NMRSIM_G.LIB
Version 1.1, July 1993
A Library of Gauss Routines for Simulating Multivariate Normal Rectangle Probabilities and Their Derivatives

by

©Vassilis Argyrou Hajivassiliou

Department of Economics, London School of Economics

1 Introduction

A companion to this code is the paper **Simulation of Multivariate Normal Rectangle Probabilities and Their Derivatives: Theoretical and Monte Carlo Results** by V.A. Hajivassiliou, London School of Economics, D.L. McFadden, University of California at Berkeley, and P.A. Ruud, University of California at Berkeley, Journal of Econometrics, 1997.

The simulation methods presented in this paper have been coded in GAUSS and in FORTRAN. Both versions of the programs are available from the author's Web page through the URL: `econ.lse.ac.uk/~vassilis/simulation/gauss`, directly from the author upon request, or by anonymous `FTP` from the site `ariadne.lse.ac.uk:pub/simulation/gauss`. Instructions are given in Section 2.

Each simulator procedure requires the following standard inputs; the interpretation of some inputs may vary from routine to routine, and not all are used in all routines:

`M`	Dimension of the multivariate normal
`VMU`	Mean of multivariate normal, an $\mathtt{M} \times 1$ vector
`W`	Covariance matrix of multivariate normal, an $\mathtt{M} \times \mathtt{M}$ array
`WI`	Inverse of `W`
`C`	Lower triangular Choleski factor of `W`, an $\mathtt{M} \times \mathtt{M}$ array
`A`	Lower bound of rectangle, an $\mathtt{M} \times 1$ vector. {When the lower bound is $-\infty$, set $\mathtt{A} = (-\mathtt{1.0E10}) * \mathtt{ONES(M,1)}$.}
`B`	Upper bound of rectangle, an $\mathtt{M} \times 1$ vector. {When the upper bound is ∞, set $\mathtt{B} = (\mathtt{1.0E10}) * \mathtt{ONES(M,1)}$.}
`NR`	Number of repetitions
`U`	Random variates, an $\mathtt{M} \times \mathtt{R}$ array
`PARM`	Parameters and constants for the simulation

All the simulators return {P,HU,HC}, where P is the scalar rectangle probability, HU is the $(\mathtt{M}+1) \times (\mathtt{M}+1)$ array of unconditional partial moments, and HC is the $(\mathtt{M}+1) \times (\mathtt{M}+1)$ array of conditional moments. Parts of the output not provided by a simulator are set to −999.

The Gauss call is of the form

```
{P,HU,HC}=XXX(M,VMU,W,WI,C,A,B,NR,U,PARM)
```

where XXX stands for the 3 or 4 letter mnemonic for the simulator in question, as follows:

AUS Approximate Unbiased Simulator

ARSE Acceptance/Rejection Simulator, Exponential comparison distribution

ARSR Acceptance/Rejection Simulator, Recursive truncated normal comparison distribution

CFS Crude Frequency Simulator

DCS Deak chi-square Simulator

GHK Geweke-Hajivassiliou-Keane Simulator

GSS Gibbs Sampler Simulator

NISE Normal Importance Sampling Simulator, Exponential comparison distribution

NIST Normal Importance Sampling Simulator, Truncated normal comparison distribution

PCF Parabolic Cylinder Function Simulator

PKFS Polynomial Kernel Smoothed Frequency Simulator

SDS Stern Decomposition Simulator

SUS Sequential Unbiased Simulator

In the FORTRAN implementation, two additional inputs are required, MMAX and NRMAX, specifying the maximum values of M and NR allocated at compilation time.

The programs include code for all statistical functions, spherical transformations, and antithetics routines that are required by the simulation algorithms, and hence are self-contained.

2 Obtaining the Simulation Code by Anonymous FTP

V.A. Hajivassiliou
LSE
email: vassilis@lse.ac.uk
January 1998


```
ftp site: ariadne.lse.ac.uk
userid: anonymous
password: your email address
directory: pub/simulation

There are three subdirectories under this:
fortran -- contains the fortran code for the 13 simulation routines described
           in Hajivassiliou, McFadden, and Ruud, 1997, "Simulating Normal
           Rectangle Probabilities and their Derivatives: Theoretical and
           Computational Results," Journal of Econometrics
gauss   -- contains the gauss code for the 13 simulation routines described
           in Hajivassiliou, McFadden, and Ruud, 1997, op.cit.
ssml    -- contains actual estimation code using smoothly simulated maximum
           likelihood for various models (see Boersch-Supan and
           Hajivassiliou, 1993, "Smooth Simulators for Normal MNP Choice
           Probabilities" Journal of Econometrics, August)
           Code for specific models is kept under separate subdirectories.
           Currently, only code for the Multinomial Probit model is available,
           which can be found in directory pub/simulation/ssml/mnp.
           This program was written by Axel Boersch-Supan of the University
           of Mannheim.  Vassilis Hajivassiliou ported it to UNIX and added
           several enhancements.

For example, to get the file "simprog.gcf" containing the gauss code,
you should issue the following:

ftp ariadne.lse.ac.uk
anonymous
whoever@whereever
cd pub/simulation/gauss
```

```
get simprog.gcf
quit
```

The PostScript code for some of my papers is kept in the directory

pub/papers/ps

and the Adobe Acrobat (PDF) code for them in directory

pub/papers/hplj

3 Description of Procedures in Library

/*+++

A. Overview

This section contains a series of GAUSS procedures for simulation of multivariate normal rectangle probabilities, and the derivatives of these probabilities with respect to the mean and covariances of the normal distribution. Table 2 indexes the procedures in this section.

TABLE 2. GAUSS PROCEDURES

SECTION	MNEUMONIC*	DESCRIPTION
I	CFS	Crude Frequency Simulator
J	NIS	Normal Importance Sampling Simulator Procs NISE and NIST use exponential and truncated normal comparison distributions, respectively
K	KFS	Polynomial Kernel Smoothed Frequency Simulator The proc is PKFS
L	SDS	Stern Decomposition Simulator
M	GHK	Geweke-Hajivassiliou-Keane Simulator
N	PCF	Parabolic Cylinder Function Simulator
O	DCS	Deak chi-square Simulator
P	ARS	Acceptance/Rejection Simulator Procs ARSE and ARSR use exponential and recursive truncated normal comparison distributions, respectively
Q	GSS	Gibbs Sampler Simulator
R	AUS	Approximate (OR Sequential) Unbiased Simulator
S	SUS	Sequential Unbiased Simulator
T	LIB_A	Statistical Functions
U	LIB_B	Spherical Transformations and Antithetics
V	LIB_C	Miscellaneous Utilities

* The procs are located in libraries named *.G, where '*' denotes the mneumonic.

Each of the simulator procedures requires the following standard inputs; the interpretation of some inputs may vary from routine to routine, and not all are used in some routines:

M	Dimension of the multivariate normal
MU	Mean of multivariate normal, a M x 1 vector
W	Covariance matrix of multivariate normal, a M x M array
WI	Inverse of W
C	Lower triangular Cholesky factor of W, a M x M array
A	Lower bound of rectangle, a M x 1 vector [When the lower bound is -infinity, set A = (-1.0E10)*ONES(M,1)]
B	Upper bound of rectangle, a M x 1 vector [When the upper bound is +infinity, set B = (+1.0E10)*ONES(M,1)]
R	Number of repetitions
U	Random variates, a M x R array
PARM	Parameters and constants for the simulation

The simulators all return {P,HU,HC}, where P is the scalar rectangle probability, HU is the (M+1) x (M+1) array of unconditional partial moments (6), and HC is the (M+1) x (M+1) array of conditional moments (7). Parts of the output not provided by a simulator are set to -999.

Interpretations of input parameters

ARSR,ARSE

These procs assume that PARM[.,1] contains seeds for the uniform random number generator RNDUS called by the routines; there is one seed for each repetition R. PARM[R+1,1] is assumed to contain a censoring limit for rejection.

ARSR and ARSE use U = UUBIG; (uniforms)

AUS

The proc assumes that PARM(.,1) contains a vector of R seeds for the normal random number generator RNDNS, PARM(1,2) contains the number of initial GHK simulators of P used in the simulation of 1/P, PARM(2,2) contains the maximum number of trials using the crude frequency simulator that are made before the process is censored. If PARM(2,2) is made large (say 1.e50), then this is computationally the same as the Sequential Unbiased Simulator (SUS). PARM(3,2) contains a seed for RNDUS.

```
AUS uses  U = UUBIG[.,1:R];                                       (uniforms)
```

CFS

This proc does not use PARM.

```
CFS uses  U = UNBIG[.,1:R]                                (standard normals)
```

DCS

This proc assumes that PARM(1,1) = DET(W).

```
DCS uses  U = USBIG[.,1:R];                (random normal grid on unit sphere)
```

GHK

This proc does not use PARM.

```
GHK uses  U = UUBIG[.,1:R];                                       (uniforms)
```

GSS

It assumes that PARM[1,1] gives the number of rounds to be used in the construction of the random draws, JL.

```
GSS uses  U = UUBIG[.,1:JL];                                      (uniforms)
```

NISE,NIST

These procs do not use PARM.

```
NISE and NIST use  U = UUBIG[.,1:R];                              (uniforms)
```

PCF

This proc assumes that PARM(1,1) = DET(W).

```
PCF uses  U = USSBIG[.,1:R];   (random normal grid on unit sphere and orthant)
```

PKF

```
---
This proc does not use W.
It assumes that PARM[1,1] contains a window width parameter.
KFS uses  U = UNBIG[.,1:R];                                 (standard normals)

SDS
---
The SDS proc assumes that PARM[1,1] contains a scalar l > 0 such that
W-lI is positive definite.
SDS uses  U = UNBIG[.,1:R];                                 (standard normals)
Non_standard feature:
--------------------
SDS assumes that C is a Cholesky factor of W-lI, not of W.

SUS
---
This procedure is the same the Asymptotically Unbiased Simulator (AUS)
with PARM(2,2) made large (1.e50).
The proc assumes that PARM(.,1) contains a vector of R seeds for the
normal random number generator RNDNS, PARM(1,2) contains the number of
initial GHK simulators of P used in the simulation of 1/P, PARM(2,2) contains
the maximum number of trials using the crude frequency simulator that are
made before the process is censored.  This is made very large.
PARM(3,2) contains a seed for RNDUS.
SUS uses  U = UUBIG[.,1:R];                                         (uniforms)

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     Test Harness
     The following program provides a simple harness for testing the procs
above.  A one-factor model is used to generate probabilities, with gaussian
quadrature used for evaluation.  The accuracy of this quadrature can be
checked against symmetric cases where the probabilities are known.
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++/
/* Accumulates results */
proc 2 = sim(m,mu,w,wi,c,a,b,r,u,parm);
```

```
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++/
/* setupfil procedure */
proc 4 = setupfil(m,resfil,meth,filreset,nfactors);
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++/
/* reads 2nd row-vector of results from existing file of past runs */
proc resold(fname);
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++/
/* saves the row-vector of results into specified file (which must exist) */
proc resnew(rfilmeth,results);
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++/
/* calculates number of time-equivalent simulations for endogenous timings */
proc endogtim(rsimbig,hseclim,etime,r,rsimtry,timecfs);
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++/
/* examines whether a key has been hit while a MC repetition was being
   carried out, and takes appropriate action */
proc 0=examkey();
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  I. Crude Frequency Simulator (CFS)
     This simulator assumes that U is an array of standard normal variates,
independent within each column, and either independent or antithetic across
columns.  It does not use PARM.  It returns both HU and HC.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
proc 3 = cfs(m,mu,w,wi,c,a,b,r,u,parm);
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  J. Normal Importance Sampling Simulator (NIS)
     This simulator provides HC = HU/P.  It provides two alternative simu-
lators, corresponding to the exponential (NISE) and independent truncated
normal (NIST) comparison distributions.  These simulators assume U is an
array of uniform random (0,1) variates.  These procs call the routine NDEN
that returns a vector of values of the multivariate normal density.
     Procedure NISE does not use C or PARM.  It calls the bilateral
exponential CDF BEXP and its inverse BEXPI from LIB_A.G.  It guarantees P
positive. Provides both HU and HC.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
```

```
proc 3 = nise(m,mu,w,wi,c,a,b,r,u,parm);
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++/
/*   Procedure NIST guarantees P positive.  It calls the proc RNDT for samp-
ling from univariate truncated normals.*/
proc 3 = nist(m,mu,w,wi,c,a,b,r,u,parm);
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  K. Polynomial Kernel-Smoothed Frequency Simulator (PKF)
     The KFS simulator uses the kernel (31), defined by the function PKER.
It assumes that U is an array of standard normal variates, independent within
rows.  It does not use W.  PARM[1,1] contains a window width parameter.  This
simulator does not return a value for HC.

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
proc 3 = pkfs(m,mu,w,wi,c,a,b,r,u,parm);
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  L. Stern Decomposition Simulator (SDS)
     The SDS proc assumes that PARM[1,1] contains a scalar l > 0 such that
W-lI is positive definite, and assumes that C is a Cholesky factor of W-lI.
It assumes that U is an array of standard normal variates, independent within
each column.

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
proc 3 = sds(m,mu,w,wi,c,a,b,r,u,parm);
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  M. Geweke-Hajivassiliou-Keane Simulator (GHK)
     The GHK procedure assumes that U is an array of uniform [0,1] variates,
independent within columns, and in general independent between columns.
It does not use PARM.  It returns both HU and HC.

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
proc 3 = ghk(m,mu,w,wi,c,a,b,r,u,parm);
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  N. Parabolic Cylinder Function Simulator (PCF)
     The PCF proc assumes that U is an array whose column vectors are
```

```
points in the intersection of the unit sphere and the negative orthant,
and that PARM[1,1] = DET(W).

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
proc 3 = pcf(m,mu,w,wi,c,a,b,r,u,parm);
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  O. Deak Chi-Square Simulator (DCS)

     The DCS assumes that U is an array whose columns are points in the
unit sphere, antithetic across columns, and that PARM[1,1] = DET(W).

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
proc 3 = dcs(m,mu,w,wi,c,a,b,r,u,parm);
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  P. Acceptance/Rejection Simulator (ARS)
     The ARS simulator is given in a version using the truncated bilateral
exponential comparison distribution (ARSE), and in a version using a recursive
truncated comparison distribution (ARSR).  PARM[.,1] is assumed to contain
seeds for the uniform random number generator RNDUS called by the routines;
there is one seed for each repetition R.  PARM[R+1,1] is assumed to contain
a censoring limit for rejection.  Should only compute HC.

The ARS method provides a mechanism for drawing from a conditional
density when transformations from uniform or standard normal variates
are not available.  We want to sample from f(x/a).  We generate x from
g(x).  In our case g(x) = ARSE.  Generate tsi from uniform.
Generate unconditional f(x) and  bound alpha. We accept x if
                tsi < f(x)/(g(x)*alpha)
x has conditional density f(x/a) = HC.  We set x = VD

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
proc 3 = arse(m,mu,w,wi,c,a,b,r,u,parm);
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
/* The following uses a recursive truncated */
/* comparison distribution for g(x)         */
proc 3 = arsr(m,mu,w,wi,c,a,b,r,u,parm);
```

```
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  Q. Gibbs's Sampler Simulator (GSS)
     The GSS simulates HC, but does not simulate P or HU.  It assumes
that PARM[1,1] gives the number of rounds to be used in the construction
of the random draws (JL). It assumes that U is an MxJL array
of uniform [0,1] random numbers.  It returns only HC.

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
proc 3 = gss(m,mu,w,wi,c,a,b,r,u,parm);
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  S. Asymptotically Unbiased Simulator (AUS)
     The AUS simulator is obtained as the product of an unbiased simulator of
H and an asymptotically unbiased simulator of 1/P.  The proc is defined with
the GHK simulator used for H and for the initial independent simulations of
P(B)/P(A), where A is the event that the first component is in the rectangle
bounds.  The proc assumes that PARM(.,1) contains a vector of R seeds for the
normal random number generator RNDNS, PARM(1,2) contains the number of
initial GHK simulators of P used in the simulation of 1/P, PARM(2,2) contains
the maximum number of trials using the crude frequency simulator that are
made before the process is censored.  If PARM(2,2) is made large (say 1.e50),
then this is computationally the same as the Sequential Unbiased Simulator
(SUS).  PARM(3,2) contains a seed for RNDUS.  Gives HU and HC.

This does continous parameter estimation of 1/p.

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
proc 3 = aus(m,mu,w,wi,c,a,b,r,u,parm);
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
/* Gives unbiased simulator of H=HU */
proc 3 = fghk(i,m,mu,w,wi,c,a,b,r,seed);
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  R. Sequentially Unbiased Simulator (SUS)
     The SUS simulator is obtained as the product of an unbiased simulator of
H and an unbiased simulator of 1/P.
Computationally, if PARM(2,2) is made large (say 1.e50), SUS is the same as
```

```
the Asymptotically Unbiased Simulator (AUS).

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
proc 3 = sus(m,mu,w,wi,c,a,b,r,u,parm);
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  T. Statistical Functions (LIB_A)
     This paragraph contains a series of procedures that return
statistical functions.  NDEN is the multivariate normal density,
CDFNI and CDFINVN are two different algorithms for calculating the
inverse of the cumulative standard univariate normal, RNDT is a proc
for drawing an array from truncated independent normals that have the
same means and variances as the multivariate density, and FI is a
proc that returns partial moments of the standard normal.
CDFINVN appears faster and more accurate than CDFNI.

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
proc 1 = nden(m,r,vd,wi,c);     @MULTIVARIATE NORMAL DENSITY, DIMENSION M@
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++/
/***CDFINVN used instead of CDFNI:
proc 1 = cdfni(p);            @INVERSE CUMULATIVE NORMAL@
/*
** CDFINVN --   computes the inverse normal cdf. The algorithm used is taken
**              from Kennedy and Gentle, STATISTICAL COMPUTING, p. 95.
**
** vah 22:57:35.78, Sun, Jul 23, 1989
** Taken intact from V149b MAXPROC3.ARC
**
**              xp=cdfinvn(p);
**
**             INPUT
**              p -- nx1 vector of probabilities (that is, 0 < p < 1)
**             OUTPUT
**             xp -- the corresponding vector of values such that: cdfn(xp)=p;
**
**              This algorithm seems to be very accurate.  It is
**              rated to be accurate to 1e-7.  However, it appears
**              to be substantially more accurate -- approximately
```

```
**              1e-10.
**
*/
proc 1=cdfinvn(p);
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++/
proc 2 = rndt(m,r,u,w,a,b);
/* RETURNS M BY R ARRAY OF  INDEP.  TRUNCATED  NORMAL  DEVIATES (IN V) AND
ASSOCIATED R BY 1 VECTOR OF LEVELS OF DENSITY (IN Y), DRAWN FROM INDEP.
NORMALS WITH SAME VARIANCES AS THE MULTIVARIATE NORMAL DENSITY.  U
IS ASSUMED TO BE A M BY R ARRAY OF UNIFORM (0,1) VARIATES. */
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++/
proc 3 = fi(a,b,kap);
/*FIRST THREE PARTIAL MOMENTS OF (Y-KAP)^K FROM A TO B FOR STANDARD NORMAL.
CALLS CDFN AND PDFN*/
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

U. Spherical Transformations and Antithetics (LIB_B)

RNDBASE is a routine that draws a random basis in K space, where K is the dimension of the random coefficient vector. ANTITH is a routine that creates a "grid" of antithetic points S on the unit sphere, starting from a random basis provided by RNDBASE. These routines will ordinarily be called for each observation. To avoid "chatter" when parameter values are changed, the seed for the random number generator for each observation should be stored so that the same outputs can be regenerated in iteration to parameter estimates.

RNDBASE takes as input a dimension K and returns a K by K array V that is column orthonormal and random, bordered by a column of random lengths of K-dim. normal random vectors. SEED1 is a seed for the random number generator.

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
/*     Starting from a random basis B, obtained from RNDBASE with seed SEED,
the ANTITH routine returns an array A with columns of length K that have an
antithetic property.  R is an integer that determines the number of columns M
as follows:  If T <= 1, then M = 2K, and A contains the columns of B and
their antipodes.  If T > 1, then M = 2K(1+(T-1)(K-1)), obtained by taking 4T
equally spaced points on each of the K(k-1)/2 great circles through pairs of
directions in the random basis.*/
```

```
proc antith(k,t,v0);
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  V. Miscellaneous Utilities (LIB_C)
     This paragraph gives six utility routines:
      HSEC1990 to return the number of 1/100th secs. since Jan 1, 1990;
      NUMTOSTR to turn a number into a string;
      GPRINTM to print a matrix in a nice format;
      FMTSDT to reset the default printing format
      FILEXIST to find out if named file exists; and
      getNandS to read a file of parameters.

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
proc hsec1990();
/*
** HSEC1990.G
**
** x=hsec1990();
**
** vah
**
** Returns number of hundredth's of a second since Jan 1, 1990
**
*/
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
proc numtostr(number,lenstr);
/*
** NUMTOSTR.G
**
** s=numtostr(number,lenstr);
**
** vah 12:05:18.07, Wed, Jul 19, 1989
** Converts a number to a string reading the same number.
** Returns a (LENSTR+1)-character string
**  If lenstr=0, the number of characters appropriate to handle the full
**  number will be returned.
**  If requested lenstr is longer than needed, pad with leading zeros.
**
```

```
**  "p" or "n" will be appended, depending on the sign of the number
*/
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++/
proc (0)=gprintm(mat,rowmax,colmax);
/*
** GPRINTM.G
**
** vah 12:54:40.03, Wed, Jul 19, 1989
**
** call gprintm(mat,rowmax,colmax);
**
** Prints matrix MAT in ROWMAX by COLMAX blocks as necessary.
** i and j headings are also printed.
**
** If ROWMAX le 0, ROWMAX = 22
** If COLMAX le 0, COLMAX = 6
*/
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++/
proc 0=fmtsdt;
/*
** FMTSDT.G
**
**  vah 13:38:12.84, Wed, Nov 16, 1988
**
**  call fmtsdt();
**
**  Resets the default format for printing
**
*/
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++/
proc filexist(fname);
/*
** FILEXIST.G
**
** vah 10:33 pm, Friday, November 27, 1987
**
** code = filexist(fname);
**
```

```
** Checks for the existence of file FNAME, which may contain wildcards.
** Returns 1 if such file(s) exist(s), 0 if not.
*/
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++/
proc getNandS(fname);
/*
** GETNANDS.G
**
** vah 22:23:03.05, Sat, Sep 23, 1989
**
** Usage: NVEC=getNandS(fname);
**
** Reads file "fname" with the following format:
** number of numbers (NUMNUM)
** number_of_numbers lines for numbers
** number of strings (NUMSTR: must not exceed 10)
** number_of_strings lines for strings
**
** Returns NVEC, the NUMNUM vector of numbers
**
** Also places in memory the NUMSTR strings line1 - lineNUMSTR
**
*/
```