

Simulation and Bootstrapping in the SAS system: A tutorial using PROC IML

Paul A. Thompson, Ph.D.
Case Western Reserve University
Cleveland, OH 44106

August 11, 1995

Abstract

Simulation and bootstrapping can both be accomplished readily in SAS. The basic ideas of simulation and bootstrapping are defined, and they are illustrated with examples from various SAS procedures. The important notions in simulation concern the control of seeds, the generation of data and the construction of matrices to which have certain forms (orthogonal or with specified covariance structures). When defining a simulation, the model for errors must be specified, to ensure that the process of creating errors is an expression of random deviation only. The process of defining a simulation, using SAS PROC IML, is demonstrated. Some important ideas, such as overall control using a control dataset, are also presented. In the bootstrapping discussion, the basic notions are presented. A simple example, presenting the ideas of bootstrapping, is also shown. Finally, the notion of obtaining information from output after redirecting these to files is discussed.

Introduction

The ideas involved in bootstrapping and simulation are not entirely consonant, yet there are some important communalities which can be examined. In particular, in doing either or both in the the SAS[®] system for data management (SAS) system, quite similar methods may be employed. In fact, when the basic question of the *source of the data to be analyzed* is addressed, code which has certain basic communalities may be used for both simulation and bootstrapping. This talk will examine the basic notions for both bootstrapping and simulation, present code for doing both, discuss the process of setting up a simulation/bootstrap, and demonstrate the method of successive abstraction by which a straight-forward project is gradually turned into a more useful, but far more abstract, macro program.

Random variable control in SAS. Control over seeds is crucial to both methods. First, seeds must be retained and controlled over trials. This is generally easily done, in the SAS system. Here is the basic approach:

1. Use one seed stream for each distinct random data source.
2. Retain seeds over calls in a seed data set. Include the seed dataset at the start of a dataset, and export the final seed at the end of the dataset to the seed dataset.
3. Use the CALL RANSOURCE(seed, result); approach. This approach preserves distinct seed streams and allows the seed to be output at the end of the data step. ransource can include RANUNI, RANNORM, RANPOI, RANBIN, RANCAU, RANEXP, RANGAM, RANTBL and RANTRI (the last two involve probabilities from user-supplied mass functions).

Here is an example:

```
DATA RANGEN(DROP=SD1-SD4) SEEDS(KEEP=SD1-SD4);  
  SET SEEDS; ARRAY RVS RV1-RV4; ARRAY SDS SD1-SD4;  
  DO GRP=1 TO 4; DO SUB=1 TO 20; DO RV=1 TO 4;  
    CALL RANNOR(SDS{I}, RVS{I}); RVS{I}=RVS{I}*3;  
  END; OUTPUT RANGEN; END; END; OUTPUT SEEDS; RUN;
```

In this case, four distinct streams are initialized and independently generated over observations. In addition, the position in the seed stream is maintained over data generation points.

The seed stream must be maintained and controlled to ensure that the random values are obtained from sequential positions in the stream. If this is not done, values are obtained from unknown places along the seed stream. If that is done, inadvertent correlations may be introduced into the generation of random values.

It bears reiterating that use of the "time-of-day" seed is completely unacceptable. This seed is entirely a "black-box," as the seed for the start of the process is entirely chance. Successive tests could be initialized at close points along the seed stream, thus introducing dependencies into the process, or at appropriately distant points along the stream. Since the process is unknown, it is also uncontrolled.

Matrix formulation for random variable generation. There are generally two questions with random variable generation. First, how can a certain covariance matrix be introduced into the generation of normally distributed random variables? Secondly, how can orthogonal vectors of r.v.s be generated?

The matrix X will be a matrix of values, in which k variables are columns and there are n subjects. The matrix multiplication $X'X$ is used to compute the SSCP matrix, rather than xx' approach used to discuss the cross-product of the column random vector x , which is composed of k random variables. The difference between the two methods is a historical peculiarity of the choice of representation in computers of rows as the vector, rather than columns. I choose this method to have a consistent matrix representation of the computation of a SSCP or covariance matrix.

Consider the $n \times k$ matrix X of random variable values for n persons and k variables. If the values have been generated randomly, the values will be distributed $X \sim \mathcal{N}(\mu, \sigma^2 I)$. The expected covariance matrix has the form $\sigma^2 I$, because distinct variables are generated independently, and thus $\sigma_{ij} = 0$, where $i \neq j$. Often, $\mu = 0$.

Sometimes, we wish to impose a correlational structure on the random matrix X , such that either $X'X = C$ (that is, the covariance of the data matrix has the form C) or that $\mathcal{E}(X'X) = C$ (that is, the covariance matrix of the expected value of the data-generating process has the form C , but the exact covariance matrix of the data itself may not have this form). The Cholesky factorization of $C = LL'$ is used here, where L is a lower triangular orthogonal matrix and C a square and symmetric covariance matrix. After obtaining L , $Y = XL'$ will have the desired covariance structure, as:

$$\mathcal{E}(Y'Y) = L\mathcal{E}(X'X)L' = L\sigma^2 IL' = \sigma^2 LL' = \sigma^2 C. \quad (1)$$

This is true for the expected value of the covariance matrix, and thus is true only in the population. Since $X'X = S^*$ (S^* having small but non-zero off-diagonal elements), Eq. 1 will result in $Y'Y = LS^*L'$. If the covariance matrix C is desired in actuality, rather than in expected value terms, the approach defined in Eq. 1 is modified by orthonormalizing X using a method such as the Gram-Schmidt technique. Then, we obtain $X'X = I$; then Eq. 1 is true in actuality as well as in expected value.

In the Gram-Schmidt orthonormalization, we have $X = PT$, where X is an arbitrary matrix, P is the desired orthonormalized matrix and T is a triangular transformation matrix. After using the orthonormalization process on X and producing P , we use the matrix P for future analyses. Then:

$$\text{If } X = PT \text{ and } Y = PL', \text{ then } Y'Y = LP'PL' = LL' = C \quad (2)$$

In all of these discussions, we have been making a tacit assumption that $X \sim \mathcal{N}$. This is important, as linear combinations of normally distributed random variables are themselves normally distributed. Thus, by transforming $X = PT$ and using P for other work, we can be assured that if $X \sim \mathcal{N}$, then $P \sim \mathcal{N}$. Similarly, if we use $Y = PL'$, then we can also know that $Y \sim \mathcal{N}$. This does not hold for other distributions. In fact, if such transformations are employed, the Central Limit Theorem indicates that $X \sim ?$, then $c'X \sim \mathcal{N}$. Thus, this distribution is preserved only when $X \sim \mathcal{N}$. Some discussion of the generation of random values which are correlated but non-normal can be found in the literature (Johnson and Ramberg 1977, Kennedy Jr. and Gentle 1980, T and Hammond 1975). However, for the purposes of this discussion, these will be ignored for purposes of simplicity and clarity.

Linear model for generation of data. When defining the generation of random values, it is important to consider the linear model for the random values. In using simulation methods, one may be interested in several levels of complexity. Eq. 3 defines a model for the one-factor ANOVA model.

$$Y_{ij} = \alpha_i + \epsilon_{ij} \text{ where } \epsilon_{ij} \sim \mathcal{N}(0, \sigma^2), \text{ or in matrix terms, } Y = \alpha + \epsilon \text{ where } \epsilon \sim \mathcal{N}(0, \sigma^2 I) \quad (3)$$

In many simulations, $\alpha_i = 0, \forall i$. In that case, all objects would differ only on the basis of random error values, and the performance of the model could be studied in that context. The model is stated in matrix terms as on the right side of Eq. 3 (which may optionally involve multiple dependent variables). In this case, errors are distributed as indicated in Eq. 3. The assumption about errors made in Eq. 3 is restrictive, and probably needs reconsideration for many real problems.

Eq. 4 presents a somewhat more complex type of error, for a model usually termed the "split-plot" in experimental design:

$$Y_{iklm} = \alpha_i + \gamma_k + \alpha\gamma_{ik} + \pi_{l(i)} + \gamma\pi_{kl(i)} + \epsilon_{klm(i)} \text{ where } \epsilon_{klm(i)} \sim \mathcal{N}(0, \Sigma), \text{ or alternatively, } Y = \alpha + \epsilon \quad (4)$$

For the moment, the model will be simplified by considering the model for one conceptual dependent variable only (although this variable will be observed under several conditions). In this case, we have a two-factor design, in which α is a *between-subject* term and γ is a *within-subject* term. The factor π is a subject term (and is treated as a factor as well, but is not tested statistically), which represents a nuisance parameter which must be considered when estimating the model. In the model, the parenthesized subscripts indicate that some terms are nested within others; subjects are observed at only one level of the between-subjects term, while they are seen at all levels of the within.

The model simplifies considerably when it is written in matrix terms (as on the right side of Eq. 4), because all values for the repeated term γ have been incorporated into the matrix Y as separate dependent variables. This is the standard approach for the "multivariate setup" for the split-plot design. In addition, there are no longer multiple rows per person, but rather g columns (where there are g levels of γ). These columns may represent observations of each person under three conditions, for instance. In that case, the covariance of the values for each person must be considered, and will very likely not be $\sigma^2 I$, because it is quite likely that the values for the variable at time t are related to values at time $t + 1$. In fact, a whole family of forms of the covariance model exist; these include auto-correlated, banded covariance, factor-analytic, auto-regressive and unstructured.

In the case of Eq. 4, we may further consider the matrix ϵ . We may consider the errors in terms of a component ϵ^a for which $\epsilon^{a'}\epsilon^a = \sigma^2 I$, a component ϵ^b for which $\epsilon^{b'}\epsilon^b = \Sigma$, a person-specific component ϵ^c (a column vector of random disturbances for each person) and a time-specific component $\epsilon^{d'}$ (a row vector of random disturbances for each time point). Then:

$$\epsilon = a\epsilon^a + b\epsilon^b + c\epsilon^c 1' + d1\epsilon^{d'} \quad (5)$$

Breaking the error term out in this fashion allows the simulation to specify the error in very precise ways, and leads directly to characterizing the various sources of error in ways which allows their better understanding. In addition, the weights a, b and c can be easily manipulated to enable the different sources of variation to contribute independently. Using this approach enables a single method for generation of random variables which is easily converted to a SAS macro formulation in the SAS/IML[®] system (SAS/IML).

Characterizing the contribution of each component. The various components described in Eq. 5 may also be considered in terms of variance components. By computing a variance attributable to each segment of the error term, the overall influence of that portion of the data can be assessed. It is a check of the process and important for the documentation of the results. See Figure 4 for an example of this technique.

SAS code to generate random deviates

Although the presentation of a complete example will not be possible for reasons of space, some important portions of the code for random value generation are presented here. These sections can be put together with other sections to make a complete data generation system for a simulation.

In previous work with the SAS system in the simulation area, I have produced code for generation of values in both SAS data steps and in the SAS/IML procedure. I strongly urge the use of the SAS/IML procedure, because the usual need for considering the covariance matrix of the data can be addressed directly and naturally in that program (it is a matrix algebra implementation, after all).

IML code for normal variant generation. SAS/IML code to implement the equations above is presented in Figure 1. The method for generating data is defined by Eq. 4, with the factorization of error components as defined in Eq. 5. Thus, there are three components returned (see Lines 1-6, and associated comment). The matrix `vlx` is orthogonalized in Line 12. The module `rangen` is called twice, once to obtain components $\epsilon^a, \epsilon^c, \epsilon^d$ and once to obtain component ϵ^b . `rangen` returns both the original random values (`vlx`) and orthogonalized values (`svlx`). Note that seeds are handled in SAS/IML variable `sds` and are returned in this variable at the end of the execution of the module.

```

start rangen(ns,ng,nt,vlx,svlx,vlc,vlr,sds);
/* ***** */
/* RANGEN - generates random values of various types */
/* ns:# subjects|ng:# groups|nt:# time points|vlx:random values */
/* svlx:standardized r v|vlc:column component|vlr:row part|sds:seed */
5 /* ***** */
vlx=j(ns*ng,nt);svlx=j(ns*ng,nt);vlr=j(ns*ng,1);vlc=j(ng,nt);cl=j(1,nt);rx=1;
sds=sds[1,(1:nt)];row=j(1,nt);sdr=sds[1,nt+1];sdc=sds[1,nt+2];cadd=j(ns*ng,1);
do j=1 to ng;do i=1 to ns;k=i+(j-1)*ns; /* For each group and subject ... */
10 call rannor(sdv,row);vlx[k,]=row; /* Overall matrix of rv */
call rannor(sdr,rx);vlr[k,]=rx;end; /* Row value */
call rannor(sdc,cl);k=(j-1)*ns+1; /* Column random value */
call gsort(zpv,t,xv,vlx[k:j*ns,]); /* Orthogonalize matrix */
svlx[k:j*ns,]=zpv;cadd[k:j*ns,1]=j(ns,1,j);vlc[j,]=cl;
15 end;sds[1,(1:nt)]=sdv;row=j(1,nt);sds[1,nt+1]=sdr;sds[1,nt+2]=sdc;
vlx=cadd||vlx;svlx=cadd||svlx; finish; /* Retain the group ident */

```

Figure 1: IML code to generate normally distributed random variables

IML code for correlation matrix generation. SAS/IML code for generating correlation matrices is presented in Figure 2. Parameters are defined and the call shown in Lines 17-22. The section for generation of compound symmetric correlation matrices is shown in Lines 25-26. Here the values are generated from a uniform distribution, and is done to ensure that all correlations are equal. For the unstructured correlation matrix, values are generated from the uniform distribution and placed into the correlation matrix in Lines 27-29. The code in Lines 32-35 ensures that the sum of squared correlations is equal for all time points, and then returns the values (including sign) to the correlations. Finally, Lines 36-38 ensures that the matrix is positive definite, by computing eigenvalues and replacing negative eigenvalues with small positive values, to allow the Cholesky factorization to be performed.

```

start covgen(hcov, ct, equal, nt, ng, sds, cent);
/* ***** */
/* COVGEN - generates covariance matrices */
20 /* hcov:Cholesky factor of cvx||cvtype:1(CS) 2(UN)||equal:All grps equal? */
/* nt:# times|ng:grps|sds:seeds|cvx:cov matrix|cent:centering matrix */
/* ***** */
vx=j(1,nt); vadd=nt+1;vadd=vadd'/30+.10;hcov=j(ng*nt,nt);sd=sds[1,nt];
do j=1 to ng; if (j = 1 | ^equal) then do;
25 if (ct=1) then do;call ranuni(sd,vx); /* Compound symmetry error here */
cvx=j(nt,nt,(vx[1,1])+i(nt)-i(nt)*vx[1,1]); end;
if (ct=2) then do;cvx=j(nt,nt,1); /* Unstructured error matrix here */
do i=1 to nt-1;call ranuni(sd, vx); /* Make a symmetric matrix */
vx=vx*1.5-.55;cvx[i,i+1:nt]=vx[,i+1:nt]; cvx[i+1:nt,i]=vx[,i+1:nt]';end;
30 scv=sign(cvx); cvx=cvx##2; /* Next section - Rsq for all = */
cvx=cvx-diag(cvx);
do i=1 to 10; /* Make cov matrix sum to constant */
cvx=cent*cvx*cent;colsum=(cvx[,+]-vecdiag(cvx))';cvx=cvx-diag(cvx);end;
cvx=scv#sqrt(cvx-min(cvx)+.05);end; /* Ensure that lowest is non-zero */
35 do while(abs(trace(cvx)-nt)>.001); cvx=cvx-diag(cvx)+i(nt);
call eigen(vl, vk, cvx); /* Compute eigenvalues - make NND */
if (max(loc(vl>0))<nt) then vl=vl[loc(vl>0),]/vadd[loc(vl<=0),];
cvx=vk*diag(vl)*vk';end;end; /* Replace - eigenvalues with + val */
hcov[(j-1)*nt+1:j*nt,]=root(cvx);end;sds[1,1:nt]=sd; finish;

```

Figure 2: IML code to generate random correlation matrices

IML code for generating missing values. SAS/IML code for the generation of missing data is presented in Figure 3. Parameters are defined and the call shown in Lines 40-47. Missing values are computed by generating a

```

40 start missgen(ns,ng,nt,sds,lev,rvx,type);
  /* ***** */
  /* MISSGEN - generates patterns for missing values */
  /* ns:# subjects|ng:# groups|nt:# time points|sds:sds */
  /* lev:level of missing|rrvx:return value|type:1(MCAR) 2(MTEOL) */
45 /* ***** */
  mult={1 2 4 8 16}'; zvx=j(1,nt-1,0); rvx=j(ns*ng,nt+1,0);
  vlz=j(1,nt-1); sdv=sds[1,1:nt-1];
  do j=1 to ng; do i=1 to ns; /* For the number of groups */
    call ranuni(sdv, vlz); /* Generate random values */
50 if (type=1) then vlz=(vlz>=lev); /* Binary-ize them using lev */
    if (type=2) then do; vlz=(vlz>=lev*.9);
    if (vlz[,+] < nt-1) then do; /* Make missing to end of line */
      vlz[,min(loc(vlz=0)):nt-1]=zvx[,min(loc(vlz=0)):nt-1];
    end; end;
55 rvx[(j-1)*ns+i,]=1||vlz||loc({1}||vlz)[,<];
    end; end;rrvx=rvx[,2:nt]*mult||rvx[,nt+1];sds[1,1:nt-1]=sdv; finish;

```

Figure 3: IML code to generate missing terms

uniform random value and comparing it to a cutoff (the first observation, for specific purposes, is never allowed to be missing). If type=1, single values are set missing. If type=2, values are set missing to the end of the set of time points if any value is missing (this is like dropping out of the trial). Finally, the missing values are converted into a decimal value which is later decoded using bit-masking techniques, as in the following example:

```
DATA UNCODE;SET VALUE;ARRAY VALS VAL1-VAL6;IF (BITV = "1....."B) THEN VAL1=.;
```

IML code for computing random values. SAS/IML code to combine random values is presented in Figure 4.

```

start makmat(hcov,ng,nt,ranv,ranva,ranvb,vra,vca,vrb,varacc,vmn,addbar);
  /* ***** */
  /* MAKMAT - combine random values into error term for simulation */
60 /* hcov:Chol factor of cov|ng:# groups|nt:# time points|ranv:final rv */
  /* ranva:first matrix(eps a)||ranvb:second matrix (eps b)||vra:row vals */
  /* vca:column vals||varacc:variance decomposition here */
  /* ***** */
  fa=&_fa.;fb=&_fb.;fc=&_fc.;fd=&_fd.; free ranv;
65 do i=1 to ng; /* For each group ... */
    hsub=hcov[(i-1)*nt+1:i*nt,];covx=hsub*hsub';
    sva=ranva[loc(ranva[,1]=i),2:ncol(ranva)]; svb=ranvb[loc(ranvb[,1]=i),2:ncol(ranva)];
    sra=vra[loc(ranva[,1]=i),]; /* Here are row (person) values */
    sca=vca[i,]; /* Here are column (time) values */
70 zsra=sra*j(1,nt);
    zsca=j(nrow(sva),1,1)*vca[i,];
    zsva=(sva+(fa+fb)/fa*fc*zsra)*hsub; /* Add in row params, then transf */
    if (addbar) then zsva= zsva+j(nrow(sva),1,1)*vmn[i,];
    zsvb=svb*fb+zsra*fc*(fa+fb)/fb; /* Uncorrelated part */
75 ranvy=zsva*fa+zsvb*fb+zsca*fd; /* Final value */
    run corrs(ranvy, "chk", corr); /* Compute correlations for data */
    diff=abs(corr-covx);*print diff; /* Compare these against target in */
    div=((ncol(corr)**2)-ncol(corr)); chkv=diff[:,,];chkx=diff[##,+]/div;
    a=shape(ranvy,1); /* Compute overall variance for grp*/
80 a=a-a[,,]*j(1,ncol(a),1); ssov=a*a';
    b=ranvy[,,]; b=b-b[:,,]*j(nrow(b),1,1);ssbtw=b'*b*ncol(ranvy);
    w=ranvy-ranvy[:,,]*j(1,ncol(ranvy),1); sswin=vecdiag(w*w')[+,+];
    varacc=varacc//((ssov||ssbtw||sswin||chkv||chkx);ranv=ranv//ranvy;
    end; ranv=ranv[,1]||ranv;finish;

```

Figure 4: IML code to combine random values

Parameters are defined and the call shown in Lines 57-63. The actual computation of the ϵ value from the components is shown in Lines 66-75. In most cases for the evaluations which I am interested in, $_fd=0$, which is the parameter for the differences between the time points (and thus this component of difference is not included in the total). However, for the purposes of evaluation of Type II performance (detection of actual differences at the appropriate level of frequency), this value would be set to some non-zero number. In terms of variance decomposition, Lines 75-83 includes code to compute the sum of squares within (over time points) and sum of squares between (groups, that is), which are two sources of variance for the design. Others are possible, but not employed, for this design.

Integrating the modules together. SAS/IML code which runs the process is presented in Figure 5.

```

65 start overall(ns, ng, nt, reps, sds, cvtype, equal);
   cnm={"testgrp" "ind" "grp" %do _i=1 %to &nt.; "yv&i." %end;"LM" "LL" "MM" "ML" "HM" "HL"
       "LMLast" "LLLast" "MMLast" "MLLast" "HMLast" "HLLast"};
   varacc=j(1,5);indv=1:ns;do ir=1 to ng;indx=indx//indv;end;cent=(i(nt)-j(nt,1,1)*j(1,nt,1)/(nt-1));
   do ir=1 to reps;
85     run rangen(ns,ng,nt,ranva,sxpa,vca,vra,vmn,sds,1);
       run rangen(ns,ng,nt,ranvb,sxpb,vcb,vrb,vxx,sds,0);
       run covgen(hcov, cvtype, equal, nt, ng,sds, covx, cent);
       run makmat(hcov, ng, nt, ranv, sxpa, ranvb, vra, vca,vrb,varacc,vmn,0);
       run missgen(ns,ng,nt,sds,.0375,rvxlm,1); run missgen(ns,ng,nt,sds,.0375,rvxll,2);
95     run missgen(ns,ng,nt,sds,.075,rvxmm,1); run missgen(ns,ng,nt,sds,.075,rvxml,2);
       run missgen(ns,ng,nt,sds,.150,rvxhm,1); run missgen(ns,ng,nt,sds,.150,rvxhl,2);
       ranv=j(nrow(ranv),1,ir)||indx||ranv||rvxlm||rvxll||rvxmm||rvxml||rvxhm||rvxhl;
       if (ir = 1) then create &_base..&_dset. from ranv [colname=cnm];
       append from ranv [colname=cnm];
100    end;
       varacc=varacc[2:ncol(varacc),];
       varacc=varacc||((varacc[,2]/varacc[,1]) || (varacc[,3]/varacc[,1]));
       cnxa={"fa" "fb" "fc" "fd"};
       cnxb={"total" "btw" "win" "cordev" "cordev2" "b_to_tot" "w_to_tot"};
105    cnx=cnxa||cnxb;
       print varacc[colname=cnxb];
       sumvar=varacc[.,]; sumvar={&_fa. &_fb. &_fc. &_fd.}||sumvar;
       print sumvar [colname=cnx];
       close &_base..&_dset.;finish;
110 use &_base..&_seed.; read all into sds[colname=sdnm]; close mxx.seed1;
       run overall(&_ns,&_ng,&_nt,&_reps,sds,&_cvtype,&_equal);
       create &_base..&_seed. from sds [colname=sdnm]; append from sds; quit;

```

Figure 5: IML code to run the whole process

Parameters are defined and the call shown in Line 85 Lines 89-93 show the calls to the various modules to define random values and combine them. The missing values are generated in Lines 93-97. Finally, Lines 110-112 show the skeleton of the overall macro which runs the entire process, although certain details are excluded. While the overall call of the macro is not presented, it is somewhat self-evident, from other variables defined in other places.

Putting it all together. The five sections of the SAS/IML development above are the heart of the simulation. After this, the choice of program to run for the analysis, the collection of statistics, the accumulation of such values, and the final evaluation of the results are somewhat less interesting. However, there are one or two important other ideas which I have found to be useful in my simulations.

1. Embed the entire process in a macro. By placing the entire simulation in a macro, the parameters for the simulation can be changed easily at one point, and careful macro design will allow that change to be reflected to all other places in the analysis.
2. Use a control dataset. In running simulations, I find that they take a long time. In one case (results of which will be presented at the MWSUG 1995 meeting), one analysis for the simulation (500 replications) takes 2-5 hours, there are 25 different portions for the analysis and there are 6 different conditions for each of these 25. Thus, a basic analysis here takes 2-4 weeks of continuous use of a PC clone. When other work interferes with the simulation, it can be difficult to restart the analysis properly, and to determine what portions of the analysis have been successfully completed. The solution here lies in the use of a control dataset. This dataset

contains all important parameters, and is consulted before beginning each step. The variable DONE controls execution; when DONE=0, the step is run, and at the end of the step, I set DONE=1. The method works something like this:

```
DATA CHK;SET CHKSET;
IF (CONDA=&_conda. & CONDB=&_condb. & CONDC=&_condc.) THEN
  CALL SYMPUT("_go_on", PUT(DONE,1.));
RUN;
%if (&_go_on.) %then %do;
stuff...stuff...stuff...stuff...stuff...stuff...end of stuff...get stuff and save it
DATA CHK;SET CHKSET; IF (CONDA=&_conda. & CONDB=&_condb. & CONDC=&_condc.) THEN DONE=1;RUN;
%end;
```

For those who are interested, I will be happy to supply copies of code which perform various types of simulations, which illustrate some of the ideas discussed here, and present some more complex techniques as well.

Bootstrapping methods in SAS

Basic notions of bootstrapping. The basic ideas of bootstrapping are discussed in a variety of sources (Diaconis and Efron 1983, Efron 1982, Efron and Tibshirani 1986, Thompson 1991). The essential notion of the bootstrap lies in using the natural variation of the actual sample to obtain some approximation of the variation inherent in a parameter. Efron (1982) discusses many types of bootstrap methods. In this paper, only the most simple and straight-forward bootstrap method is used.

Consider a parameter θ , which is some unobserved function G of the population. When the parameter is estimated, a sample (termed $\mathcal{F}$) is drawn from the population, and Function G is computed using the sample. This is also called a *functional* (Efron and Tibshirani 1986) statistic:

$$\hat{\theta}_{\mathcal{F}} = G(x_1, x_2, \dots, x_n). \quad (6)$$

Since the statistic will vary from sample to sample, it is important to consider the *variability* of the parameter estimate. While the variability of many parameters is obtained by theoretical approaches, the variability of others is more difficult to obtain. In such cases, bootstrapping approximations may be interesting.

To set up a bootstrap, the observed sample $\mathcal{F}$ is the source of a number of *resamples* or bootstrap samples. These resamples $\mathcal{F}_i^*$ are obtained by sampling n observations *with replacement* from the original sample of n cases. Most resamples are different, because sampling is done with replacement (of course, they *may* all be the same, but this is not likely). Then each resample is used to obtain a separate estimate of the parameter $\hat{\theta}_{(i)}^* = G[\mathcal{F}_i^*]$. The estimates will differ in value, as each is obtained from a different sample. The distribution of $\hat{\theta}_{(i)}^*$ is termed a *bootstrap sampling distribution* (Efron 1982).

In all, there are n^n possible resamples. With 2 cases in the sample, there are 4 resamples, with 3 there are 27, with 4 there are 256 and with 5 there are 3,125. With 10 observations, this comes to 10,000,000,000 resamples, which is a large number. With the large number of possible resamples, it is clearly impossible to obtain all possible resamples, when even a modest number of cases are found in the sample. Therefore, the usual practice is to sample a relatively small number of bootstrap samples, on the order of 200-1000 samples.

At that point, several things are done with the bootstrap sampling distribution. Possibly the most interesting uses of this distribution lie in the computation of the standard deviation (the bootstrap standard error estimate) and the various order statistics (the 5th, 10th, 25th, 75th, 90th and 95th percentile points are often used). A variety of uses of bootstrap estimates are demonstrated in Thompson (1991).

The bootstrapping algorithm

In SAS, simple bootstrapping can be done in a simple way. Figure 6 demonstrates the overall approach. In this approach, we begin by generating a dataset containing a number of bootstrap resamples. Seed control is maintained by retaining the seeds in a seed dataset, and then outputting the final seed to the seed file. The number of bootstrap resamples generated here (100) is merely an example, and is not meant to advise the user on the correct number of bootstrap resamples to actually select. The CEIL function produces the integer at or above the decimal chosen, and thus are integers corresponding to the identification of cases in the original sample. It goes without saying that a corresponding ID value must be found in the sample.

The FREQ step is a randomization check. The frequency count should be uniform, and a simple χ^2 test can be used to test the randomization process to ensure that this phase of the process is well designed. After this, the data are sorted by IDN and then merged in with the dataset DATASRC, in which the original sample is stored. After this, the

data are sorted by BOOT. This reorganizes the data in relationship to the bootstrap resamples. Next, the bootstrap process is then run using any SAS procedure which allows the use of the BY variable mechanism (hint: this is a set which is isotonic with the set of SAS procedures). Finally, the dataset OUTF is produced as a standard output dataset from PROC REG. This procedure produces a large number of statistics and routinely places these in SAS datasets. Other approaches are needed, when using procedures which do not produce output datasets, or when the statistic in question is not placed into an output dataset in a convenient manner.

```
DATA BOOTAMP; SET SEEDS; KEEP IDN BOOT;
/* ***** */
/* BOOTAMP is the dataset for select of bootstrap sample */
/* BOOT:bootstrap sample||IDN:Subject ID number */
5 /* ***** */
DO BOOT=1 TO 100; DO SUBCNT=1 TO 30;
CALL RANUNI(SD1,IDN); IDN=CEIL(IDN*30); OUTPUT;
END; END; RUN;
PROC FREQ DATA=BOOTAMP; TABLES IDN; RUN; /* Fr(IDN) for checking */
10 PROC SORT DATA=BOOTAMP; BY IDN; RUN; /* Order by subject number */
DATA BOOTRUN; MERGE BOOTAMP(IN=INB) DATASRC(IN=IND); BY IDN;
IF (INB & ^IND) THEN PUT "Error - In boot, not dataset" BOOT= IDN= _N_=;
RUN;
PROC SORT DATA=BOOTRUN; BY BOOT; RUN; /* Order by boot sample count */
15 PROC REG DATA=BOOTRUN OUTEST=OUTR; /* Do regression for each boot */
MODEL RES=VARYA VARYB; BY BOOT; RUN;
PROC MEANS DATA=OUTR; VAR _RMSE_ VARYA VARYB; RUN;
```

Figure 6: A bootstrap model for regression analysis

A macro for bootstrapping in SAS

A macro for bootstrapping in SAS is presented in Figure 7. This is considerably abstracted from the approach taken

```
%macro _boot(_base=, _dset=, _b=, _idv=, _bset=, _seedf=, _seedn=);
%*****
%* macro _boot - generates bootstrap sample, compute, and find results *;
%* parameters||_base:base||_dset:dataset||_b:number of bootstrap samples *;
5 %* _idv:ID variable||_bset:bootstrap dataset||_seedf:seed file *;
%* _seedn:seed name *;
%* ***** *;
PROC MEANS DATA=_base.._dset. NOPRINT; VAR _idv.; OUTPUT OUT=SUBV N=SUBCNT; RUN;
DATA _bset. (KEEP=BOOT & _idv.) _seedf. (KEEP=_seedn.); MERGE SUBV _seedf.;
10 DO BOOT=1 TO _b.; DO IND=1 TO SUBCNT; CALL RANUNI(_seedn., VAL); _idv.=CEIL(VAL*SUBCNT);
OUTPUT _bset.; END; END; CALL SYMPUT("_ns", COMPRESS(PUT(SUBCNT,5))); OUTPUT _seedf.; RUN;
PROC FREQ DATA=_bset.; TABLES _idv. / NOPRINT OUT=TAOUT; RUN;
DATA TAOUT; SET TAOUT END=EOF; CHISQ+(((COUNT-_b.)*2)/_b.);
IF (EOF) THEN DO; PROBV=1-PROBCHI(CHISQ,_ns.); PUT "Chi-sq:" CHISQ " p <= " PROBV; END; RUN;
15 PROC SORT DATA=_bset.; BY _idv.; RUN; PROC SORT DATA=_base.._dset.; BY _idv.; RUN;
DATA _bset.; MERGE _base.._dset. (IN=INDS) _bset. (IN=INBS); BY _idv.;
IF (INBS & ^INDS) THEN PUT "In bootstrap, but not master" BOOT= _idv.=; RUN;
PROC SORT DATA=_bset.; BY BOOT; RUN;
PROC PRINT TO PRINT=_prv. NEW; RUN;
20 PROC REG DATA=_bset.; VAR DVA DVB=IVA IVB; RESTRICT IVA=IVB; RUN;
PROC REG DATA=_bset.; VAR DVA DVB=IVA IVB; RUN;
PROC PRINT TO; RUN;
```

Figure 7: A bootstrap macro for restricted regression analysis

in Figure 6. This macro allows a far simpler specification of the number of resamples (this is parameter _b) and includes a simple test of the random generation of ID numbers. The various summary quantities and order statistics from the bootstrap sampling distribution are obtained directly from PROC UNIVARIATE, which computes such values as a matter of course.

The macro could be further abstracted by including the actual procedure to be run as a module, and then including the name of the module as a macro; for instance, it could be called _analmod. Then, including the line:

`%&_analmmod.;`

in place of Lines 13-22 could be included.

Decoding output and reading statistics

One very promising direction in the SAS system lies in the output delivery system (ODS), currently implemented in the MIXED procedure, but nowhere else. This approach allows the user to decide which statistics need to be placed in a file. It is not clear when the ODS will be fully implemented in the entire system. Until this is done, persons performing bootstrapping and simulation will probably be forced to fall back upon a relatively standard approach, reading output from SAS procedures using SAS datasteps.

The basic procedure that is followed in my laboratory for this process is as follows:

1. Use a set of contiguous titles, in which no blank lines are included. This allows the procedure to use the title to determine when a procedure has finished a section, as most (if not all) SAS procedures only output titles at the beginning of a section (more on this later)
2. Use a large pagesize, 4000-5000 if possible. This ensures that all printing for a section will be done in one logical page
3. Use a standard linesize, possibly 80. The exact number is not important, but if it is not controlled, many procedures will place information in different columns with different linesize values. Thus, the whole process is simplified by the control of linesize
4. Optionally, use `OPTIONS NODATE NONUMBER`. Occasionally, dates and numbers force an unexpected line to be printed, and many counting processes can be thrown off by this
5. Route output to a file using `PROC PRINTTO`. Then, using a subsequent SAS datastep, read the output and find the information which is needed.
6. Generally, the basic approach lies in finding a particular piece of text, and then reading the value in relationship to the text, either on the same line or on a line in the same general area. Sometimes, relative positioning using `INDEX` is needed, but these are more complex, and probably aren't routinely needed.

Figure 8 presents an example. In this case, the choice of when to output information is made when particular information is found in the file. Line 27 contains control information; since many of the values are found on different lines, they are `RETAINED` until the final value is encountered. This section of the code generates two datasets, which will be merged together to form a dataset with the statistics of interest. These may then be used as in Thompson (1991). Lines 32-42 is the heart of the section, in which the information is read off the file in a position determined on the basis of certain text landmarks. Output to one of the two datasets is performed in Lines 43-44, which check for the restriction condition to determine where output is routed. Use of the macro will be discussed later.

Conclusion

This tutorial has examined tools and techniques for performing simulation and bootstrapping in SAS. The two methods are not antithetical, but the relationships are tenuous. Nonetheless, there are some common notions which are involved in both. First, careful control of seeds over trials, and from one place in the analysis to another, is vital to the integrity of the approach, and the long-term viability of the research effort. Second, the statistics which are required for evaluation of either method are often not presented in a routine basis in output datasets. Thus, methods for extraction of such values from output must often be used, because these methods are commonly used when evaluating statistical methods which are non-standard. Thus, controlling random generation of values and reading output are needed in both.

```

%macro _rd(_p);%_p._T=INPUT(SUBSTR(LINE,50,11),11.);
%_p._P=INPUT(SUBSTR(LINE,68, 6), 6.);%mend _rd;
25 %macro _nms(_e);F&_e. PROB&_e. RSQ&_e. INT_T&_e. INT_P&_e.
IVA_T&_e. IVA_P&_e. IVB_T&_e. IVB_P&_e.%mend _nms
DATA RR(KEEP=_NAME_ BOOT RST_TR RST_PR %_nms(R))
RUR(KEEP=_NAME_ BOOT %_nms(U));
INFILE "redo.out" LRECL=80 PAD;INPUT LINE $CHAR80.;
30 RETAIN BOOT _NAME_; DROP LINE BOOTS;
ARRAY VL %_nms(R); ARRAY UL %_nms(U);
IF (INDEX(LINE,"BOOT")) THEN DO;BOOTS=SUBSTR(LINE,INDEX(LINE,"BOOT")+5);
BOOTS=SUBSTR(BOOTS,1,INDEX(BOOTS," "));BOOT=INPUT(BOOTS,3.);END;
IF (INDEX(LINE,"Dependen")) THEN DO;_NAME_=SUBSTR(LINE, INDEX(LINE,"")+1);END;
35 IF (INDEX(LINE,"Squares")) THEN DO;
INPUT / LINE $CHAR80.;F=INPUT(SUBSTR(LINE,58,10),10.);
PROB=INPUT(SUBSTR(LINE,73,6),6.);
INPUT /// LINE $CHAR80.;RSQ=INPUT(SUBSTR(LINE,54,8),8.);
INPUT ////////// LINE $CHAR80.;%_rd(int);
40 INPUT LINE $CHAR80.;%_rd(iva);
INPUT LINE $CHAR80.;%_rd(ivb);
INPUT LINE $CHAR80.;
IF (SUBSTR(LINE, 7,4) = "REST") THEN DO;%_rd(rst);OUTPUT RR;END;
ELSE DO; DO I=1 TO 9; UL{I}=VL{I}; END; OUTPUT RUR; END;
45 END;
RUN;
DATA BOTH;MERGE RR RUR;BY BOOT _NAME_;
ARRAY VL %_nms(R); ARRAY UL %_nms(U); ARRAY DL %_nms(D);
DO I=1 TO 9;DL{I}=VL{I}-UL{I}; END;RUN;
50 PROC UNIVARIATE;VAR RST_T %_nms{D};RUN;
%mend _boot;

```

Figure 8: Reading PROC REG output with a datastep

References

- Diaconis, P. M. and B. S. Efron (1983). (may) computer-intensive methods in statistics. *Scientific American* 23, 116-130.
- Efron, B. S. (1982). *The jackknife, the bootstrap and other resampling plans*. Philadelphia, PA: Society for Industrial and Applied Mathematics.
- Efron, B. S. and R. Tibshirani (1986). Bootstrap methods for standard errors, confidence intervals and other measures of statistical accuracy. *Statistical Science* 1, 54-74.
- Johnson, M. E. and J. S. Ramberg (1977). *Elliptically symmetric distributions: Characterizations and random variate generation*, pp. 262-265. Washington DC: American Statistical Association.
- Kennedy Jr., W. J. and J. E. Gentle (1980). *Statistical Computing*. New York, NY: Marcel Dekker, Inc.
- T, L. S. and J. L. Hammond (1975). Generation of pseudo-random numbers with specified univariate distributions and correlation coefficients. *IEEE Transactions on Systems, Man and Cybernetics* 5, 557-560.
- Thompson, P. A. (1991). Re-sampling approaches to hypothesis testing in structural models. *Multivariate Behavioral Research* 26, 737-763.

Contacting the author. For more information about simulation and bootstrapping, please contact:

Paul A. Thompson, Ph.D.
Department of Psychiatry, Laboratory of Biological Psychiatry
Case Western Reserve University
Cleveland, OH 44106
Phone: (216) 844-8946, Fax: (216) 844-5403

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brands and product names are registered trademarks or trademarks of their respective companies.