Efficient Repeated Generation of Random Samples

Hank Ledden and Laura Bauer
Board of Governors of the Federal Reserve System

Abstract

Often in the evaluation of sampling techniques and estimation procedures, it is necessary to generate multiple random samples. A SAS® user wishing to generate such samples can do so by using either SAS Macro language or SAS Interactive Matrix Language (SAS/IML®). The question now arises as to which software is more appropriate to use. Using SAS macro involves multiple repetitions of the data step and multiple executions of the output statement, while SAS/IML involves extensive subscripting and manipulation of matrices. To determine which software is most appropriate, we will compare the two on the basis of efficiency, as measured by execution time, amount of region required, amount of I/O involved and total computing cost. The end result of this study will be the construction of an efficient generalized random sample generation procedure.

I. Introduction

This study compares two methods that use different SAS® software to repeatedly generate stratified random samples (SRS). One method utilizes basic SAS data step processing within a macro, while the other utilizes the matrix data structure and associated operations of SAS/IML®. We will evaluate each method and identify the advantages gained when using one method over the other. Examination of CPU processing time, region size and disk I/O used for each method will reveal where the tradeoffs between the methods appear.

The design of this study was conceived in the fall of 1987, shortly after we conducted a large project that required the generation of 100 SRS for eight sample sizes ranging from 270 to 1000. The samples were drawn fairly easily using SAS macro, but we were curious if perhaps there was a better approach. The macro approach relied on repeated executions of the data step. Consequently, as the sample size increased, the iterations of the data step increased correspondingly, causing disk I/O to increase substantially, resulting in significant increases in CPU processing time and total estimated cost.

We then focused on SAS/IML as an alternative. The fundamental data element in this software is a two dimensional array or matrix. Instead of sequentially processing single data elements, SAS/IML software operates on entire matrices. Matrix operations presumably would require less disk I/O and thus cost less, unless there was an offsetting increase in CPU processing time.

II. Design of the Efficiency Study

A. Goal of the Study and Parameters of Interest

The intent of this study is to compare two alternative methods for generating multiple SRS. We are therefore interested in evaluating their relative performance over several levels of sample size (n), numbers of samples (m), and numbers of strata (H). Intuitively, the number of samples we generate will have the greatest effect on the overall cost of the job, although we expect that the cost will also increase as the sample size is increased. We do not, however, anticipate any significant effect on job performance by either increasing or decreasing the number of strata, therefore we will consider only two levels (H=5, 10).

In order to cover a sufficiently broad range of values of the control parameters, we will investigate five different sample sizes which correspond to sampling proportions of 1%, 2.5%, 5%, 10%, and 20% of the universe (n/N). The numbers of samples to be generated were chosen to be m = 10, 25, 50, 100, 250, and 500 in the hope of being able to investigate the functional form of the cost curve for each approach. We would like to know, for example, if the rate of cost increase diminishes as the number of samples is increased. The range that we have selected should allow us to address this question.

B. Efficiency Comparison Criteria

The ultimate comparison criterion for this study is the total cost of the job, which at our installation is the sum of CPU, I/O, and printing costs. The CPU cost is a function of the actual processing time and the amount of region used. All of these cost components, with the exception of printing costs, will be recorded and utilized in making an overall assessment of the relative efficiency of the two methods.

C. The Universe

The universe for this study consists of 5940 commercial banks that reported total deposits of $1 billion or less as of November 21, 1988. Stratification and allocation of the universe was performed using total deposits as a size variable and total transaction accounts as the variable of interest to be estimated, with total deposits as the covariate. Strata limits for H = 5 and H = 10 were formulated using the sum-root-rule (Cochran, pp. 127-129). After the strata limits were assigned, Neyman Allocation was used to determine the number of banks (nH) to sample from each stratum based on the population with each stratum (Nh) (Cochran pp. 98-99). The allocation was performed for sample sizes n = 60, 150, 300, 600, and 1200, which correspond roughly to the sampling proportions discussed in Section II A.

The universe data, consisting of BANK (bank number) and STRATUM, and the sample parameters, consisting of STRATUM, NPOP (Nh), NH (nH), and SAMPSIZE (n) were each stored in a permanent SAS data set. Thus, the inputs to the SRS generators were constructed for the cases of interest.

III. Methodology

A. The Basic Algorithm

Both methods employ essentially the same algorithm for randomly selecting n bank numbers from a set of N within each stratum. A particular bank number is selected for the sample only if a randomly generated Uniform(0,1) deviate falls within the interval (0,NUM/DENUM), which is initially defined by NUM=nH and
DENOM = N_p. This interval is continuously adjusted to reflect the changing probability with which observations are added to the sample. The DENOM counter is decremented by one as each observation is processed, whereas the NUM counter is decremented only when the current observation is added to the sample (SAS® Applications Guide, p.228). Appendix I provides a flowchart of this basic simple random sample generation algorithm.

Although they employ the same algorithm, each method implements it differently. These differences are discussed in the next two subsections.

**B. The Macro/DATA Step Approach**

This program (see Appendix II A) uses the BY statement within a series of data steps contained in a macro to sequentially build a sample data set, observation by observation, one stratum at a time. The first two steps in the program merge the universe data with the sample parameters, BY STRATUM, so that each observation in the universe has the proper NH (n_p), NPOP (N_p), and SAMPsize (n_p) associated with it. The next step creates an empty data set of n observations, SAMP=SMP_SZ, where the value of macro variable &SMP_SZ is equal to the size of the sample (n_p) to be drawn. This data set will be merged later in the program with the first set of random samples that are drawn.

Generation of the random samples occurs in the next data step, DATA SAMPLES, which employs the basic algorithm described above. Specifically, a bank is selected for the sample if the value of variable RAND, a randomly generated deviate, is less than the value of variable PROB, which is obtained by dividing NUM by DENOM. As each observation is processed, the value of PROB changes, reflecting the adjustments made to the values of NUM and DENOM, which are initially set to n_p and N_p respectively. Once the correct number of sample banks for each STRATUM has been drawn, they are then merged in the next DATA step with the empty data set SAMP=SMP_SZ, thus completing the generation of one SRS of size n.

By repeatedly executing these last two DATA steps, a final data set is created which contains the desired number of samples for a given sample size. This iterative process is performed by a macro DO statement, coded in the program as %DO K=1 TO &SMP_NM, where the value of macro variable &SMP_NM corresponds to the number of samples (m) to be drawn. This %DO statement that enables us to build the final data set of n observations with m variables, SAMP1-SAMPm, containing the randomly selected bank numbers of the m samples.

**C. PROC IML/Matrix Approach**

The application of SAS/IML to the construction of the SRS generator also makes use of the macro facility as the shell of a "subroutine" that is coded primarily in IML. By using the macro code, the parameters of interest can be easily changed and passed into the SRS generation code itself. There are two DATA steps within the subroutine (macro) which set up the data to be read into matrix form by PROC IML.

The IML code (see Appendix II B) involves the formulation of a universe vector of the bank numbers of dimension n x 1 and a sample parameter matrix of dimension N x 2 containing the N_p and n_p used for the initial definition of NUM and DENOM for each stratum. Since the strata identification numbers are not contained in the universe vector, the IML SRS generator relies on the assumption that the SAMPNUM numbers are indeed correct and that the UNIV vector is sorted by stratum prior to input to IML. (This assumption is the very reason that the DATA steps and sorts are included within the subroutine, outside of IML.)

The actual random sample generation is performed within each stratum by a loop which scrolls through the observations in the given stratum. This loop, in turn, is nested within a loop which goes through the strata one at a time and reinitializes the NUM and DENOM values for the given stratum. This pair of nested loops is interwoven within a third loop that is executed m times in order to generate the desired number of samples.

Unlike the macro/DATA step method, the IML code does not output the sample banks one at a time. Instead, it utilizes a pair of counters which record the position (or location) of each sample bank within the UNIV vector. Once all position numbers have been collected for a given sample, the bank numbers are put into a SMP vector through the one-step referencing of the universe vector. Although the sample is generated within a loop, the universe vector is referenced only one time for each sample, not once for each observation placed in the sample. Then subsequent samples are treated as individual observations and are appended to one open SAS data set. This reduces the amount of disk I/O involved in the task. However, in order to form a data set exactly like the one created by the macro/DATA step approach, PROC TRANSPOSE must be applied to the data set in order to interchange "observations" with "variables."

**IV. Discussion of Results**

Table I contains the results of the comparison study. The response measures used in making our evaluations of the two methods are: CPU time and its cost, disk I/O and its cost, region used, and total estimated cost of the batch job. Performing over all combinations of the two strata levels (H = 5,10), five levels of sample size (n = 50, 150, 500, 1200), and six levels of sample size (m = 10, 25, 50, 100, 250, 500), the relative importance, in terms of costs, of these control parameters is quite clear. As anticipated, the number of samples to be generated has the most significant effect in terms of increasing costs for both methods, whereas the number of strata seems to make no significant difference at all (at least on the range we investigated, which is why we did not feel it necessary to run the generator for m = 500 and H = 5 other than for n = 50 just as an example). Increasing the sample size also results in increased cost, although not to the same degree as increasing the number of samples.

Overall, it appears that the macro method outperforms the IML method to a limited degree when the number of samples to be generated is small (m < 100) (see Figure 1). But, once the number of samples generated exceed 100, the IML approach proves much less costly with respect to the total estimated cost, due in part to the lower I/O cost associated with IML. Examination of the CPU and disk I/O cost curves in Figure 2 reveals the increasing rate at which the macro I/O cost increases as m increases. Discontinuities of costs seem to be in effect for the macro I/O costs, while the IML I/O costs are essentially linear with a near zero slope (see Figure 3).

Although the IML method provides a sizable savings in cost over the macro method, when the number of samples to be...
generated exceeds 100, the savings in overall cost must be weighed against the increased execution time and larger region size needed to run the IML program. Examination of CPU processing time for the IML method reveals that it far exceeds that of the macro method (see Figure 4). This increased CPU processing time and larger region requirement could prove an inconvenience to the user. For example, at our installation, batch jobs exceeding a region request of 2752K of CPU time can be run only during nonprime computing hours, i.e., between 6pm and 8am daily, or on weekends. These considerations, although not measurable in a purely quantifiable sense, must play a part in evaluating the relative efficiency of these two methods.

However, it is nonetheless interesting to see just how influential (in terms of cost) the tradeoff between the increased CPU processing time of IML and the increased disk I/O of the macro approach can become when the number of samples to be generated is increased. The large differential in terms of overall cost of the batch jobs when \( m \) was large was somewhat unexpected.

V. Conclusion

In the course of our study to determine which of the two methods was more appropriate for generating multiple SRSs, we discovered that the choice of method is somewhat conditioned. If the number of samples to be generated is less than or equal to 100, then use of the macro approach would be more appropriate. However, once the number of samples to be generated exceeds 100, the preferred method would be dependent upon what is most important to the user. If lower overall cost is the user's most immediate concern, the IML method should be used. If the user measures efficiency by the real time it takes to accomplish the task, then the user should employ the macro approach. Thus, the choice of method depends upon the number of samples to be generated and on the user's interpretation of efficiency.

Table I: Results of the Efficiency Study

<table>
<thead>
<tr>
<th>Resources</th>
<th>CAS-TOOL</th>
<th>MACRO/MACRO</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU TIME</td>
<td>9.63</td>
<td>22.14</td>
</tr>
<tr>
<td>DISK I/O</td>
<td>27.94</td>
<td>25.48</td>
</tr>
<tr>
<td>REGION 1</td>
<td>0.82</td>
<td>0.78</td>
</tr>
<tr>
<td>REGION 2</td>
<td>1.28</td>
<td>1.28</td>
</tr>
<tr>
<td>REGION 3</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>REGION 4</td>
<td>2.00</td>
<td>2.00</td>
</tr>
<tr>
<td>REGION 5</td>
<td>3.00</td>
<td>3.00</td>
</tr>
<tr>
<td>REGION 6</td>
<td>4.00</td>
<td>4.00</td>
</tr>
<tr>
<td>REGION 7</td>
<td>5.00</td>
<td>5.00</td>
</tr>
<tr>
<td>REGION 8</td>
<td>6.00</td>
<td>6.00</td>
</tr>
<tr>
<td>REGION 9</td>
<td>7.00</td>
<td>7.00</td>
</tr>
<tr>
<td>REGION 10</td>
<td>8.00</td>
<td>8.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Costs</th>
<th>CAS-TOOL</th>
<th>MACRO/MACRO</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU COST</td>
<td>1.98</td>
<td>4.00</td>
</tr>
<tr>
<td>DISK COST</td>
<td>2.00</td>
<td>2.00</td>
</tr>
<tr>
<td>REGION 1 Cost</td>
<td>0.37</td>
<td>0.37</td>
</tr>
<tr>
<td>REGION 2 Cost</td>
<td>0.37</td>
<td>0.37</td>
</tr>
<tr>
<td>REGION 3 Cost</td>
<td>0.37</td>
<td>0.37</td>
</tr>
<tr>
<td>REGION 4 Cost</td>
<td>0.37</td>
<td>0.37</td>
</tr>
<tr>
<td>REGION 5 Cost</td>
<td>0.37</td>
<td>0.37</td>
</tr>
<tr>
<td>REGION 6 Cost</td>
<td>0.37</td>
<td>0.37</td>
</tr>
<tr>
<td>REGION 7 Cost</td>
<td>0.37</td>
<td>0.37</td>
</tr>
<tr>
<td>REGION 8 Cost</td>
<td>0.37</td>
<td>0.37</td>
</tr>
<tr>
<td>REGION 9 Cost</td>
<td>0.37</td>
<td>0.37</td>
</tr>
<tr>
<td>REGION 10 Cost</td>
<td>0.37</td>
<td>0.37</td>
</tr>
</tbody>
</table>
Figures: Efficiency Comparisons for $H=10$, $n=600$, as a Function of Number of Samples Generated

1. Total Job Cost

2. CPU and I/O Cost

3. Disk I/O

4. CPU Processing Time
Acknowledgements

SAS® and SAS/Graph™ are registered trademarks of SAS Institute Inc., Cary, North Carolina, U.S.A.

SAS/IML® is a trademark of SAS Institute Inc., Cary, North Carolina, U.S.A.

The authors would like to thank Sam Skolnik who provided us with the basic code for the cum-root-f and Neyman Allocation algorithms.

This paper does not necessarily reflect the views of the Board of Governors of the Federal Reserve System; therefore, no official endorsement should be inferred.

References


Author Contact

Hank Leddon
Federal Reserve Board
20th and C Streets, N.W.
Washington, D.C. 20551
(202) 728-5966

Appendix I: Algorithm for Generating Simple Random Samples

```
NUM = n
DENOM = N

READ NEXT OBS

RAND = U(0,1)

RAND < NUM/DENOM ?

yes

OUTPUT OBS TO SAMPLE

NUM - 1

no

DENOM - 1

no

DENOM = 0 ?

yes

STOP
```
A: Code for the Macro/DATA Step Approach

%MACRO SAMPGEN(SMP_SZ, SMP_NM);
/* SET SAMP DATA */
DATA SAMPNUMS;
SET KEEPERI.SMPNUM;
IF SAMPampire EQ &SMP_SZ;
/* MERGE UNIV & SAMP DATA BY STRATUM */
DATA SAMPLES;
MERGE EDDSUNIV SAMPNUMS;
BY STRATUM;
/* CREATE DUMMY DATA SET */
DATA SAMPLER;
MERGE EDDSUNIV SAMPNUMS;
BY STRATUM;
/* CREATE DUMMY DATA SET */
DATA SAMP(SAMP_SIZE DROP=K);
DO K=1 TO &SMP_SZ;
OUTPUT;
END;
/* GENERATE SAMPLES */
DO K=1 TO &SMP_NM;
DATA SAMPLES(KEEP=SAMPK);
SET SAMPLES;
BY STRATUM;
IF FIRST.STRATUM THEN DO;
NUM=NH;
END;
DENOM=NPOP;
RETAIN NUM DENOM;
END;
RAND=UNIFORM(0);
PROB=NUM/DENOM;
IF RAND < PROB THEN DO;
SAMP=BANK;
OUTPUT;
END;
NUM=NUM-1;
END;
DENOM=DENOM-1;
/* MERGE EACH SUCCESSIVE DRAW OF SRS */
DATA SAMP(SAMP_SIZE SAMPLES);
MERGE SAMP(SAMP_SIZE SAMPLES);
%MEND SAMPGEN;
/* SET UNIV DATA */
DATA EDDSUNIV;
SET KEEPERI.EUNIV;
/* CALL SAMPGEN FOR DESIRED SAMP_SZ */
%SAMPGEN(600,500)

B: Code for the SAS/IML Approach

%MACRO SRSGENER(N,M,H,SAMP_SIZE);
DATA UNIVERSE KEEP=BANK;
SET KEEPERI.EUNIV;
/* ASSUMES DATA SETS ARE ALREADY */
/* SORTED BY STRATUM */
DATA SAMPNUM KEEP=NPOP NH;
SET KEEPERI.SAMPNUM;
IF SAMP_SIZEayne;
PROC IML;
START READIN;
USE SAMPNUM;
READ ALL INTO NH;
USE UNIVERSE;
READ ALL INTO UNIV;
FINISH;
/* END OF READIN */
RUN READIN;
START FILLITIN;
BANKS=J(1,N,1.0);
DO K=1 TO NM;
SC=1; BC=1;
DO L=1 TO NH;
NUM=NPOP(1,1.0);
WH=NPOP(1,1.1);
DEN=NHH;
DO I=1 TO NH;
RAND=UNIFORM(0);
PROB=NUM/DEN;
START CHECK;
IF RAND < PROB THEN DO;
BANKS(1,SC,1)=BC;
SC=SC+1;
END;
END;
BANKS=INT(BANKS);
SAMP=UNIV(1,BANKS(1,1),1);
START PUTIOUT;
IF (K=1) THEN DO;
CREATE SAMS FROM SAMP;
END;
APPEND FROM SAMP;
FINISH;
/* END OF PUTIOUT */
RUN PUTIOUT;
END;
FREE BANKS UNIV ALLNOM;
FINISH;
/* END OF FILLITIN */
RUN FILLITIN;
CLOSE SAMS;
QUIT;
PROC TRANSPOSE DATA=SAMPS OUT=SAMPLES PREFIX=SAMP;
%MEND SRSGENER;
/* CALL MACRO SRSGENER */
%SRSGENER(150,500,10,SAMPLES);