SESUG Paper 135-2018

# Introduction to Data Simulation

Jason Brinkley, Abt Associates Inc.

## ABSTRACT

Creating synthetic data via simulation can often be a powerful tool for a wide variety of analyses. The purpose of this workshop is to provide a basic overview of simulating data for a variety of purposes. Examples will include power calculations, sensitivity analysis, and exploring nonstandard analyses. The workshop is designed for the mid-level analyst who has basic knowledge of data management, visualizations and basic statistical analyses such as correlations and t-tests.

## INTRODUCTION

SAS® Software is a powerful analytic platform from which one can not only analyze collected data but also create synthetic data for multiple purposes. Synthetic data can be very useful in exploring new analytic techniques or stress testing certain analyses in a way that does not put individuals at risk for data confidentiality issues. In addition, synthetic data can be created to have a given and known structure which allows the analyst to have some insight into what the resulting output "should" look like and offers opportunities to explore how robust the current techniques are to deviations from perceived norms.

A very popular mechanism for creating synthetic data is to program a data simulation study, which for most statistical programmers often employs so called *Monte Carlo* methods. The idea behind Monte Carlo simulations is to utilize some sort of random number generation process and create repeated samples of synthetic data created from some set of known statistical distributions. Deemed Monte Carlo based on the idea that this methodology was originally set up to explore the impact of repeatedly play a game over and over again to explore properties of games (e.g. casino games). In some cases, Monte Carlo simulation studies allow us to explore the impact of complicated real life scenarios in a way that (while computationally intensive) does not require intricate mathematical formulae.

The goal of this document and corresponding workshop is to introduce existing SAS programmers to the concept of data simulation using features that already exist in most SAS programmers commonly used features. There are many opportunities for optimizing code but the idea here is to cover concepts in an easy to digest way and as such efficiency is not always the primary goal. The focus will be on using the Data step and features available in the base SAS/STAT software. While having some background in statistics may help facilitate discussion, it will not be assumed that the reader is well versed in statistics beyond general basic concepts.

## OPTIONS FOR SIMULATING DATA IN SAS

The interested reader should see the text *Simulating Data with SAS*® by Rick Wicklin as the seminal text to explore various ways to simulate data using SAS software. Wicklin describes the book as "a how-to book for statistical programmers who use SAS software and who want to simulate data efficiently". Wicklin uses a variety of SAS features to simulate data, including the SAS Data step, Proc IML, and the IML Plus programming language available in SAS/IML Studio®.

This current discussion will focus on creating synthetic data using mostly the Data step, however there will be an optional exploration of Proc Simnormal which is available in SAS/STAT software starting with SAS version 9.2.

## TRADITIONAL METHODS FOR SIMULATING IN THE DATA STEP

The traditional method for simulating data in the Data step is to use a combination of do loops and a random number generating function. The following example uses the standard approach:

**Figure 1 – Traditional Method**

```
Data Example1;
       do i = 1 to 10;
       x = rannor(23456);
       output;
       end;
run;
```

This code has created two variables (*i* and *x*) in a working dataset entitled *Example1*. There are 10 observations indexed by *i* and each observation has a value *x* which has been randomly generated from a normal distribution using the function '*rannor*'. The normal distribution is also known as the Gaussian distribution which is a statistical distribution whose data follow the so-called 'bell-curve' which has most of the data congregating in the middle and a smaller number of observations in the 'tails' with as many low observations as high observations. The *rannor* function alone generates simulated data from this distribution centered with mean 0 and standard deviation 1 (also known as a *Standard Normal Distribution* with virtually all data points fall between -3 to 3). The number '*23456*' is known as a random seed which allows us to create random numbers in a way that is replicable and stable between numbers. The seed is an important aspect from the perspective that it allows us to create synthetic data that has an element of randomness but in a way that allows for replication and reproducibility between computers and systems.

## MODERN METHODS FOR SIMULATING IN THE DATA STEP

While the traditional approach for simulating data has been used in SAS for decades, recent exploration has found that the functions such as *rannor* do not behave as randomly as originally thought. Therefore there has been a shift to a new set of functions in what we will refer to as the modern approach:

**Figure 2 – Modern Method**

```
Data Example2;
call streaminit(23456);
       do i = 1 to 10;
       x = rand("Normal",0,1);
       output;
       end;
run;
```

This code has created the exact same kind of data with values *i* and *x* being 10 indexed data observations from a normal distribution with mean 0 and standard deviation 1. The *rand* function is updated to have better random properties and can generate data from a wide variety of statistical distributions. Data would have to be transformed from the *rannor* function in order to have specific means and standard deviations, but the *rand* function allows users to immediately specify the desired mean and standard deviation for the proposed data. Likewise the *streaminit* function initialized the random seed to be used in any iteration of the *rand* function which can be used multiple times in a specific dataset. Consider the following:

**Figure 3 – Multiple Variables**

```
Data Example3;
call streaminit(12345);
      do i = 1 to 100;
      x1 = rand("Normal", 50, 3.2);
      x2 = rand("Binomial", .8,63);
      x3 = rand("Bernoulli", .2);
      x4 = rand("Uniform");
      output;
      end;
run;
```

Here we have created 100 observations (indexed by *i*) with four variables (*x1, x2, x3, x4*) which come from four different statistical distributions:

- The values for *x1* come from a normal distribution (bell-shaped) with mean 50 and standard deviation 3.
- The values for *x2* come from a binomial distribution which counts the number of success in *n* trial with probability of success *p*. Here *p* = 0.8 and *n* = 63 trials; interestingly the binomial distribution becomes more bell-shaped when you have a large number of observations. It *converges* to a normal distribution with a mean of *n* x *p* = 0.8 x 63 = 50.4 and standard deviation square root (*n* x *p* x *(1-p))* = 63 x 0.8 x 0.2 = 3.2.
- The values of *x3* follow a Bernoulli distribution which outputs simple values of 0 or 1 with the probability of a 1 indexed by *p* which here is .2; implying 20% of observations will be 1 and the other 80% will be 0.
- The values of *x4* follow a uniform distribution which can take any number between 0 and 1 with equal probability. This means that approximately 50% of *x4* values should fall at or below 0.5 and 20% of observation should fall at 0.8 or above.

Note that using the same seed via *streaminit* versus using that seed in *rannor* will produce different output. For reproducibility one needs the same seed and the same functions. There is a great number of statistical distributions from which SAS will create synthetic data via simulation but these are among the most commonly used. Bell-shaped data is among the most easily understood so the focus on this introduction will be on that data. The data from *x1* are continuous which means that SAS creates values with many decimal numbers attached (e.g. 50.792700472) but *x2* creates discrete whole numbers that will behave approximately like a bell-curve (e.g. 50 or 51) and these two distributions have approximately the same mean and standard deviation. Likewise, one can create binary data (0, 1) using a Bernoulli distribution where 20% of observations are 1; or one can create uniform data ranging from 0 to 1 and then create a new variable using IF and THEN statements to label 0 for any value below 0.8 and 1 for any value above 0.8. Both of these methods can be used to create (0, 1) data. The uniform data does have an advantage in creating multiple values (say 1, 2, or 3) based on whatever cut-points one may wish to use. While effective, there is perhaps a more efficient way to create such data which will be addressed in a subsequent section.

## CREATING CORRELATED DATA IN THE DATA STEP

While the code in Figure 3 is very useful in creating data where there are multiple variables in the data set, those variables were generated independently. That is to say that while they use the same random seed, the *rand* commands are generated separately and there is no association between the variables in the produced data. Any bivariate analysis or visualizations should show nothing but random scatter. In many cases it is desired to produce two variables that are associated with one another. Most of the advanced techniques to do this go beyond the capacity of the data step and necessitate a more intricate programming language such as Proc IML. There is, however, a simple statistical trick to create two normally distributed variables with a fixed correlation. Figure 4 provides an example of implementation:

**Figure 4 – Two Correlated Normally Distributed Variables**

```
Data Example4;
       call streaminit(5678);
       keep x y;
       mu1=10; mu2=20; var1=4; var2=9; rho=.5;
              do i = 1 to 1000;
              x = rand("Normal", 0,1);
       y = rho*x+sqrt(1-rho**2)*rand("Normal",0,1);

       x = mu1 + sqrt(var1)*x;
       y = mu2 + sqrt(var2)*y;
              output;
              end;
run;
```

The code starts with creating a standard normal distribution of *x* values; but then using the *rand* command to generate *y* with the additional caveats of incorporating *x* and the value *rho* which is the correlation between *x* and *y*. After some recoding, the end result is *x* that has a bell-shaped distribution with mean 10 and standard deviation 2 (square root of 4) and *y* that has a bell-shaped distribution with mean 20 and standard deviation 3 (square root of 9). The major addition from the highlighted code is that *x* and *y* will have a measured correlation of approximated 0.50.

## CREATING MULTIVARIATE NORMAL DATA USING PROC SIMNORMAL (OPTIONAL)

Creating more than two associated variables at a time can be very challenging, especially if you want the associated variables to have a specific structure. In the previous example, it was desired to have two variables whose measured correlation is 0.5. When one moves beyond two variables, the number of parameters to estimate can explode. In a three variable 'multivariate normal' setting (with say *x1*, *x2,* and *x3*) then one needs to fix the means of all three variables, the standard deviation of all three variables, and the correlation between each pair (*x1,x2*), (*x1,x3*), and (*x2,x3*). That is 9 distinct quantities to estimate; in general with *k* different variables one needs $(k^2 +3k)/2$ distinct parameters. Thus for 5 variables, one needs 20 distinct parameters (means, standard deviations, and correlations). This can be quite challenging for a Data step evaluation, but the procedure Proc Simnorm in SAS can generate this data without issues using what the requisite inputs. Consider again the two variable scenario from Figure 4 reconsidered under the Proc Simnorm procedure (special thanks to SAS online documentation) in Figure 5:

**Figure 5 – Two Correlated Normally Distributed Variables Using Proc Simnorm**

```
Data Example5 (type=COV) ;
   input _TYPE_ $ 1-4 _NAME_ $ 9-10 S1 S2 ;
   datalines ;
COV      x1       4   3
COV      x2       3   9
MEAN              10   20
run;

Proc Simnorm data=Example5 outsim=ssim
                numreal = 1000
                seed = 54321 ;
   var x1 x2 ;
run;
```

Here, the data are fed into Proc Simnorm in a matrix style format; the means for *x1* and *x2* are specified as before, however the rest of the data are input via a *variance-covariance* matrix format. The diagonal elements are the variances as denoted before (4 and 9) however the highlighted value is the covariance between the two variables which can be found as:

*Covariance(x1, x2) = Correlation(x1, x2) x standard deviation(x1) x standard deviation(x2)*

In our example *Covariance(x1, x2) = 0.5 x 2 x 3 = 3*. The basic outline for Figure 5 can be used to create multivariate normal data for any number of inputs, provided the means and variance-covariance matrix are properly specified.

## CREATING CATEGORICAL OR TABLE DATA IN THE DATA STEP

Turning away from associated continuous data, let's explore using the *rand* function to create nominal or table data. Figure 6 below illustrates the use of the *rand* command to create data in what SAS calls 'Table' format:

**Figure 6 – Simulating Categorical Data**

```
Data Example6;
call streaminit(234567);
      do j = 1 to 100;
      x1 = rand("Table", .2,.3,.3,.2);
      output;
      end;
run;
```

This code creates a variable *x1* that has four levels (1, 2, 3, 4) where approximately 20% of observations are valued as 1, 30% are level 2, 30% are level 3, and 20% are level 4. The goal with the *Table* command is to create separate levels according to some specified set of proportions that must sum to 1. One can further refine *x1* in the data step to either add a label or transform it into some type of categorical variable (say (A, B, C, D) instead of (1, 2, 3, 4) or (Strongly Disagree, Disagree, Agree, Strongly Agree)). Or one can create table data where the four values match up to the percent occurrence of some set of cells within a table. Suppose that we wanted to create a 2x2 contingency table with rows *X* and columns *Y*; one could number the table cells 1-4 and then apply this simulated code in the way demonstrated in Figure 7 below.

**Figure 7 – Creating Table Data**

```
Data Table;
set Example6;
if x1 = 1 then X = 0;
if x1 = 1 then Y = 0;
if x1 = 2 then X = 0;
if x1 = 2 then Y = 1;
if x1 = 3 then X = 1;
if x1 = 3 then Y = 0;
if x1 = 4 then X = 1;
if x1 = 4 then Y = 1;
keep X Y;
run;
```

In this way we can use the simulated data either as a categorical variable with 4 levels or as two categorical variables with joint occurrence probabilities as outlined in our *rand* function. This naturally extends to any two-way table by simply counting the number of interior cells (so a 2x5 table has 10 cells) and then determining what percentage of the data should occur in each cell.

## MONTE CARLO REPLICATION: DATA OR MACRO?

The preceding sections illustrate easy to use code, mostly within the data step, to create a single synthetic dataset for use in analyses. However, in order to extend oneself into a true *Monte Carlo* simulation study many synthetic datasets should be created and evaluated on the question of interest. In general, there are two ways to do this with one way highly preferred. The less preferred route is to create each dataset individually and perform the necessary calculations and then create and process the next data set and so on. This could be done by coding the process up in one instance and then using macro commands to iterate the process. This is not the preferred route as it is extremely inefficient and uses a lot of computational resources. Instead, SAS prefers simulation studies to generate multiple synthetic datasets at one time and then use a combination of BY statements and ODS output to run analyses on multiple datasets in the same procedure. Examples are illustrated below.

## USING SIMULATED DATA – STATISTICAL POWER CALCULATIONS

A common scenario for using simulated data is in determining statistical power and sample size for complex situations. In many cases formula are available for calculating exact statistical power but as the scenarios from which data might arise, the formula may become less robust. In addition, simulation studies allow the use to change many different inputs quickly and simply re-run code to get power estimates. The example we will use is from basic data generated as below:

**Figure 8 – Simulating Data for Power**

```
Data PowerEx;
call streaminit(4321);

      samplesize = 20;
      do sample = 1 to 100;
            do group = 1 to 2;
            do i = 1 to samplesize;
            if group = 1 then x = rand("Normal", 50, 10);
            if group = 2 then x = rand("Normal", 40, 10);
            output;
            end;
            end;
      end;
run;
```

The purpose of this example is to create synthetic data from two groups that have different statistical distributions, in this case the data come from normal distributions both with standard deviation 10 but with different means (40 and 50). We generate 20 observations from each distribution and we repeat this process 100 times. We see many of the data generation techniques being used here with a couple of key differences. The goal in a simulation study is to perform a procedure many times and explore sample to sample variation in results. So with a small sample size and two groups for comparison, it might be desired to perform a t-test on this data. By creating 100 distinct datasets we can actually perform 100 distinct tests and use those results to tell us how effective a t-test is in this scenario.

So the figure below shows a commonly used technique in simulation studies to explore a tests properties. ODS options are used to select a specific table from the Proc ttest output. Then the procedure is run using a by statement for sample which while time consuming is many more times efficient than to run the ttest procedure many times for each dataset. Here ODS provides output for two tests under conditions of

'Equal Variances' and 'Unequal Variances'. The sort procedure aligns the output for the right tests and the subsequent data set counts the number of instances for which we get a p-value (*Probt*) that is less than a specific significance level (here specified as 0.05). The results for this simulation suggest that approximately 87% of simulated datasets find statistically significant differences between the two groups using a t-test, which is our estimate for statistical power. With the programming done, the analyst can rerun the code across many instances of sample size or change the standard deviation values to explore equal and unequal variance tests or change the significant level to determine what impact that may have on results.

**Figure 9 – Proc ttest Across Many Datasets**

```
ods output ttests=tests;

proc ttest plots=none;
by sample;
class group;
var x;
run;

Proc sort data=tests;
by Variances;
run;

Data tests;
set tests;
reject = 0;
if Probt < 0.05 then reject = 1;
run;
```

## USING SIMULATED DATA – ADDING VARIATION TO DATA

An alternate way for which one can use synthetic data is to help create 'variations' of a given dataset. Suppose there already exists a given dataset and one needs to create a slightly different version of the data in order to explore some feature or metric. This can be particularly useful if the metric of interest does not have stable or standard statistical properties. In this way one can explore how versatile a measure is using only one example dataset that can be altered in ways to help highlights strengths or weaknesses in the measure. In Figure 10, data from a binomial distribution is again simulated to give us some count data that falls between approximately 45 to 55; the question of interest is how we can create other 'similar' data sets to this dataset which is a very different question than whether we can recreate similar datasets from some statistical distribution.

**Figure 10 – Example Data for Manipulation**

```
Data InputData;
call streaminit(12345);
      do i = 1 to 100;
      x1 = rand("Binomial", .8,63);
      output;
      end;
run;
```

Figure 11 illustrates a mechanism by which we can use the already discussed commands to create some noise to be added to the existing data. Here we create two scenarios where we create integers that range from -2 to 2 at random. The first scenario creates an equal number of each integer while scenario 2 creates mostly -2 and +2 values. Just like in the power example, we create multiple datasets. But then we merge the noise data to the original data to create multiple 'new' datasets that are similar to the original data but with the values shifted along the noise range at random. The original data has an approximate bell shape, the noise added in scenario 1 will do very little to the shape of the distribution but will add extra variation while scenario 2 tends to make values more extreme impacting the tail ends of the distribution, impacting both shape and variation. These characteristics may play a big role in whatever it is that is the measurement intent.

**Figure 11 – Simulating Noise and Combining Data**

```
Data NoiseData;
call streaminit(23456);
      do scenario = 1 to 2;
      do j = 1 to 1000;
      do i = 1 to 100;

      *scenario 1 - even noise;
      if scenario =1 then x2 = rand("Table", .2,.2,.2,.2,.2);

      *scenario 2 - uneven and extreme noise;
      if scenario =2 then x2 = rand("Table", .35,.1,.1,.1,.35);
      noise = 3 - x2;
      output;
      end;
      end;
      end;
      drop x2;
run;

proc sort;
by i;
run;

Data Combined;
merge NoiseData InputData;
by i;
x1_noise = x1 + noise;
run;
```

## CONCLUDING REMARKS

The goal of this document and workshop is to provide a soft introduction to data simulation using the SAS Data step which can create a versatile range of different synthetic data. Applications extend well beyond those listed here but hopefully this provides a starting point for programmers to think through the many different ways that synthetic data can be created and utilized in important settings. Synthetic data has value in understanding new procedures or in helping to explore the deviations current output might have when compared to the same analyses performed under alternate data conditions. While the use of statistical simulation studies was once mostly the purview of statisticians looking to expand or justify new methodology, the practical use of simulated data in real world contexts is only now becoming apparent as a way to do quality control, test systems performance, or better understand commonly used metrics.

## REFERENCES

Wicklin, R. (2013). *Simulating data with SAS*. SAS Institute.

SAS Institute Inc., SAS 9.4 Help and Documentation, Cary, NC: SAS Institute Inc., 2002-2016.

## ACKNOWLEDGMENTS

Special thanks to the SAS Online Documentation and to Rick Wicklin's book Simulating Data with SAS® as well as his blog, the Do Loop (http://blogs.sas.com/content/iml/) which continues to be a source of inspiration for those who enjoy SAS programming and simulation.

## CONTACT INFORMATION

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

Name: Jason Brinkley
Enterprise: Abt Associates Inc
Address: 5001 South Miami Blvd
City, State ZIP: Durham, NC 27703
E-mail: Jason_Brinkley@abtassoc.com