALOR(1) .NE. 0) RETURN
      IFIRST = VALOR(1) + 1
      CALL WORK (RDATA(IFIRST), LVAR)
      IF (CNMV(1) .NE. 0) RETURN
      IF (PUTATT(1, IVAR) .NE. 0) RETURN
      RETURN
      END
```

## Some Extra Details

### Diagnostics

In the examples above, any mistakes in the input resulted in an immediate exit from subroutine OWN. If this happens, for example if there was no free space to set up a variate, the routine which calls subroutine OWN will print a standard Genstat diagnostic to the current output file. The diagnostic is controlled by the coded variable DIAG in common /DIAGPK/, which is usually zero but is set to a positive integer when a fault is found.

```
COMMON/DIAGPK/SPN(10),DIAG,MAXSP,NSP,STATNO,NDIAG
DOUBLE PRECISION SPN
INTEGER *2            DIAG,MAXSP,NSP,STATNO,NDIAG
```

In fact, the result of the standard functions GETATT, SVALOR, etc. is set equal to the value of DIAG.

You can therefore set up your own diagnostics, if you want, by referring to DIAG and printing messages or by using some coded value to represent some particular error. If DIAG has a non-zero value when control returns from your OWN subroutine, a standard diagnostic message will be printed. You should refer to the Implementor's Manual, Section 2.3, for information about all the codes. Typically, you may want to have one of the VA diagnostics printed, e.g. VA 4 Values not set. A VA message will be printed if the value of DIAG is 170 plus the relevant VA diagnostic number (maximum 20); thus VA 4 corresponds to DIAG=174.

If DIAG may be non-zero, you should include a call to subroutine DIAGUP before exiting from subroutine OWN. This will include a reference to the OWN subroutine in the diagnostic trace and will help to show that the failure has occurred through use of 'OWN'. For example, the end of your OWN subroutine may look like the following.

```
      IF (PUTATT(1, IVAR) .NE. 0) GO TO 1001
      RETURN
1001  CALL DIAGUP(8H      OWN)
      RETURN
```

**Workspace**

In the examples above, it was assumed that all calculations in the Fortran subprogram would be done with local variables or with values of standard structures. However, you can make use of the free workspace available in large blank common arrays if you want to. This is advisable particularly if you need variable amounts of workspace, e.g. depending on the size of input structures.

To set aside a block of workspace, you must invoke the standard function GWSP and include the common /WSP/ in subroutine OWN (this common is not available in the block data module).

```
COMMON/WSP/    ENDWSP,LSTWSP,WSPOR
INTEGER        ENDWSP,LSTWSP,WSPOR
```

Function GWSP has two arguments: the number of values required and their mode. The value of the function is the diagnostic code, which is non-zero if the parameters are invalid or there is not enough free space available. After invoking GWSP, the variable WSPOR in common /WSP/ holds the origin of the block of values in the relevant array in blank common, so that the first location of the workspace used is RDATA(WSPOR+1) in mode 2, or ISDATA(WSPOR+1) in mode 4.

If you want workspace for double length reals or long integers, use modes 1 and 3, and arrays DATA and IDATA, respectively.

**Example**

To pass an array of M real values to be used as workspace in a prepared Fortran subprogram.

```
IF (GWSP(M, 2) .NE. 0) RETURN
IFIRST = WSPOR+1
CALL WORK(RDATA(IFIRST), M)
```

**Output**

Output from your Fortran subprogram can be produced by Fortran WRITE statements. Such output may get mixed up with standard Genstat output because the latter is handled with a buffer; if this is a problem, there are standard routines in Genstat for outputting values and text through the buffer, described in the Implementor's Manual. You should use the correct unit numbers for output: these vary between implementations of Genstat. You can find out the unit numbers by running the standard version of Genstat and giving the command:

```
'ENVIRONMENT/PRINT=P'
```

The current output unit number is available in variable QWW in common /PERIPH/. (The current unit can be changed by giving 'OUTPUT' statements in your Genstat program.)

```
 COMMON/PERIPH/BSWF,MAXBSN,NLRPR,QRL(4),QRR,QWL(4),QWW,SQR(4),
1              LINE(4),MAXNLP,NCOPR(4),NINCH
 INTEGER       BSWF,MAXBSN,NLRPR,QRL,   QRR,QWL,   QWW,SQR,
 INTEGER *2    LINE,   MAXNLP,NCOPR,   NINCH
```

**Example**

```
      WRITE(QWW, 23) (RDATA(I), I = J1, J2)
   23 FORMAT(' Results:' / (10F12.4))
```

**Avoiding the Default Declaration**

When you want to set up your own structures in subroutine OWN, it can be inconvenient if GETATT automatically sets them up as variates of standard length before you can specify their attributes. The automatic declaration by GETATT can be suppressed by setting variable ACCSW

in common /MAINAC/ to 0 before invoking GETATT. You must reset ACCSW to 1 after calling GETATT.

```
      COMMON/MAINAC/IORATT,IORID,IORSUF,IORTAG,IORTYP,IORVAL,
     1              ACCSW,INDTYP(30),LATTBL(30),MAXID,MAXUNN,NID
      INTEGER       IORATT,IORID,IORSUF,IORTAG,IORTYP,IORVAL
      INTEGER *2    ACCSW,INDTYP    ,LATTBL    ,MAXID,MAXUNN,NID
```

**Example**

To set up a previously undeclared identifier as a matrix with NR rows and NC columns.

```
      ACCSW = 0
      IF (GETATT(1, N) .NE. 0) RETURN
      ACCSW = 1
C         Check that N refers to a new structure
      IF (TYPE(1) .NE. ISMV) RETURN
      TYPE(1) = 11
      MODE(1) = 2
      NVAL(1) = NR*NC
      NROW(1) = NR
      NCOL(1) = NC
      IF (SVALOR(1) .NE. 0) RETURN
      IF (PUTATT(1) .NE. 0) RETURN
```

## Caution

You must be careful not to cause confusion with the standard version of Genstat. The names of your Fortran subprograms and any common blocks in them must not clash with names in the Genstat code: for safety, use names of the form OWN... for all of these.

We also suggest you make it clear in all Genstat output that your version of Genstat has been modified. You can do this by editing the Fortran subprogram STARTJ in module CPC of the Genstat code. There is a call to the subroutine ENCAPB there which prints the header message output at the start of each Genstat job. Please insert another call to ENCAPB after this call, to print a message giving your name, as in the following:

```
CALL ENCAPB(1,0,26HGENSTAT V    RELEASE 4.04B,26)
CALL ENCAPB(1,0,46H  MODIFIED BY P.W. LANE & P.G.N. DIGBY 10-4-85,46)
```

# An Implementation of the Genstat 'OWN' Directive

*L.G. Underhill*
*Department of Mathematics and Statistics*
*University of Cape Town*
*Rondebosch 7700*
*South Africa*

This article should be read in conjunction with that of Lane and Digby earlier in this issue. It describes a specific application of the 'OWN' directive, and follows their procedure in detail. The particular Fortran program to be linked into Genstat is a nonmetric regression subroutine. This is a first step towards the inclusion within Genstat of a nonmetric scaling directive, of which nonmetric regression is an integral part (Kruskal 1964, Greenacre and Underhill 1982). Given an ordered set of numbers $d_1, d_2, ..., d_m$ and a set of weights $w_1, w_2, ..., w_m$, the problem in nonmetric regression is to obtain

$$f(d_1), f(d_2), ..., f(d_m)$$

such that

$$f(d_1) \leq f(d_2) \leq ... \leq f(d_m)$$

and

$$\sum_{i=1}^{m} (d_i - f(d_i))^2 \quad \text{is a minimum.}$$

Nonmetric regression is described in some detail in both of the references. The full listing of the subroutine which interprets the 'OWN' directive is given, to serve as a model for other users. The nonmetric regression subroutines are also included.

The Fortran subroutine to be included in Genstat is called FITWT and has five formal parameters:

| | | |
|---|---|---|
| M | : integer | the length of the arrays (*m* above) |
| DHAT | : real array | on input the vector of distances, *d*, in the required order; on output, the vector of monotonically ordered 'pseudo-distances', $f(d)$ |
| WT | : real array | a vector of weights |
| WHAT | : real array | working space |
| DOTHER | : short integer array | working space |

The OWN subroutine listed below is documented so that, apart from the COMMONS and EQUIVALANCES, each line of code is explained by the preceding comment. Read in conjunction with Lane and Dibgy's article, the subroutine should prove straightforward to follow. The diagnostic codes at the end of the subroutine have been obtained from Section 2.3 of the Implementors Manual. The subroutine has been written so that up to ten different Fortran subroutines can be linked into Genstat simultaneously. The syntax of the 'OWN' directive permits only one list:

'OWN' *ilist*

The syntax of my 'OWN' directive is

'OWN' *1,variate1,variate2*

The first item in the list is used as an index to the particular subroutine wanted. (So far I have only implemented 'OWN' 1.) *Variate1* is the ordered vector $d_1, d_2, ..., d_m$ and *variate2* is the set of weights $w_1, w_2, ..., w_m$. The nonmetric regression $f(d_1), f(d_2), ..., f(d_m)$ is returned through *variate1*, and *variate2* is unchanged.

The output of a Genstat programme which uses the 'OWN' directive for monotonic regression is also included. It demonstrates, firstly, that the method works, secondly, that the linking in of the diagnostics works (the program failed on the fourth call to 'OWN' because the variates were of unequal length) and, thirdly, that the list of items for the 'OWN' directive need not be the same as the list of formal parameters in the Fortran subroutine. As a Fortran programmer I took a long time to realise that only two of the five parameters for my subroutine needed to go into the 'OWN' item-list. The length of the arrays could be determined within the interpreter and two of the arrays were working space which did not require formally declared variates.

I recommend that future versions of Genstat include, in the standard 'OWN' interpreter subroutine, all the necessary COMMONs and EQUIVALENCEs, as detailed by Lane and Digby. This will make the linkage of user subroutines simpler. The ability to incorporate specialised Fortran programs could become a feature of Genstat.

Incidentally, working from the draft version of Lane and Digby's paper, it took me about six hours to write and debug the interpreter for the 'OWN' directive to include the monotonic regression subroutine. With the experience gained, I could probably add other subroutines in under an hour.

## The Fortran Code

```
      SUBROUTINE OWN
C
C          LES UNDERHILL    MARCH 1985
C
C-----------------------------------------------------------------------
C
      COMMON       DATA(1)
      DIMENSION    CDATA(1),RDATA(1),IDATA(1),ISDATA(1),TABLE(1)
      DOUBLE PRECISION DATA,CDATA
      REAL         RDATA
      INTEGER      IDATA
      INTEGER *2   ISDATA
C
      EQUIVALENCE (DATA(1),CDATA(1),RDATA(1),IDATA(1),
     1             ISDATA(1),TABLE(1))
C
      COMMON/DATAC/IDENT(3),ATTOR(3),TYPE(3),VALOR(3),NVAL(3),MODE(3),
     1  VECNO(3),DESC(3),MVPTR(3),ACC(3),SET(3),UNIVEC(3),STAVEC(3),
     2  SUBCLS(3),NIND(3),NUN(3),NSV(3),SPEC1(3),SPEC2(3),SPEC3(3),
     3  IDSUF(3),ENDDAT,LATT,LSTDAT,MAXDAT,PCI
      DOUBLE PRECISION IDENT
      INTEGER       ACC,ADDPTR(3),ATT(57),ATTOR,BMINW(3),DESC,DIAVEC(3),
     1  ENDDAT,FSET(3),IDIV(3),IDSUF,KROOTS(3),LATT,LNVEC(3),LSTDAT,
     2  MARG(3),MATVEC(3),MAXDAT,MODE,MVPTR,NCOL(3),NIND,NLEV(3),
     3  NRC(3),NROW(3),NSV,NUMEX(3),NUN,NVAL,NVAR(3),PCI,SCALNO(3),
     4  SDMVEC(3),SET,SETC(3),SETR(3),SPEC1,SPEC2,SPEC3,STAVEC,SUBCLS,
     5  TABVEC(3),TYPE,UNIVEC,USET(3),VALOR,VECNO,VSET(3),VTYPE(3),
     6  WTVAR(3)
C
```

```
      COMMON/SYSCON/DBLANK,DEPS,DMV,DTOL,EPS,MRSI,RMV,TOL,
     1              IMV,MASK,NBB,
     2              ISMV,ISZERO,MSI,NBI,NBV(5),NVDR(5),NIR,NSII,NSIR
      DOUBLE PRECISION
     1              DBLANK,DEPS,DMV,DTOL
      REAL          BLANK,EPS,MRSI,RMV,TOL
      INTEGER       IBLANK,IMV,MASK,NBB
      INTEGER *2    ISBLNK,ISMV,ISZERO,MSI,NBI,NIR,NSII,NSIR,
     1              NBV,NVDR,NIDR,NRDR,NSIDR,NSWDR
C
      EQUIVALENCE (ATT(1),ATTOR(1)),
     1   (SET(1),LNVEC(1),NUMEX(1),SDMVEC(1),SETR(1),VTYPE(1),WTVAR(1)),
     2   (UNIVEC(1),NLEV(1),NVAR(1),SETC(1),TABVEC(1)),
     3   (STAVEC(1),IDIV(1),KROOTS(1),NRC(1)),
     4   (SUBCLS(1),ADDPTR(1),BMINW(1),MARG(1),NROW(1),VSET(1)),
     5   (NIND(1),FSET(1),MATVEC(1),NCOL(1),USET(1)),
     6   (NUN(1),DIAVEC(1)),  (NSV(1),SCALNO(1))
C
      EQUIVALENCE (DBLANK,BLANK,IBLANK,ISBLNK),
     1      (NSWDR,NSIDR,NVDR(4)), (NIDR,NVDR(3)), (NRDR,NVDR(2))
C
      COMMON/DIAGPK/SPN(10),DIAG,MAXSP,NSP,STATNO,NDIAG
      DOUBLE PRECISION SPN
      INTEGER *2    DIAG,MAXSP,NSP,STATNO,NDIAG
C
      COMMON/WSP/ENDWSP,LSTWSP,WSPOR
      INTEGER    ENDWSP,LSTWSP,WSPOR
C
C          INITIALISED LOCAL
C
      DOUBLE PRECISION
     1                SPNAME
C
C          FUNCTIONS
C
      INTEGER *2 GETATT,PUTATT,SVALOR,GWSP
      DATA SPNAME /8HOWN     /
C
C---------------------------------------------------------------------
C
C          NO OF ITEMS IN LIST OF 'OWN' DIRECTIVE
      LISLNG=ISDATA(PCI+2)
C          FIRST ITEM IN LIST SHOWS WHICH 'OWN' DIRECTIVE, 'OWN' 1 TO 'OWN' 10
      NUMOWN= ISDATA(PCI+3)
      GO TO (10,20,30,40,50,60,70,80,90,100) NUMOWN
C
```

```
C          OWN 1 : MONOTONIC REGRESSION OF ORDERED SET DHAT WITH WEIGHTS WT
C
C          OWN 1   MUST HAVE 3 ITEMS
10    IF(LISLNG.NE.3) GOTO 1029
C          REFERENCE NUMBERS OF SECOND AND THIRD ITEMS IN LIST
      IVAR1=ISDATA(PCI+4)
      IVAR2=ISDATA(PCI+5)
C          GETATT  SHOULD RETURN A ZERO, 2nd & 3rd ITEMS TO ''BANKS'' 1 & 2
      IF(GETATT(1,IVAR1).NE.0) GOTO 1000
      IF(GETATT(2,IVAR2).NE.0) GOTO 1000
C          CHECKS THAT ITEMS IN BANKS 1 AND 2 ARE VARIATES
      IF(TYPE(1).NE.4)         GOTO 1181
      IF(TYPE(2).NE.4)         GOTO 1181
C          VALUES FOR THE VARIATES MUST HAVE BEEN SET
      IF(VALOR(1).EQ.0)        GOTO 1174
      IF(VALOR(2).EQ.0)        GOTO 1174
C          NUMBER OF VALUES IN THE VARIATES MUST BE EQUAL AND NON-ZERO
      LENG1=NVAL(1)
      LENG2=NVAL(2)
      IF(LENG1.NE.LENG2)       GOTO 1183
      IF(LENG1.EQ.0.OR.LENG2.EQ.0) GOTO 1172
C          ORIGINS OF VARIATES IN BANKS 1 & 2
      IADR1=VALOR(1) + 1
      IADR2=VALOR(2) + 1
C          SET ASIDE REAL WORKSPACE OF LENGTH  LENG1
      IF(GWSP(LENG1,2).NE.0)   GOTO 1000
C          ORIGIN OF REAL WORKSPACE
      IADR3=WSPOR + 1
C          SET ASIDE SHORT INTEGER WORKSPACE OF LENGTH  LENG1
      IF(GWSP(LENG1,4).NE.0)   GOTO 1000
C          ORIGIN OF SHORT INTEGER WORKSPACE
      IADR4=WSPOR + 1
C          CALL TO NONMETRIC REGRESSION SUBROUTINE, NORMALLY
C     CALL FITWT(M,    DIST,        WT,          WHAT,        DOTHER)
      CALL FITWT(LENG1,RDATA(IADR1),RDATA(IADR2),RDATA(IADR3),ISDATA(IADR4))
      GOTO 200
C          PROVISION FOR FURTHER OWN DIRECTIVES, 'OWN' 2...'OWN' 10
20    CONTINUE
      GOTO 200
30    CONTINUE
      GOTO 200
40    CONTINUE
      GOTO 200
50    CONTINUE
      GOTO 200
60    CONTINUE
      GOTO 200
70    CONTINUE
      GOTO 200
```

```
80    CONTINUE
      GOTO 200
90    CONTINUE
      GOTO 200
100   CONTINUE
      GOTO 200
200   RETURN
C       DIAGNOSTICS
C       NUMERICAL VALUES FOR CODES ARE IN SECTION 2.3 OF IMPLEMENTORS GUIDE
C       SX-19 WRONG LIST LENGTH
1029  DIAG=29
      GOTO 1000
C       VA-2  ATTRIBUTES NO SET
1172  DIAG=172
      GOTO 1000
C       VA-4  VALUES NOT SET
1174  DIAG=174
      GOTO 1000
C       VA-11 INVALID TYPE
1181  DIAG=181
      GOTO 1000
C       VA-13 INCOMPATIBLE NUMBER OF VALUES
1183  DIAG=183
1000  CALL DIAGUP(SPNAME)
      GOTO 200
      END
C
C
C       SUBROUTINES SATSFY, JOINWT AND FITWT PERFORM THE NONMETRIC REGRESSION
C
C
      SUBROUTINE SATSFY(DHAT,DOTHER,M,NUPDOW,IACTIV,SATIS,NEXT,
     * NEXT1,IACTV1,NOBLOC)
      INTEGER *2 DOTHER(5000),SATIS
      REAL DHAT(5000)
      SATIS = 1
      IF (NUPDOW.EQ.-1. AND. IACTIV.EQ.1) RETURN
      IF (DOTHER(IACTIV).NE.0) GO TO 2
      IACTV1 = IACTIV
      NOACBL = 1
      GO TO 1
2     IACTV1 = IACTIV + 1
      NOACBL = DOTHER(IACTIV)
    1 IF (NUPDOW .EQ. -1) GO TO 4
      NEXT = IACTIV + NOACBL
      IF (NEXT.EQ.M+1) RETURN
      IF (DOTHER(NEXT) .NE.0) GO TO 3
```

```
    5 NEXT1 = NEXT
      NOBLOC = NOACBL+1
      GO TO 6
4     NEXT = IACTIV-1
      IF(DOTHER(IACTIV-1 ).EQ.0) GO TO 5
      NEXT = DOTHER(IACTIV-1)
3     NOBLOC = NOACBL + DOTHER(NEXT)
      NEXT1 = NEXT+1
6     IF ((NUPDOW*DHAT(IACTIV)).GT.(NUPDOW*DHAT(NEXT))) SATIS=-1
      RETURN
      END
C
C
C
      SUBROUTINE JOINWT (DHAT,WHAT,DOTHER,I,J,I1,J1,NOBLOC,M)
      INTEGER *2 DOTHER(5000)
      REAL DHAT(5000),WHAT(5000)
      II = MIN0 (I,J)
      IEND = II+NOBLOC-1
      DHAT(II+1) = DHAT(J1)+DHAT(I1)
      WHAT(II) = WHAT(I)+WHAT(J)
      DOTHER(II) = NOBLOC
      DHAT(II) = DHAT(II+1)/WHAT(II)
      DOTHER(IEND) = II
      I=II
      J=II
      RETURN
      END
C
C
C
      SUBROUTINE FITWT(M,DHAT,WT,WHAT,DOTHER)
      REAL DHAT(5000),WT(5000),WHAT(5000)
      INTEGER *2 DOTHER(5000),SATIS
      DO 1 J=1,M
      WHAT(J)=WT(J)
      DHAT(J) = DHAT(J) * WHAT(J)
    1 DOTHER(J) = 0
      IACTIV = 1
    4 NUPDOW=1
5     CALL SATSFY(DHAT,DOTHER,M,NUPDOW,IACTIV,SATIS,NEXT,NEXT1,
     * IACTV1,NOBLOC)
      IF (SATIS.EQ.-1) GO TO 2
      NUPDOW = -NUPDOW
      CALL SATSFY(DHAT,DOTHER,M,NUPDOW,IACTIV,SATIS,NEXT,NEXT1,
     * IACTV1,NOBLOC)
      IF(SATIS.EQ.-1) GO TO 2
      IF(DOTHER(IACTIV).NE.0) IACTIV = IACTIV+DOTHER(IACTIV)-1
      IACTIV = IACTIV+1
```

```
      IF(IACTIV.NE.M+1) GO TO 4
      J = 1
    3 J = J+1
      IF (J.EQ.M+2) RETURN
      IF (DOTHER(J-1).EQ.0) GO TQ 3
      K = J+DOTHER(J-1)-2
      DO 6 KJ=J,K
6     DHAT(KJ) = DHAT(J-1)
      J=K+1
      GO TO 3
2     CALLJOINWT(DHAT,WHAT,DOTHER,NEXT,IACTIV,NEXT1,IACTV1,NOBLOC,M)
      NUPDOW = -NUPDOW
      GO TO 5
      END
```

## Test Run

```
 GENSTAT V    RELEASE 4.04B
   MODIFIED BY LES UNDERHILL, MARCH 1985
 COPYRIGHT 1984 LAWES AGRICULTURAL TRUST (ROTHAMSTED EXPERIMENTAL STATION)

    1  'refe' nonmetric
    2  'vari' d$5=1,3,2,5,10
    3  :w$5=5(1.)
    4  :dhat$5
    5  'calc' dhat=d
    6  ''find the nonmetric regression of the ordered set d with weights  w=1''
    7  ''note that, apart from the 3 and the 2, d is in ascending order''
    8  'own' 1,dhat,w
    9  'print/p' d,w,dhat $ 10.4
   10  'run'



         d          w       dhat
    1.0000     1.0000     1.0000
    3.0000     1.0000     2.5000
    2.0000     1.0000     2.5000
    5.0000     1.0000     5.0000
   10.0000     1.0000    10.0000

   11  'vari' x$10=10,9,8,7,6,5,4,3,2,1
   12  : wt$10=10(1.)
   13  : xhat $ 10
   14  'calc' xhat=x
   15  ''find the nonmetric regression of the ordered set x with weights  w=1''
  -16  ''note that the vector x is in descending order, so that the nonmetric
   17  regression  xhat  is the average of the x's''
   18  'own' 1,xhat,wt
   19  'print/p' x,wt,xhat$ 10.4
   20  'run'
```

```
         x         wt       xhat
   10.0000    1.0000    5.5000
    9.0000    1.0000    5.5000
    8.0000    1.0000    5.5000
    7.0000    1.0000    5.5000
    6.0000    1.0000    5.5000
    5.0000    1.0000    5.5000
    4.0000    1.0000    5.5000
    3.0000    1.0000    5.5000
    2.0000    1.0000    5.5000
    1.0000    1.0000    5.5000

 21  'valu' x=5,4,3,2,1,10,9,8,7,6
 22  'calc' xhat=x
 23  'valu' wt=5(.75),1.5,.5,3(1.)
 24  ''weights are now not all equal to one''
 25  'own' 1,xhat,wt
 26  'print/p' x,wt,xhat $ 10.4
-27  ''the own directive should now fail  -  the variates xhat and wt are of
 28  different lengths''
 29  'own' 1,x,w
 30  'run'



         x         wt       xhat
    5.0000    0.7500    3.0000
    4.0000    0.7500    3.0000
    3.0000    0.7500    3.0000
    2.0000    0.7500    3.0000
    1.0000    0.7500    3.0000
   10.0000    1.5000    8.1000
    9.0000    0.5000    8.1000
    8.0000    1.0000    8.1000
    7.0000    1.0000    8.1000
    6.0000    1.0000    8.1000

******** FAULT IN STATEMENT 1 ON LINE 29, CODE VA 13
Invalid or incompatible numbers of values

** TRACE  OWN

***** DUMP *****

*** RECENTLY REFERENCED STRUCTURES

    IDENTIFIER    TYPE LENGTH  VALUES MISSING REF.NO. BANK
             x VARIATE     10 PRESENT       0      -5    1
             w VARIATE      5 PRESENT       0      -3    2
```

```
*** USE OF STORE  (DATA UNITS)
       TOTAL STORAGE        32768
 STRUCTURE DIRECTORY          675
     DATA STRUCTURES          149
          STATEMENTS           44
      USED WORKSPACE            0
      FREE WORKSPACE        31900

*** STRUCTURES
          NAMED UNNAMED
NO. USED      7       4
 MAXIMUM    100      50

  31  'close'

******** END OF nonmetri. MAXIMUM OF 892 DATA UNITS USED AT LINE 26 (31876 LEFT)
```

## References

[1] Greenacre, M.J. and Underhill, L.G.
Scaling a data matrix in a low-dimensional Euclidean space.
In *'Topics in Applied Multivariate Analysis'* (Hawkins, D.M. ed), 183-268, 1982.

[2] Kruskal, J.B.
Multidimensional scaling by optimizing goodness-of-fit to a non-metric hypothesis.
*Psychometrika* **29**, 1-27 & 115-129, 1964.

# Some Uses of the 'OWN' Directive: Interfaces between Genstat and other Packages and Interruption of Genstat Sessions

*A. Bouvier*
*INRA, Laboratoire de Biometrie*
*Institut National de la Recherche Agronomique*
*Domaine de Vilvert*
*78350 Jouy-en-Josas*
*France*

**Abstract**

The Genstat OWN directive is intended for testing or implementing local facilities at a site. In Genstat as distributed, an OWN statement has no effect, because the Fortran subroutine which is invoked by the statement merely returns without doing anything. Any user with access to the Fortran source code of Genstat can replace this null subroutine with one designed for his own needs (as described in the preceding article by Lane and Digby). We show here how we used the OWN directive:

(1) to transfer data structures between Genstat and another package;

(2) to interrupt a Genstat session, allowing commands to be given to the local operating system before resuming the session.

## Transferring Data Structures between Genstat and another Package

### The Interfaced Package

An interface between Genstat and the data base management system Socrate has already been programmed on an IRIS-80 computer, system SIRIS8 (Bouvier, 1984). Another interface is now available on a DPS8 computer, running Multics, between Genstat and the interactive data manipulation and analysis system 'Consistent System' (or CS) which includes a relational data base management sub-system called Janus. (CS is produced commercially by RCI, Cambridge, Massachusetts, USA.)

### The Syntax of the OWN Directive

The OWN directive manages the transfer of data structures. We have added options to the OWN directive by modifying the arrays in the Genstat code which are used by the compiler. Option OUTPUT indicates the direction of the transfer; the setting OUTPUT=Y means that a data structure is to be transferred from Genstat to CS. The default setting of the option OUTPUT (OUTPUT=N) means that the transfer is directed from CS to Genstat. When the option DIAG=Y is set, if CS structures already exist with the same name as those to be created, a diagnostic is printed and no transfer takes place. If the option PRINT=Y is set, the transferred values are printed. The option COMP=Y (COMP standing for 'complete') means that CS labels must also be transferred (every CS structure can have labels, i.e. alphanumeric names which identify the dimensions or the elements).

The arguments of the OWN directive are the identifiers of the structures to be transferred; the structures created have the same names, or derived names when several structures are created to store all the information originally contained in a single structure.

### Matching Data Types

The types of data structures in CS and Genstat are not the same and equivalences had to be found. For example, to transfer CS labels, a Genstat NAME structure is created in addition to the structure which stores the numerical values.

CS structures of more than two dimensions are not transferred into multi-way tables because the structures have a precise meaning in Genstat which does not always correspond to the user's intention. They are transferred into VARIATE structures, which are more flexible, and FACTOR structures are generated to store the dimensional indices.

In the other direction, Genstat structures may be labelled by other structures; these are not transferred into CS structures but their values are used to form CS labels.

**Error Messages**

Sixteen new diagnostic messages have been introduced with codes beginning with the two letters 'CS'.

For example, CS 5 means 'the data structure to be transferred cannot be found'.

**Programming Details**

We wrote the OWN subroutine (which manages the OWN directive) in the language PL/1, because it is the basic language of Multics and CS. This subroutine calls the Genstat Fortran subroutines which access and create Genstat data structures and print diagnostic messages (FINDIN, GETATT, DIAGUP) and it calls CS PL/1 subroutines to access and create CS data structures.

Some values in Genstat arrays were also modified, to introduce the options and the new error codes.

**Example**: transferring a vector and a scalar from CS to Genstat.

```
ec cs                       ** Open the CS session
R                           ** 'R' is the CS ready-message

print struct                ** Print the vector to be transferred:
ident                       ** an integer vector, with a dimension
  l1 l2 l3 l4               ** label, 'ident', and element labels
   1  2  3  4               ** 'l1', 'l2', 'l3', 'l4'.

R
print iscal                 ** Print the second CS structure:
   999                      ** an integer scalar.

R
exmul genstat               ** Open the Genstat session:
                            ** 'exmul' allows the execution of a
                            ** Multics command from CS.

                            ** Genstat program.
'HEADING' gen="struct"      ** Declaration of the HEADING
                            ** structures which contain the names
'HEADING' g="iscal"         ** of the structure to be
                            ** transferred.

'OWN/COMP=Y' gen,g          ** The 'OWN' directive asks for the
                            ** transfer of the CS files into Genstat
                            ** structures.  The option COMP means
```

```
                              **  that the labels must be transferred.

**    The system prints the name, type and size of the created Genstat
      structures.

***   CREATION D'UNE STRUCTURE NAME    **  ''ident'' contains the element
        DE NOM struct DE 4 ELEMENTS    **  labels.  The name of this
                                       **  structure is the dimension
                                       **  label.

***   CREATION D'UNE STRUCTURE INTE    **  The CS file named ''struct'' has
        DE NOM struct DE 4 ELEMENTS    **  been transferred into an
                                       **  identically named Genstat
                                       **  structure.

***   CREATION D'UNE STRUCTURE SCAL    **  The scalar ''iscal'' has also
          DE NOM iscal DE 1 ELEMENT    **  been transferred into an
                                       **  identically named Genstat
                                       **  structure.
```

## Interrupting a Genstat Session

### Why Interrupt a Genstat Session?

The other use we have made of the OWN directive is to interrupt a Genstat session temporarily. This has been done for two reasons:

(1) in the Genstat-CS interface system, interruptions are necessary when consulting CS structures;

(2) interruptions are necessary for adding new input, output or user files; the correspondence between the names of those files and the numbers used in the Genstat Fortran subroutines (and therefore the numbers indicated in the INPUT, OUTPUT or FILE directives) must be made outside the Genstat session.

### The Syntax of the OWN Directive

To interrupt the session, the OWN directive is used without any arguments; so no ambiguity exists between this and the previous use of OWN (for Genstat-CS data structure transfers).

### Programming Details

Genstat has been implemented in such a way that all data structures and working areas are kept. In the Multics system, the correspondence between the file numbers and the associated Fortran numbers is set up in an introductory PL/1 program written by D Clark of the University of Bath and heavily altered by A Blackman of Bristol University.

We modified this program so that new file names can be introduced after an interruption, with execution restarting where it was left.

**Example:** interrupting a Genstat session

```
          .                    **  GENSTAT PROGRAM
          .
          .
'OWN'                          **  'OWN' asks to interrupt the session.
                               **  This interruption is taken into
                               **  account at the first 'RUN' that
'RUN'                          **  follows the 'OWN' statement.

r 16:44 0.212 1                **  We are back at the Multics level:
                               **  this is the Multics ready-message.

genstat -in2 data              **  Re-open the Genstat session with a new
                               **  option, ''-in2'' which introduces the name of
                               **  a file to be read: (data in this file will
                               **  be read by a 'READ' statement in the
                               **  Genstat program).
'HEADING' h=''suite''
          .
          .                    **  Subsequent Genstat program.
          .
```

## Conclusion

We created this interface between CS and Genstat for local needs. CS is our basic package: it is very interactive, easy to learn, and very convenient for formatting data, making simple statistical analysis and piloting other packages. Genstat is used as a second level package: it is more powerful in elaborate statistical analysis.

The possibility of the local implementor 'opening' Genstat to other software is certainly an important improvement in the package's facilities.

## Reference

[1] Bouvier, A.
Interfacing Genstat and a database management system.
*Genstat Newsletter*, **13**, 6-8, 1984.

# Efficient Performance of Genstat on a VAX

*J. Sherington*
*D. Gilson*
*Statistics Department*
*The Agricultural Institute*
*19 Sandymount Avenue*
*Dublin 4*
*Eire*

## Introduction

A VAX computer running under VMS is a virtual memory machine. Memory is divided into pages each 512 bytes in size. When an image is activated, the system allocates a section of memory known as a working set. There are three parameters which determine set size for each process, WSDEFAULT, WSQUOTA and WSEXTENT.

The system initially gives the process WSDEFAULT pages; if more pages are required, the system will allow the process to grow up to WSQUOTA. If at this stage the process requires more memory, the system will give a loan of some more pages up to WSEXTENT. The memory in this loan region (pages between WSQUOTA and WSEXTENT) can be taken back at any time when the system requires it to give to some other process. Whenever reference is made to a page which is not in the working set, it is known as a page fault. We noticed when running Genstat that a large number of page faults occurred, which had a detrimental effect on the overall performance of the system. We therefore conducted a small experiment to determine the effect of altering the working set size.

## The Experiment

The computer was a VAX 11/750 with two megabytes of memory, used for a mixture of statistical and administrative work. Four different Genstat (version 4.04A) programs were run with the working set size (both WSQUOTA and WSEXTENT) given the values 200, 300 ,..., 1000 (and no loan region). They were run overnight when no other work was being done on the computer.

The Genstat programs were chosen to give a range of CPU time (approximately 1 to 10 minutes), a range of quantity of data and a variety of common statistical procedures (analysis of variance, tabulation, regression and correlation).

Brief descriptions of the programs follow. The numbers given are generally only approximate.

A. – One minute CPU time.
   – 120 variates were read, and 7 more were calculated using VSUM. There were 100 observations.
   – A three-factor analysis of variance was carried out on seven variates.

B. – Five minutes CPU time.
   – 30 variates were read for 500 observations.
   – For each variate, two tables of means were produced (TABULATE directive) and a completely randomised analysis of variance was performed.

C. – Five minutes CPU time.
   – The data space option S=2 was used to double the space available. 250 variates were read and 350 variates were calculated. There were 28 observations.
   – 30 8×8 correlation matrices were calculated and a two-factor analysis of variance was carried out on all 600 variates.

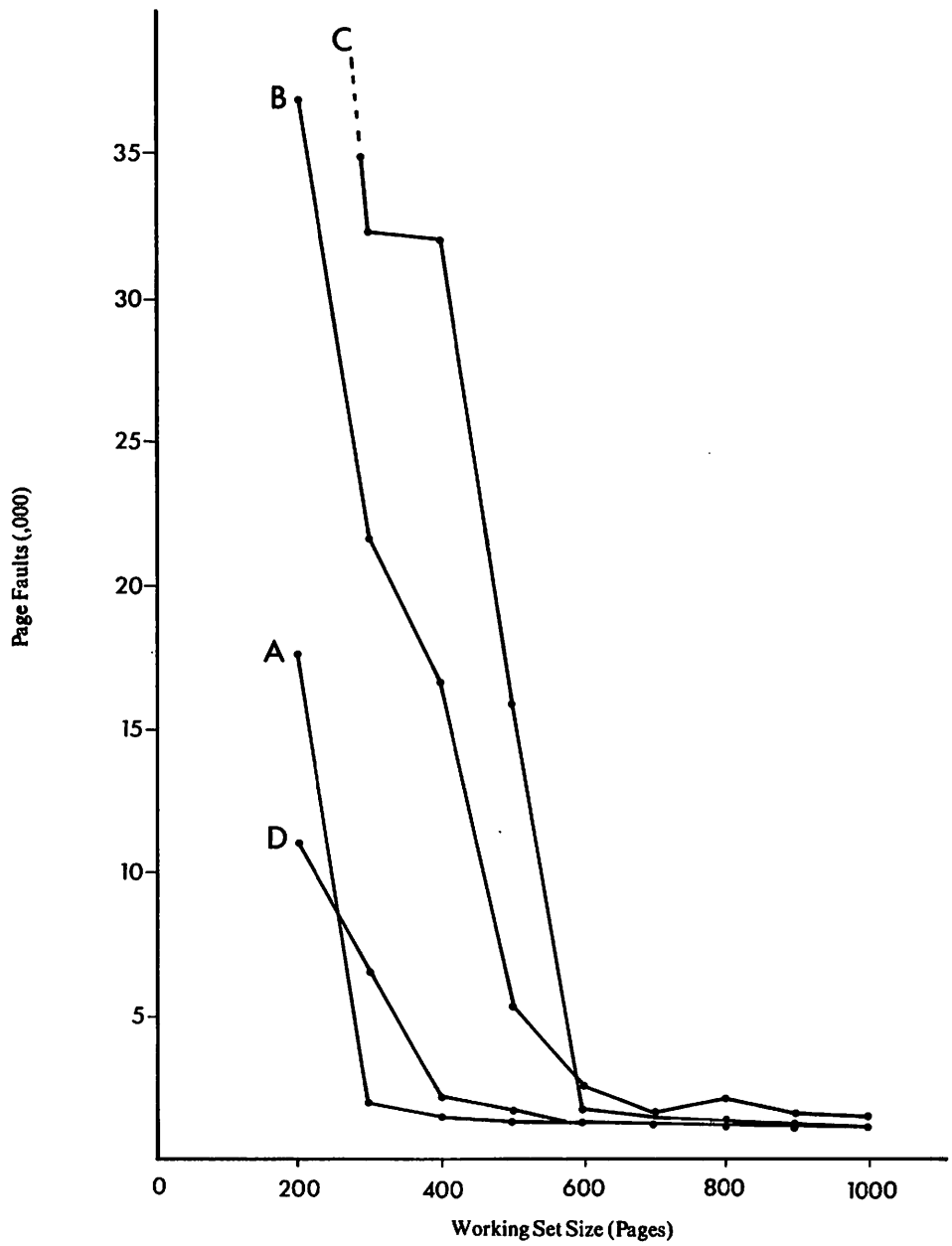
D. – Ten minutes CPU time.
   – 2 variates and 3 factors were read for 700 observations.

– An unbalanced analysis of variance using the `FIT`, `ADD` and `PREDICT/S` directives was performed for both variates.

The CPU time and number of page faults were recorded for each run.

**Results**

The increase in working set size had a dramatic effect on the number of page faults, as can be seen in Figure 1. For job A, there appeared to be little benefit in increasing above 300 pages, while for job D, 400 pages seemed to be the optimum. For both jobs B and C the number of page faults kept reducing up to working set sizes of about 600-700 pages, but there was no further reduction at greater sizes. These were the two jobs with the largest amounts of data.




Graph of Page Faults against Working Set Size
for Four Genstat Programs

**Figure 1**

The reduction in CPU time was less dramatic, but nevertheless worthwhile, for jobs A, B and C which had reductions of at least 12%.

The following table gives the number of page faults and CPU time for the default working set size of 200 pages and for an 'optimum' working set size of 700 pages.

| | Page Faults | | CPU Time seconds | | |
|---|---|---|---|---|---|
| Working set | 200 | 700 | 200 | 700 | Reduction |
| Job A | 17700 | 1300* | 85 | 73* | 14% |
| Job B | 36800 | 1600 | 316 | 268 | 15% |
| Job C | 58100 | 1500 | 327 | 288 | 12% |
| Job D | 11200 | 1200 | 603 | 597 | 1% |

(* Working Set size = 600 rather than 700)

## Conclusion

Although this experiment is obviously limited in scope, the range of programs used in this experiment covers a large proportion of the type of work we do on Genstat.

The standard working set size (`WSQUOTA`) of 200 pages in the VAX/VMS operating system would appear to be too small for most uses of Genstat. Also, jobs with large amounts of data would seem to benefit more than those with small amounts in terms of page faults. A working set size of 700 pages should be sufficient for all users except those regularly processing very large data sets, who may benefit from an even larger working set. It is not desirable to give all users very large working set quotas, as this would limit the number of processes in memory, thus downgrading the system.

As a result of this experiment we increased the working set sizes of all Genstat users in the Agricultural Institute. The working set quota (`WSQUOTA`) was increased to 700 pages and the working set extent (`WSEXTENT`) to 1024 (0.5 MB). This had a noticeable effect in increasing the overall efficiency of the system in terms of process through-put. The individual users also benefit by using less CPU time and system resources to complete the same amount of work. These modifications should help the performance of all similar VAX/VMS installations.