Paper SAS1387-2015

Ten Tips for Simulating Data with SAS®

Rick Wicklin, SAS Institute Inc.

ABSTRACT

Data simulation is a fundamental tool for statistical programmers. SAS® software provides many techniques for simulating data from a variety of statistical models.

However, not all techniques are equally efficient. An efficient simulation can run in seconds, whereas an inefficient simulation might require days to run. This paper presents 10 techniques that enable you to write efficient simulations in SAS. Examples include how to simulate data from a complex distribution and how to use simulated data to approximate the sampling distribution of a statistic.

INTRODUCTION

Simulation is a brute-force computational technique that relies on repeating a computation on many different random samples in order to estimate a statistical quantity. However, "brute-force" does not have to mean "slow"! The tips in this paper can help you write simulations that run hundreds of times faster than a naive simulation. The first five tips describe how to simulate complex data that have specified statistical properties. The last five describe how to write efficient programs in SAS that apply simulation-based techniques to practical problems such as estimating power.

This paper is based on tips and techniques that appear in Wicklin (2013a). Some of the examples have also appeared on *The DO Loop* blog (Wicklin 2010a). Each tip is accompanied by a complete SAS program. Many of the techniques use only the DATA step and Base SAS® or SAS/STAT® procedures. However, some of the multivariate techniques use the SAS/IML® matrix language. For an introduction to the SAS/IML programming language, see Wicklin (2010b).

The IML procedure is included as part of SAS® University Edition, which is free for students, professors, researchers, and adult learners. All the programs in this paper can be run in SAS University Edition.

TIP 1: HOW TO SIMULATE DATA FROM A CONTINUOUS DISTRIBUTION

You can use the RAND function in the SAS DATA step to simulate from an elementary probability distribution such as a normal, uniform, or exponential distribution. The first parameter of the RAND function is a string that specifies the name of the distribution. Subsequent parameters specify the values of the shape, location, or scale parameters for the distribution.

Longtime SAS programmers might recall older random number functions such as the RANUNI, RANNOR, and RANBIN functions. These functions use a linear congruential algorithm that was popular in the 1970s. There are several reasons why you should not use these older functions for statistical simulation (Wicklin 2013b). The primary reason is that pseudorandom numbers that come from a linear congruential algorithm are not as "random" (statistically speaking) as pseudorandom numbers that come from the Mersenne-Twister algorithm (Matsumoto and Nishimura 1998), which is used by the RAND function.

The STREAMINIT subroutine is used to set the seed for the random number stream. The following DATA step simulates 100 independent values from the standard normal and uniform distributions. The subsequent PROC PRINT step displays the first five observations in Figure 1.

```
data Rand(keep=x u);
call streaminit(4321);                /* set seed */
do i = 1 to 100;                      /* generate 100 random values */
   x = rand("Normal");                /* x ~ N(0,1) */
   u = rand("Uniform");               /* u ~ U(0,1) */
   output;
end;
run;
```

```
proc print data=Rand(obs=5);
run;
```

Figure 1 Five Random Observations from Normal and Uniform Distributions

Obs	x	u
1	1.24067	0.92960
2	-0.53532	0.20874
3	-1.01394	0.45677
4	0.68965	0.27118
5	-0.67680	0.87254

In SAS/IML software you can use RANDGEN function to fill each element of an allocated matrix with random draws from a probability distribution. You can also use the RANDFUN function, which returns a matrix of a specified size. For both functions, you can use the RANDSEED subroutine to set the random number seed as follows:

```
proc iml;
call randseed(1234);                /* set seed */
x = j(50, 2);                       /* allocate 50 x 2 matrix */
call randgen(x, "Normal");          /* fill matrix, x ~ N(0,1) */
u = randfun(100, "Uniform");        /* return 100 x 1 vector, u ~ U(0,1) */
```

TIP 2: HOW TO SIMULATE DATA FROM A DISCRETE DISTRIBUTION

The previous section shows that the RAND function supports common continuous probability distributions. The RAND function also supports common discrete probability distributions such as the Bernoulli, binomial, and Poisson distributions.

In addition to these familiar parametric distributions, the RAND function supports the “table” distribution, which enables you to specify the probabilities of selecting each element in a set of k categories. This distribution is useful when you want to simulate categorical data according to the empirical frequencies in an observed set of data.

For example, suppose a call center classifies calls into three categories: “Easy” calls account for 50% of the calls, “Specialized” calls account for 30%, and “Hard” calls account for the remaining 20%. If you want to simulate the categories for 100 random calls, you can use the “table” distribution and specify that the first category (Easy) occurs with probability 0.5, the second category (Specialized) occurs with probability 0.3, and the third probability occurs with probability 0.2. Then the “table” distribution returns the values 1, 2, or 3, as shown in Figure 2, which is created by the following statements:

```
data Categories(keep=Type);
call streaminit(4321);
array p[3] (0.5 0.3 0.2);       /* probabilities */
do i = 1 to 100;
   Type = rand("Table", of p[*]); /* use OF operator */
   output;
end;
run;

proc format;
   value Call  1='Easy' 2='Specialized' 3='Hard';
run;

proc freq data=Categories;
   format Type Call.;
   tables Type / nocum;
run;
```

Notice that the OF operator is used because the probabilities are contained in a DATA step array. You could also list the three probabilities in the RAND function by using a comma-separated list.

Figure 2 shows the distribution of the three categories in a random sample of 100 draws. The value 1, which is formatted as “Easy,” appears 48 times in this random sample. The value 2 (“Specialized”) appears 31 times. The value 3 (“Hard”) appears 21 times. This example illustrates sampling variability: the empirical distribution for the sample is close to, but not identical to, the distribution for the population.

Figure 2 Frequencies in Random Sample of 100 Categories

The FREQ Procedure

Type	Frequency	Percent
Easy	48	48.00
Specialized	31	31.00
Hard	21	21.00

The RANDGEN subroutine in the SAS/IML language also supports the “table” distribution. You can put the probabilities into a vector and pass it to the RANDGEN subroutine as follows:

```
proc iml;
call randseed(4321);
p = {0.5 0.3 0.2};
Type = j(100, 1);                   /* allocate vector */
call randgen(Type, "Table", p);     /* fill with 1,2,3 */
```

TIP 3: HOW TO SIMULATE DATA FROM A MIXTURE OF DISTRIBUTIONS

You can combine the “table” distribution with other distributions to generate a finite mixture distribution. A finite mixture distribution is composed of k components. If f_i is the probability density function (PDF) of the ith component, then the PDF of the mixture is $g(x) = \Sigma_{i=1}^{k} \pi_i f_i(x)$, where $\Sigma_{i=1}^{k} \pi_i = 1$ and the π_i are called the mixing probabilities. The “table” distribution enables you to randomly select a subpopulation according to the mixing probabilities.

For example, the section “TIP 2: HOW TO SIMULATE DATA FROM A DISCRETE DISTRIBUTION” shows how to simulate the categories for 100 random calls to a call center. If you assume a distribution of times for each category of calls, you can simulate the time required to answer a call. For example, assume that the time needed to answer a call for each category is normally distributed according to Table 1.

Table 1 Parameters for Normally Distributed Times

Question	Mean	Standard Deviation
Easy	3	1
Specialized	8	2
Hard	10	3

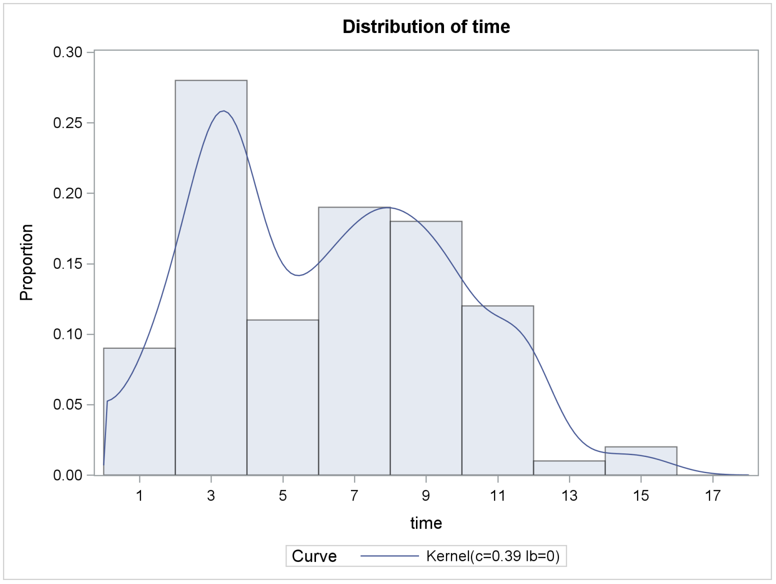
If the calls come in at random, the distribution of times is a finite mixture distribution that combines the three normal distributions. The following DATA step simulates the time required to answer a random sample of 100 phone calls:

```
data Calls(drop=i);
call streaminit(12345);
array prob [3] _temporary_ (0.5 0.3 0.2);      /* mixing probabilities */
do i = 1 to 100;
   Type = rand("Table", of prob[*]);           /* returns 1, 2, or 3 */
   if      Type=1 then time = rand("Normal",  3, 1);
   else if Type=2 then time = rand("Normal",  8, 2);
   else                time = rand("Normal", 10, 3);
   output;
end;
run;
```

The following call to PROC UNIVARIATE displays the distribution of times in Figure 3. The distribution is a mixture of three normal components. The component modes near $T = 3$, $T = 8$, and $T = 10$ are evident.

```
proc univariate data=Calls;
   ods select Histogram;
   histogram time / vscale=proportion kernel(lower=0 c=SJPI);
run;
```

Figure 3 Sample from Mixture Distribution, $N = 100$



In a similar way, you can simulate from a contaminated normal distribution (Tukey 1960), which is often a convenient way to generate normal data that have outliers (Wicklin 2013a, p. 121). The contaminated normal distribution is a two-component mixture distribution in which both components are normally distributed and have a common mean. A popular contaminated normal model simulates values from an $N(0,1)$ distribution with probability 0.9 and from an $N(0,10)$ distribution with probability 0.1.

TIP 4: HOW TO SIMULATE DATA FROM A COMPLEX DISTRIBUTION

If you look in the SAS documentation for the RAND function, you might mistakenly conclude that the function supports only about 20 distributions. Not true! You can combine these simple built-in distributions to generate countless other distributions. For example, the following techniques enable you to create new distributions:

- Translating and scaling: The RAND function does not support location and scale parameters for every distribution, but it is easy to adjust the location and scale. If X is any random variable from a location-scale family of distributions, then $Y = \theta + \sigma X$ is a random variable (from the same distribution) that has a new location and scale parameter. For example, the RAND function does not support a scale parameter for the exponential distribution, but if E is an exponential random variable that has unit scale, then σE is an exponential random variable that has scale parameter σ.

- Transforming: The previous technique applies an affine transformation, but you can apply other transformations to convert one distribution into another. The canonical example is the lognormal distribution: If X is normally distributed with parameters μ and σ, then $Y = \exp(X)$ is lognormally distributed. The power function distribution is another example. If E is a standard exponential random variable, then $Z = (1 - \exp(-E))^{1/\alpha}$ follows a standard power function distribution with parameter α (Devroye 1986, p. 262).

- Acceptance-rejection techniques: If you simulate normal variates and throw away the negative values, the remaining data follow a truncated normal distribution. A similar algorithm will simulate data from a truncated Poisson distribution. These truncated distributions are examples of the general acceptance-rejection technique (Wicklin 2013a, p. 126)

- The inverse CDF transformation: If you know the cumulative distribution function (CDF) of a probability distribution, then you can generate a random sample from that distribution. A continuous CDF, F, is a one-to-one mapping of the domain of the CDF into the interval $(0, 1)$. Therefore, if U is a random uniform variable on $(0, 1)$, then $X = F^{-1}(U)$ has the distribution F. Wicklin (2013a, p. 116) contains examples.

TIP 5: HOW TO SIMULATE DATA FROM A MULTIVARIATE DISTRIBUTION

The RAND function in the DATA step is a powerful tool for simulating data from univariate distributions. However, the SAS/IML language, an interactive matrix language, is the tool of choice for simulating correlated data from multivariate distributions. SAS/IML software contains many built-in functions for simulating data from standard univariate and multivariate distributions. It also supports the matrix computations required to implement algorithms that sample from less common distributions.

A useful multivariate distribution is the multivariate normal (MVN) distribution. The parameters for the MVN distribution are a mean vector and a covariance matrix. You can use the RANDNORMAL function in SAS/IML software to simulate observations from an MVN distribution. The following program samples 1,000 observations from a trivariate normal distribution. The RANDNORMAL function returns a 1000×3 matrix, where each row is an observation for the three correlated variables. You can use the MEAN and COV functions to display the sample means and covariances. Figure 4 shows that the sample statistics are close to the population parameters.

```
proc iml;
Mean = {1, 2, 3};                /* population means */
Cov = {3 2 1,                    /* population covariances */
       2 4 0,
       1 0 5};
N = 1000;                        /* sample size */
call randseed(123);
X = RandNormal(N, Mean, Cov);    /* x is a 1000 x 3 matrix */
SampleMean = mean(X);
SampleCov = cov(X);
varNames = "x1":"x3";
print SampleMean[colname=varNames],
      SampleCov[colname=varNames rowname=VarNames];

/* write sample to SAS data set for plotting */
create MVN from X[colname=varNames];  append from X;  close MVN;
quit;
```

Figure 4 Sample Mean and Covariance Matrix for Simulated MVN Data

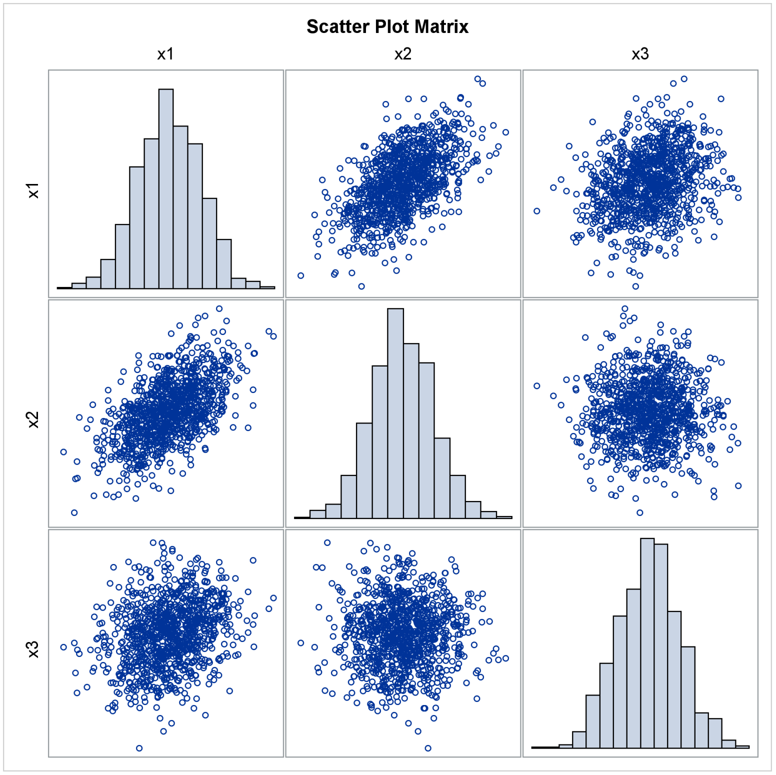
```
             SampleMean
        x1          x2          x3
 0.9823293   1.9762625   3.1103913

               SampleCov
           x1          x2          x3
x1  3.0775945   1.9871478    1.102642
x2  1.9871478   4.0518345   0.0027428
x3   1.102642   0.0027428   5.3153554
```

You can use the CORR procedure to display the scatter plot matrix for the MVN sample, as follows. Figure 5 shows that the marginal distribution for each variable (displayed as histograms on the diagonal) appears to be normal, as do the pairwise bivariate distributions (displayed as scatter plots). This is a characteristic of MVN data: all marginal distributions are normally distributed.

```
/* create scatter plot matrix of simulated data */
proc corr data=MVN plots(maxpoints=NONE)=matrix(histogram);
   var x:;
run;
```

Figure 5 Univariate and Bivariate Marginal Distributions for Simulated MVN Data



If you do not have a license for SAS/IML software, you can use the SIMNORMAL procedure in SAS/STAT software to simulate MVN data.

The SAS/IML language provides functions for simulating from other distributions, including the multivariate t distribution, time series models, and the Wishart distribution, which is a distribution of covariance matrices. The language provides built-in support for the discrete multinomial distribution and provides tools for simulating from correlated binary and ordinal distributions (Wicklin 2013a, Chapter 9). You can also use SAS/IML software to simulate spatial point patterns and Gaussian random fields (Wicklin 2013a, Chapter 14).

TIP 6: HOW TO EFFICIENTLY APPROXIMATE A SAMPLING DISTRIBUTION

The next tip is the most important in this paper: Use a BY statement to analyze many simulated samples in a single procedure call.

Because of random variation, if you simulate multiple samples from the same model, the statistics for the samples are likely to be different. The distribution of the sample statistics is an *approximate sampling distribution* (ASD) for the statistic. The spread of the ASD (for example, the standard deviation) quantifies the precision of the estimate.

For some statistics, such as the sample mean, the theoretical sampling distribution is known or can be approximated for large samples. However, the sampling distribution for many statistics is revealed only through simulation studies.

The process of generating many samples and computing many statistics is known as *Monte Carlo simulation*. The canonical example of a Monte Carlo simulation is computing the ASD of the mean. Suppose you are interested in the sampling distribution of the mean for samples of size 10 that are drawn from a $U(0,1)$ distribution. To generate the ASD efficiently in SAS:

1. Generate a data set that contains many samples of size 10. Create a BY variable that identifies each sample.

2. Compute the means of each sample by using the BY statement in the MEANS procedure.

3. Visualize and compute descriptive statistics for the distribution of the sample means.

The following DATA step implements Step 1:

```
/* Step 1: Generate a data set that contains many samples */
%let N = 10;                      /* sample size */
%let NumSamples = 1000;           /* number of samples */
data Sim;
call streaminit(123);
do SampleID = 1 to &NumSamples;   /* ID variable for each sample */
   do i = 1 to &N;
      x = rand("Uniform");
      output;
   end;
end;
run;
```

The Sim data set contains 10,000 observations. The first 10 observations have the value **SampleID** = 1. The next 10 observations have the value **SampleID** = 2. The last 10 observations have the value **SampleID** = 1000. Because of the structure of the data set, you can analyze all 1,000 samples by making a *single call* to a SAS procedure!

The following statements illustrate the most important technique in this paper: using a BY statement to analyze many samples at one time. In this case, you can call PROC MEANS to obtain the sample mean for each sample. The 1,000 sample means are saved to an output data set called OutStats.

```
/* Step 2: Compute the mean of each sample */
proc means data=Sim noprint;
   by SampleID;
   var x;
   output out=OutStats mean=SampleMean;
run;
```

The Monte Carlo simulation is complete. You can call PROC UNIVARIATE to visualize the approximate sampling distribution of the mean and to compute basic descriptive statistics for the ASD:

```
/* Step 3: Visualize and compute descriptive statistics for the ASD */
ods select Moments Histogram;
proc univariate data=OutStats;
   label SampleMean = "Sample Mean of U(0,1) Data";
   var SampleMean;
   histogram SampleMean / normal;      /* overlay normal fit */
run;
```

Figure 6 Summary of the Sampling Distribution of the Mean of $U(0,1)$ Data, $N = 10$

The UNIVARIATE Procedure
Variable: SampleMean (Sample Mean of U(0,1) Data)

Moments			
N	1000	**Sum Weights**	1000
Mean	0.50264072	**Sum Observations**	502.640718
Std Deviation	0.09254832	**Variance**	0.00856519
Skewness	-0.019496	**Kurtosis**	0.28029163
Uncorrected SS	261.204319	**Corrected SS**	8.55662721
Coeff Variation	18.4124209	**Std Error Mean**	0.00292663

Figure 7 Approximate Sampling Distribution of the Mean of $U(0, 1)$ Data, $N = 10$

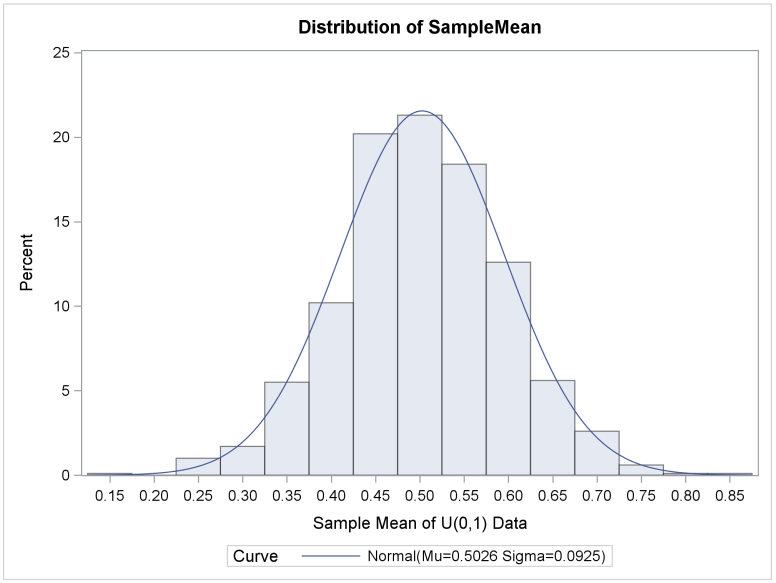


Figure 6 shows descriptive statistics for the **SampleMean** variable. The Monte Carlo estimate of the mean is 0.503; the standard deviation (0.093) estimates the standard error of the mean. Figure 7 shows a histogram of the **SampleMean** variable, which appears to be approximately normally distributed.

An alternate way to perform a Monte Carlo simulation is to use SAS/IML software. The following program computes an ASD of the mean for samples of size 10 that contain $U(0, 1)$ data. Each sample is stored as a row of a matrix. This example shows an efficient way to simulate and analyze many univariate samples in PROC IML. The results of the program are shown in Figure 8. The sample statistics are identical to the results shown in Figure 6.

```
%let N = 10;
%let NumSamples = 1000;
proc iml;
call randseed(123);
x = j(&NumSamples,&N);       /* many samples (rows), each of size N */
call randgen(x, "Uniform");  /* 1. Simulate data                    */
s = x[,:];                   /* 2. Compute statistic for each row   */
Mean = mean(s);              /* 3. Summarize and analyze ASD        */
StdDev = std(s);
call qntl(q, s, {0.05 0.95});
print Mean StdDev (q`)[colname={"5th Pctl" "95th Pctl"}];
```

Figure 8 Analysis of the ASD of the Mean of $U(0, 1)$ Data, $N = 10$

Mean	StdDev	5th Pctl	95th Pctl
0.5026407	0.0925483	0.3540121	0.6588903

Notice the following features of the SAS/IML program:

- There are no loops.
- Three functions are used to generate the samples: RANDSEED, J, and RANDGEN. A single call to the RANDGEN routine fills the entire matrix with random values.
- The colon subscript reduction operator (:) is used to compute the mean of each row of the `x` matrix.

In the program, the column vector `s` contains the ASD. The mean, standard deviation, and quantile of the ASD are computed by using the MEAN, STD, and QNTL functions, respectively. These functions operate on each column of their matrix argument. Although the results are identical to the results from the DATA step and PROC MEANS, the SAS/IML program is more compact. Furthermore, the SAS/IML program can run faster than the equivalent Base SAS computation because the SAS/IML program does not use data sets to exchange information between procedures.

TIP 7: HOW TO SPEED UP A SIMULATION BY SUPPRESSING DISPLAYED OUTPUT

The careful reader will have noticed that the NOPRINT option was used in the PROC MEANS statement in the previous section. This is intentional. By default, most SAS procedures produce a lot of output. However, there is no need to display the output for each BY-group analysis. Instead, you should suppress the tables and use the OUTPUT statement to write the 1,000 sample means to a data set.

About 50 SAS/STAT procedures support the NOPRINT option. For other procedures, you can use ODS to suppress output. You might also want to use the NONOTES option to suppress the writing of notes to the SAS log. Finally, the ODS RESULTS OFF statement prevents ODS from tracking output in the Results window.

The simple act of suppressing output can dramatically increase the speed of a simulation. Furthermore, if you run SAS interactively and do not suppress the output, you might encounter the dreaded “Output WINDOW FULL” dialog box, which states “Window is full and must be cleared.” Not only is this annoying, but it prevents your simulation from running to completion.

The following SAS macros enable you to turn off ODS Graphics, exclude the display of ODS tables, and suppress other unnecessary output:

```
%macro ODSOff();          /* call prior to BY-group processing */
   ods graphics off;
   ods exclude all;       /* all open destinations */
   ods results off;       /* no updates to tree view */
   options nonotes;       /* optional, but sometimes useful */
%mend;

%macro ODSOn();           /* call after BY-group processing */
   ods graphics on;
   ods exclude none;
   ods results on;
   options notes;
%mend;
```

The section “TIP 10: HOW TO SIMULATE DATA TO ASSESS THE POWER OF A STATISTICAL TEST” contains an example that uses the %ODSOff and %ODSOn macros.

In general, the SAS/IML language does not produce any output unless the programmer explicitly specifies a PRINT statement. Consequently, it is not usually necessary to suppress the output of SAS/IML programs.

TIP 8: HOW TO SPEED UP A SIMULATION BY AVOIDING MACRO LOOPS

The most common mistake that SAS programmers make when they write a simulation is that they use a macro loop instead of using the BY-group method that is described in Tip 6. The following program computes the same quantities as the program in Tip 6, but it uses a macro loop, which is less efficient. Avoid writing programs like this:

```
/*******************************************/
/* THIS CODE IS INEFFICIENT. DO NOT USE. */
/*******************************************/
%macro Simulate(N, NumSamples);
options nonotes;                        /* turn off notes to log    */
proc datasets nolist;
   delete OutStats;                     /* delete data if it exists */
run;
```

```
%do i = 1 %to &NumSamples;
   data Temp;                          /* create one sample        */
   call streaminit(0);
   do i = 1 to &N;
      x = rand("Uniform");
      output;
   end;
   run;

   proc means data=Temp noprint;       /* compute one statistic    */
      var x;
      output out=Out mean=SampleMean;
   run;

   proc append base=OutStats data=Out;    /* accumulate statistics */
   run;
%end;
options notes;
%mend;

/* call macro to simulate data and compute ASD. VERY SLOW! */
%Simulate(10, 1000)             /* means of 1000 samples of size 10 */
```

How long does it take to run this macro loop? Whereas the BY-group processing in Tip 6 runs essentially instantaneously, the macro loop runs hundreds of times slower (about 30 seconds). For a more complex simulation, Novikov (2003) reports that the macro-loop implementation was 80–100 times slower than the BY-group technique.

This approach is slow because each small computation requires a lot of overhead cost. The DATA step and the MEANS procedure are called 1,000 times, but they generate or analyze only 10 observations in each call. This is inefficient because every time that SAS encounters a procedure call, it must parse the SAS code, open the data set, load the data into memory, do the computation, close the data set, and exit the procedure. When a procedure computes complicated statistics on a large data set, these overhead costs are small relative to the computation performed by the procedure. However, for this example, the overhead costs are large relative to the computational work.

Be warned: If you do not use the NONOTES option, then the performance of the %SIMULATE macro is even worse. When the number of simulations is large, you might fill the SAS log with inconsequential notes.

Wicklin (2013a, Chapter 6) provides many other tips that enable a simulation to run faster. For SAS/IML programmers, the most important tip is to *vectorize* computations. This means that you should write a relatively small number of statements and function calls, each of which performs a lot of work. For example, avoid loops over rows or elements of matrices. Instead, use matrix and vector computations.

TIP 9: HOW TO SIMULATE DATA TO ASSESS REGRESSION ESTIMATES

Wicklin (2013a, Chapter 6) has four chapters devoted to simulating data from various regression models. This paper presents only a simple linear regression model.

A regression model has three parts: the explanatory variables, a random error term, and a model for the response variable.

The error term is the source of random variation in the model. When the variance of the error term is small, the response depends almost entirely on the explanatory variables, and the parameter estimates have small uncertainty. A large variance simulates noisy data. For fixed-effects models, the errors are usually assumed to be uncorrelated and to have zero mean and constant variance. Other regression models (for example, time series models) make different assumptions.

The regression model itself describes how the response variable is related to the explanatory variables and the error term. The simplest model is a linear regression, where the response is a linear combination of the explanatory variables and the error. More complicated models (such as logistic regression) incorporate a link function that relates the mean response to the explanatory variables.

For example, suppose that you simulate data from the least squares regression model

$$Y_i = 1 + X_i/2 + Z_i/3 + \epsilon_i$$

where $\epsilon_i \sim N(0,1)$ and $i = 1 \ldots N$. You can analyze the simulated data by using various regression methods. Because you know the exact values of the parameters, you can compare the regression estimates from each method.

The values for the variables X and Z can be real data from an experiment or from an observational study, or they can be simulated values. For convenience, the following DATA step simulates two independent normally distributed explanatory variables. In a similar way, you can create synthetic data sets that contain arbitrarily many variables and arbitrarily many observations.

```
%let N = 50;                                /* sample size */
data Explanatory(keep=x z);
call streaminit(12345);
do i = 1 to &N;
   x = rand("Normal");
   z = rand("Normal");
   output;
end;
run;
```

Regardless of whether the explanatory variables were simulated or observed, you can use the DATA step to simulate the response variable for the linear regression model. The following statements model the X and Z variables as fixed effects:

```
/* Simulate multiple samples from a regression model */
%let NumSamples = 1000;                     /* number of samples     */
data RegSim(drop=eta rmse);
call streaminit(123);
rmse = 1;                                   /* scale of error term   */
set Explanatory;                            /* implicit loop over obs*/
ObsNum = _N_;                               /* observation number    */
eta = 1 + X/2 + Z/3;                        /* linear predictor      */
do SampleID = 1 to &NumSamples;
   Y = eta + rand("Normal", 0, rmse);       /* random error term     */
   output;
end;
run;

proc sort data=RegSim;              /* sort for BY-group processing  */
   by SampleID ObsNum;
run;
```

The RegSim data set contains 1,000 samples of size $N = 50$. For each sample, the explanatory variables are identical. However, the response variable (**Y**) is different for each sample because of the random variation from the error term.

Wicklin (2013a, Chapter 11) describes techniques for simulating the data in BY-group order so that you do not need a separate call to sort the data.

You can use the BY-group technique from the section "TIP 6: HOW TO EFFICIENTLY APPROXIMATE A SAMPLING DISTRIBUTION" to estimate the regression coefficients for each sample. The following call to the REG procedure computes parameter estimates for each simulated sample. The parameter estimates are saved to the OutEst output data set. The distribution of those statistics forms an approximate sampling distribution. The subsequent call to the MEANS procedure computes univariate descriptive statistics for each parameter estimate:

```
proc reg data=RegSim outest=OutEst NOPRINT;
   by SampleID;
   model y = x z;
quit;

proc means nolabels data=OutEst Mean Std P5 P95;
   var Intercept x z;
run;
```

Figure 9 Summary Statistics for the Approximate Sampling Distribution of Parameter Estimates

```
                 The MEANS Procedure

Variable        Mean     Std Dev    5th Pctl   95th Pctl
Intercept  1.0024931   0.1497273   0.7580550   1.2551858
x          0.5077344   0.1675608   0.2229376   0.7733572
z          0.3314210   0.1360661   0.1056315   0.5544892
```

Figure 9 summarizes the approximate sampling distribution (ASD) of each parameter estimate. Notice that the sample means for the parameter estimates are extremely close to the value of the regression parameters.

You could use the CORR procedure to explore the multivariate nature of the parameter estimates. For example, the correlation matrix of the variables that contain the parameter estimates is an estimate of the "correlations of the betas." In a similar way, you could visualize the ASD of the root mean square error.

For examples of simulating data from generalized linear models or from mixed models, see Wicklin (2013a, Chapter 12). For logistic regression, see Wicklin (2014).

TIP 10: HOW TO SIMULATE DATA TO ASSESS THE POWER OF A STATISTICAL TEST

In the previous section, the coefficient of the Z variable is $1/3$. However, the error term is comparatively large, so for some random samples the Z term is not statistically significant at the 95% confidence level.

The *power* of a statistical test is the probability that the test can detect an effect when the effect truly exists. You can use the TEST statement in the REG procedure to compute an F statistic that tests the null hypothesis that the Z parameter is 0. As mentioned in the previous paragraph, the F test rejects the null hypothesis (commits a Type II error) for some samples of size 50 when the coefficient of the Z variable is $1/3$. But for how many? What is the probability of rejecting the null hypothesis?

You can use simulation to estimate the power of the F test. When the parameter is truly 0, the F test should reject the null hypothesis 5% of the time, assuming a 95% confidence level. When the parameter is sufficiently large in magnitude, the F test should reject the null hypothesis almost 100% of the time. For intermediate values of the parameter, the probability that the F test rejects the null hypothesis is an increasing function of the parameter. The power curve can be computed by using the MULTREG statement in the POWER procedure. However, you can also use simulation to approximate the power curve.

The following program shows how you can use simulation to estimate the power curve for the F test for various magnitudes of the Z parameter. The program specifies the true model as $Y_i = 1 + X_i/2 + \gamma Z_i + \epsilon_i$, where $\epsilon_i \sim N(0,1)$ and where the parameter γ ranges over the interval $[0, 0.6]$. This problem was discussed in Greene (2000). The values of the explanatory variable are contained in the Explanatory data set, which was created in the previous section.

```
%let N = 50;                                /* sample size          */
%let NumSamples = 1000;                     /* number of samples    */
data PowerSim(drop=eta);
call streaminit(1);
set Explanatory;                            /* fixed values for x and z */
do gamma = 0 to 0.6 by 0.05;
   eta = 1 + x/2 + gamma*z;                 /* linear predictor     */
   do SampleID = 1 to &NumSamples;
      y = eta + rand("Normal");
      output;
   end;
end;
run;

proc sort data=PowerSim;
   by gamma SampleID;
run;
```

This data set contains many samples. For each value of γ there are 1,000 samples, each of size 50. The following call to PROC REG analyzes 13,000 samples. The TEST statement produces the TestANOVA table, which is saved to a SAS data set by using the ODS OUTPUT statement, as follows:

```
/* Turn off output when calling PROC for simulation */
%ODSOff
proc reg data=PowerSim;
   by gamma SampleID;
   model y = x z;
   test z=0;
   ods output TestANOVA=TestAnova(where=(Source="Numerator"));
quit;
%ODSOn

proc print data=TestANOVA(firstobs=13 obs=17);
   var gamma SampleID FValue ProbF;
run;
```

Figure 10 Five p-Values from an F Test

Obs	gamma	SampleID	FValue	ProbF
13	0	13	4.45	0.0403
14	0	14	2.61	0.1127
15	0	15	0.10	0.7513
16	0	16	2.52	0.1192
17	0	17	0.17	0.6805

The results of a few tests are shown in Figure 10. The **ProbF** variable contains the p-values for the F test on each sample. You can see that most p-values are greater than 0.05, but some (for example, the 15th observation) are small, which indicates that the F test rejects the null hypothesis for that sample. The following DATA step creates an indicator variable that has the value 1 if the null hypothesis is rejected at the 0.05 significance level. The FREQ procedure is then used to count the proportion of times that the null hypothesis is rejected for each value of the γ coefficient. Figure 11 is a plot of the proportion of times that the null hypothesis is rejected.

```
/* Construct an indicator variable for observations that reject H0 */
data Results;
   set TestANOVA;
   Reject = (ProbF <= 0.05);                   /* indicator variable */
run;
```

```
/* compute proportion of times H0 is rejected for each gamma value;
   compute upper/lower confidence intervals for the proportion */
proc freq data=Results noprint;
   by gamma;
   tables Reject / nocum  binomial(level='1');
   output out=Signif binomial;
run;

/* plot the proportion versus the magnitude of gamma */
title "Power of F Test for gamma=0";
title2 "N = &N";
proc sgplot data=Signif noautolegend;
   scatter x=gamma y=_BIN_ / yerrorlower=L_Bin yerrorupper=U_Bin;
   series x=gamma y=_BIN_;
   yaxis values=(0 to 1 by 0.1) grid
         label="Power (1 - P[Type II Error])";
   xaxis label="Gamma" grid;
run;
```

Figure 11 Power Curve for Test of Whether a Coefficient Is 0

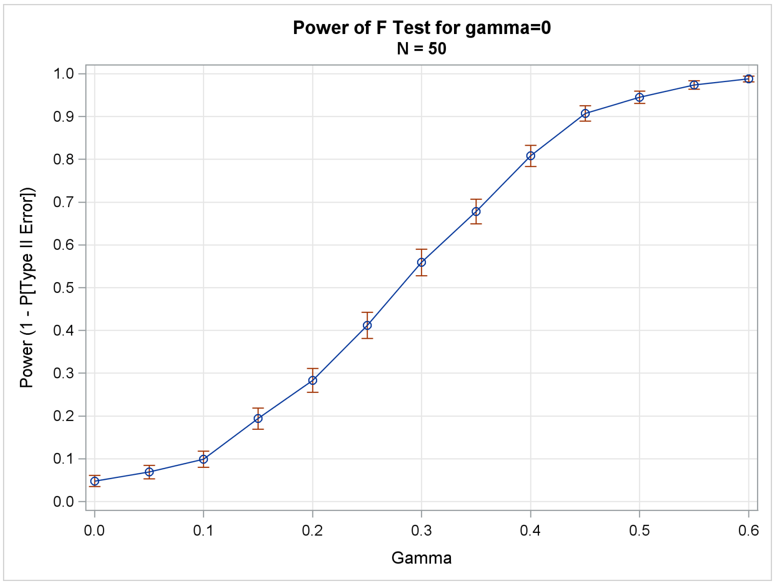


Figure 11 enables you to estimate the power of the F test for various values of γ, the coefficient of the Z variable. As expected, the value of the power at $\gamma = 0$ is $\alpha = 0.05$. When γ is greater than 0.6, the F test almost never commits a Type II error. For intermediate values, the graph enables you to estimate the power. For example, for $\gamma = 1/3$ the power appears to be about 0.65.

CONCLUSION

Although data simulation is a brute-force computational technique, it doesn't have to be slow. The tips in this paper can help you write simulations that run quickly so that you can spend more time analyzing the results and less time waiting for them.

The length of this paper does not permit a discussion about the advantages of using the SAS/IML language for simulation. However, the language is an essential tool for running advanced simulation studies in SAS software. For more information, see Wicklin (2013a), which provides dozens of advanced examples.

REFERENCES

Devroye, L. (1986). *Non-uniform Random Variate Generation*. New York: Springer-Verlag. http://luc.devroye.org/rnbookindex.html.

Greene, W. H. (2000). *Econometric Analysis*. 4th ed. Upper Saddle River, NJ: Prentice-Hall.

Matsumoto, M., and Nishimura, T. (1998). "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-random Number Generator." *ACM Transactions on Modeling and Computer Simulation* 8:3–30.

Novikov, I. (2003). "A Remark on Efficient Simulations in SAS." *Journal of the Royal Statistical Society, Series D* 52:83–86. http://www.jstor.org/stable/4128171.

Tukey, J. W. (1960). "A Survey of Sampling from Contaminated Distributions." In *Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling*, edited by I. Olkin, S. G. Ghurye, W. Hoeffding, W. G. Madow, and H. B. Mann, 448–485. Stanford, CA: Stanford University Press.

Wicklin, R. (2010a). "The DO Loop." http://blogs.sas.com/content/iml/.

Wicklin, R. (2010b). *Statistical Programming with SAS/IML Software*. Cary, NC: SAS Institute Inc.

Wicklin, R. (2013a). *Simulating Data with SAS*. Cary, NC: SAS Institute Inc.

Wicklin, R. (2013b). "Six Reasons You Should Stop Using the RANUNI Function to Generate Random Numbers." July. http://blogs.sas.com/content/iml/2013/07/10/stop-using-ranuni/.

Wicklin, R. (2014). "Simulate Many Samples from a Logistic Regression Model." June. http://blogs.sas.com/content/iml/2014/06/27/simulate-many-samples-from-a-logistic-regression-model/.

ACKNOWLEDGMENTS

The author is grateful to Bob Rodriguez and Phil Gibbs for reading early drafts of this paper. Thanks also to John Castelloe and Stephen Mistler for discussions about power.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author:

Rick Wicklin
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513