SAS® GLOBAL FORUM 2020 #SASGF

SAS4400-2020

Simulating Data for Complex Linear Models

Phil Gibbs and Kathleen Kiernan, SAS Institute Inc.

ABSTRACT

One of the core tools of any statistician is working with linear models, from simple or multiple regression models to more complex, generalized linear mixed models. Data simulation enables you to be more comfortable with new types of models, by providing data to a model that will give known results. This paper introduces you to the random number functions in the SAS® DATA step and shows you how to construct programs to simulate data for a variety of models. The paper briefly discusses basic areas of data simulation, such as linear regression. The paper then shows how to generate data for more complex mixed models, including repeated measures models under a variety of within-subject covariance structures. Also, the paper covers generalized linear mixed models like logistic and Poisson. Finally, nonlinear mixed models are discussed. After reading this paper, you should be able to better understand models with difficult parameterizations, especially in the covariance structure.

INTRODUCTION

Data simulation writes code that can generate a random sample of data for a statistical model. Simulation is a powerful tool for helping any statistician to better understand the structure of statistical models. By using simulation, you can see how the model pieces fit together to predict a response. With data that is simulated, you already know the approximate results of a model fit even before the parameter estimation is done. A quick look at the model results tells you if you have programmed the simulation correctly. The values that you use in the simulation code should match the parameters that you see in the SAS procedure output. You can even identify situations where parameters are confounded or inseparable.

Another area where simulation is helpful is in prospective power analysis. Many power analyses have closed-form solutions, meaning that the power calculations can be done using formulas derived from the statistical model. SAS® Studio includes tasks within the software for many of these situations. If the power analysis does not have a closed-form solution, then simulation can approximate the power of a statistical test. See Chapter 4 of *SAS for Mixed Models* (Stroup et al. 2018) for an example of this use of simulation.

This paper briefly introduces the simulation tools that are available in SAS. You will learn how to use simulation in more complex models. The paper discusses linear and generalized linear models and how to simulate data in terms of both R-side and G-side covariance structures. The paper also covers zero-inflated models and nonlinear mixed models.

RESOURCES FOR SIMULATING DATA IN SAS®

This paper uses the SAS DATA step for most of the examples. SAS/IML® software can be used for simulation as well. Wicklin (2013) is a great resource that discusses how to use SAS/IML in simulations. Although DATA step code is easier to interpret, SAS/IML code is more efficient in producing simulation results. In addition, some simulation tasks are easier to accomplish with SAS/IML. Simulating high-dimensional correlated data is one such task. This type of simulation uses matrix operations on the covariance matrix of the desired data.

The primary tool for simulation in the DATA step is the RAND function. The RAND function simulates data from 29 distributions, including normal, exponential, Poisson, and Gamma. The RAND function uses a hybrid Mersenne-Twister algorithm that provides good performance and good randomness traits in the streams that it generates. For more details about the RAND function and the available algorithms for generating random number streams, see the SAS® 9.4 documentation (SAS Institute Inc. 2019a).

You can use older linear congruential-style random number generators (as used in functions like RANUNI, RANNOR, RANEXP, and so on) in simulation programs. However, the random streams provided by the RAND function are far superior, so you should use the RAND function in place of these older functions.

SAS does have procedures that simulate random numbers. In SAS/STAT® software, the SIMNORMAL procedure generates multivariate random normal variates while the SIM2D procedure simulates spatial data in a random Gaussian field for two dimensions. The new SIMSYSTEM procedure in SAS® Visual Statistics 8.5 simulates data from distributions in the Johnson and Pearson systems. Simulation using PROC SIMSYSTEM is extremely flexible, enabling the specification of the distribution moments rather than a named statistical distribution.

GETTING STARTED: A LINEAR MODEL WITH CLASS EFFECTS AND COVARIATES

This section of the paper reviews simulation techniques that later sections of the paper demonstrate in more complex ways. This example uses a linear model that involves a class effect with three treatment levels, and two covariates, X1 and X2. The functional form of the model is as follows:

$$Y_{ij} = \mu + TRT_i + 2.5*X1_{ij} + 1.7*X2_{ij} + \varepsilon_{ij}$$

X1 follows a uniform distribution on the interval [1,3], and X2 follows a normal distribution with a mean of 4.3 and a variance of 1.21. The intercept of the model, μ, is 1.5. The residual variance, which is the variance of ε_{ij}, is 2.25. The simulation generates 50 replications (reps) of 200 observations, or 10,000 total observations.

Here is the SAS code for this simulation:

```
data simulation;
   array trt{3} trt1-trt3 (10 30 20);
   call streaminit(689127);
   do rep=1 to 50;
      do i=1 to 200;
         x1=rand("uniform",1,3);
         x2=rand("normal",4.3,1.1);
         e=rand("normal",0,1.5);
         y=1.5 + trt{rand("integer",1,3)} + 2.5*x1 + 1.7*x2 + e;
         output;
       end;
    end;
run;
```

This simulation uses an array structure in the DATA step to store parameter values for the treatment effect. The first treatment level (TRT1) sees an adjustment to the intercept of the model of 10 units, the second level (TRT2) sees an adjustment of 30 units, and the third level (TRT3) sees an adjustment of 20 units. The array provides a compact and efficient way to specify the treatment effect for this model.

The CALL STREAMINIT routine provides a seed to begin the simulation. The seed value, when using the default hybrid Mersenne-Twister algorithm for generating random numbers, can be any positive integer value that is less than **`4294967295 (2**32-1)`**.

The RAND function generates the values for the covariates, X1 and X2, as well as for the residual error, ε, for each observation. Observations generated from the uniform distribution default to the interval [0,1], with the ability to specify different endpoints to that interval as parameters in the function call. The normal distribution defaults to the standard normal, with a mean of 0 and a standard deviation of 1. With the use of function parameters, the function can specify a different mean and standard deviation.

The RAND function also generates random levels for the treatment by using the INTEGER distribution. The INTEGER distribution returns a random integer on the interval, including the minimum and maximum values specified as the two arguments on the RAND function call. Here, the minimum value is 1 and the maximum value is 3, which generates random integers from the set {1,2,3}.

The most efficient way to generate this simulation and process the results is to generate all the observations in a single SAS data set and process the simulation through BY processing in the model-fitting SAS procedure that is used. The data can be simulated and processed one simulation replication at a time, but that approach can take hours of system CPU time. Simulation of all the data at once and taking advantage of BY processing can cut model fit time by many orders of magnitude. However, for very large simulations, disk space becomes an issue. In some instances, you might need to split the simulation into smaller pieces to avoid storage constraints.

After the simulation runs, it is easy to see whether the data and code have produced the correct results. Use the UNIVARIATE procedure to check the distribution of the covariates X1 and X2 and the residual error ε. Check the moments of each distribution and use plots of the generated data to make sure that these results are generated correctly. The goodness-of-fit tests for distributional fit from PROC UNIVARIATE are very powerful and can detect any small departures from the distribution. You can ignore those tests for most simulation work. However, if you choose to use them, interpret them with considerable caution.

Use the GLM procedure and BY-group processing to generate parameter estimates for each sample. You should suppress SAS® Output Delivery System (ODS) tables and graphs during this step (Wicklin 2013, p. 97). PROC UNIVARIATE can assess the distribution of the estimates of these parameters.

A simulation like this example can build a bootstrap analysis of the parameters or of quantiles for the resulting dependent variable. The remaining simulation examples in this paper use a single replication but can be built in a similar fashion.

RANDOM EFFECT MODELS: SPLIT-PLOT DESIGNS

Specifying fixed effects in a simulation is simple to do, because the values needed for fixed effects are hardcoded into the simulation. More complicated simulations require the use of random effects. Specifying levels for G-side random effects is not quite as easy as it is to do for those hardcoded fixed effects.

A split-plot model is a simple example of a model involving both fixed and random effects. Example 81.1 in the MIXED procedure chapter of the *SAS/STAT 15.1 User's Guide* (SAS Institute Inc. 2019b) is a gentle introduction to random effects and the split-plot design. The design in that chapter has two experimental factors, *A* and *B*. Levels of *A* are randomly assigned at the whole-plot level, and levels of *B* are assigned to the split-plot levels within each whole plot. The design results in a balanced split-plot model, and the whole plots are arranged in a randomized complete block structure.

The model for the design in the code below involves those two fixed effects, *A* and *B*, as well as the random terms *BLOCK* and *A*BLOCK*. Stroup (1989) provides data for this design. You can simulate data for this balanced design in a DATA step, as shown here:

```
data SplitPlot;
   array AEffect{3} A1-A3 (10 15 10);
   array BEffect{2} B1-B2 (5 10);
   array AxBEffect{6} AB1-AB6 (2 4 2 1 1 0);
   call streaminit(78124523);
   do block = 1 to 100;
      RandBlock = rand('normal',0,7.9);
      do A = 1 to 3;
         RandAxBlock = rand('normal',0,3.923);
         do B = 1 to 2;
            e = rand('normal',0,3.06);
            y = 10 + AEffect{A} + BEffect{B} + AxBEffect{(A-1)*2 + B} +
            RandBlock + RandAxBlock + E;
            output;
         end;
      end;
    end;
run;
```

You can vary the number of blocks to fit the simulation needs. A large number of blocks allows for verification of the simulation results. The *BLOCK* and *A*BLOCK* variances should be close to the values that are specified in the simulation (accounting for sampling variation). The parameter estimates or least square means (LS-means) for the fixed effects should be very close to the simulated values.

You can construct the LS-means for the *A*B* interaction from the model parameters in the simulation. Table 1 presents those calculations.

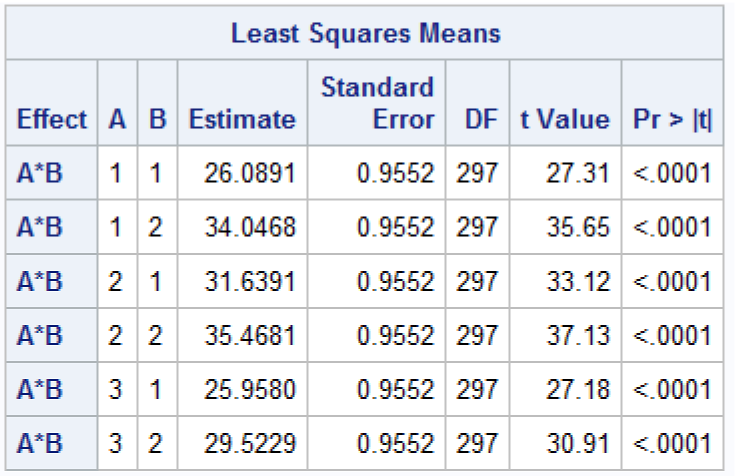
Effect	**LS-means Calculations**	**LS-means Simulations**
*A*1*B*1	10 + 10 + 5 + 2	27
*A*1*B*2	10 + 10 + 10 + 4	34
*A*2*B*1	10 + 15 + 5 + 2	32
*A*2*B*2	10 + 15 + 10 + 1	36
*A*3*B*1	10 + 10 + 5 + 1	26
*A*3*B*2	10 + 10 + 10 + 0	30

Table 1. LS-means Calculations for the Split Plot in a Randomized Complete Block Design

You can create output from the LSMEANS statement with the following code (see Output 1):

```
proc mixed data=SplitPlot;
   class a b block;
   model y = a b a*b / s;
   random block a*block;
   lsmeans a*b;
run;
```

As you can see in the output, the simulated values agree very well with the fitted LS-means, indicating that the fixed-effects portion of this simulation is programmed correctly.



Least Squares Means							
Effect	A	B	Estimate	Standard Error	DF	t Value	Pr > |t|
A*B	1	1	26.0891	0.9552	297	27.31	<.0001
A*B	1	2	34.0468	0.9552	297	35.65	<.0001
A*B	2	1	31.6391	0.9552	297	33.12	<.0001
A*B	2	2	35.4681	0.9552	297	37.13	<.0001
A*B	3	1	25.9580	0.9552	297	27.18	<.0001
A*B	3	2	29.5229	0.9552	297	30.91	<.0001

Output 1. Results from the LSMEANS Statement

A slightly more complicated example of a split-plot design is presented in *SAS for Mixed Models* (Stroup et al. 2018). This example involves a semiconductor study. Several modes of a process condition (ET) are studied for their effect on resistance in computer chips. A random set of wafers is chosen, and three wafers are assigned to each of four modes of ET. Resistances are measured on each of four positions (POS) on each wafer after processing.

In the following code, ET and POS are considered fixed effects. WAFER is a random effect. Either PROC MIXED or the GLIMMIX procedure can fit the appropriate model for this experiment:

```
proc glimmix data=ResistanceStudy;
    class ET Wafer Pos;
    model Resistance = ET Pos ET*Pos;
    random int / subject=Wafer(ET);
run;
```

There are several good approaches for simulating data for this model. The following example uses the SAS DATA step and breaks the overall task of this simulation down into manageable pieces of code. The first step is to generate the realizations of the random effect of WAFER by using the RAND function. Random effects in a mixed model are assumed to follow a normal distribution with a mean of 0. This example applies a variance of 2.25 to the WAFER-level residuals. The *RANDSORT* variable holds a random uniform value that will be used to assign wafers to the modes of ET:

```
data Wafers;
   call streaminit(789510345);
   do Wafer=1 to 12;
      RandSort=rand('uniform');
      WaferEffect=rand('normal',0,1.5);
      output;
   end;
run;

proc sort data=Wafers;
    by RandSort;
run;
```

Because the realizations of the random levels of WaferEffect are already random, this sorting step is not necessary. But it is included here to show one of the ways that wafers could be randomly assigned to levels of the process condition ET.

Data for the rest of the design is generated by hardcoding levels for the fixed effects using arrays. Modes 1–4 of ET are assigned to the randomly sorted WAFER values. Each of the four POS levels is generated for each wafer, random error is generated for each observation (from a normal distribution with a mean of 0 and a variance of 0.49), and the linear predictor is built from the form of the model as specified in the PROC GLIMMIX model above:

```
data ResistanceStudy(keep=Wafer ET Pos Resistance);
   array ETEffect{4} ET1-ET4 (10 12 8 12);
   array PosEffect{4} Pos1-Pos4 (2.5 2 1.5 2);
   set Wafers;
   call streaminit(153478);
   ET=ceil(4*_n_/12);
   do Pos=1 to 4;
      e = rand('normal',0,.7);
      Resistance=3 + ETEffect{ET} + PosEffect{Pos} +
      ETEffect{ET}*PosEffect{Pos} + WaferEffect + e;
      output;
   end;
run;
```

Note the use of the KEEP= data set option in the DATA statement. It is essential, especially in large simulations, to keep only the variables that are needed in the simulation. The fixed-effects parameters for ET1–ET4, POS1–POS4, and the value of the residual ε—while important in simulating the response for this model—can all be dropped from the data set after the data is ready to be used in a simulation study. For the savvy DATA step programmer, temporary array elements can be used for the parameters on ET and POS. These temporary elements do not appear in the output data set for the simulation (Wicklin 2013).

An important aside here is that while the parameter estimates of POS1–POS4 and the residual variance can be estimated well from this simulation, the parameters of ET1–ET4 are more difficult to replicate. There are only 12 realizations of the random effect of WAFER, with three each assigned to each level of ET. One of the assumptions of this random effect model is that these realizations for WAFER will follow a normal distribution with a mean of 0 for each level of ET. With only three realizations of this normal distribution for each ET, it is likely that the mean of those realizations will not be 0, which can interfere in duplicating the expected parameter estimates for ET.

REPEATED MEASURES MODELS AND R-SIDE COVARIANCE STRUCTURES

Simulating data for repeated measures models requires more complexity in the simulation code. R-side covariance structures need to specify the covariance matrix for the repeated effect. You can specify the covariance matrix either by using the DATA step and PROC SIMNORMAL or by using the IML procedure; the latter method is more straightforward.

Here is the specification in PROC IML for one of the simpler structures, compound symmetry, for a three-level repeated measure effect:

```
proc iml;
   cov={4 2 2,
```

(code continued)

```
          2 4 2,
          2 2 4};
     mean={0 0 0};
     n=2000;
     call randseed(61345);
     z=RandNormal(n,mean,Cov);

     create CSErrors from z;
     append from z;
  quit;
```

The compound symmetry structure has the variances of the three repeated levels along the diagonal and the common covariance between the levels in the off-diagonal elements. Repeated measures models assume that the mean of the residual errors is 0. The RANDSEED function is used to specify a seed value, so that these results can be replicated. The RANDNORMAL function takes the number of simulation replications, the mean vector, and covariance matrix as arguments and produces *N* realizations of the random errors. If these realizations should be generated to a SAS data set, use the CREATE and APPEND statements to move the data from a SAS/IML matrix. To check that the errors are simulated correctly in SAS/IML software, use the MEAN and COV functions:

```
  SampleMean=mean(z);
  SampleCov=cov(z);
  print SampleMean, SampleCov;
```

Alternatively, you can use PROC SIMNORMAL to generate repeated measures error structures in SAS. The specification of the covariance matrix and mean vector is done by using a special SAS data set structure. The *_TYPE_* variable indicates the role of each observation in this data set. The *_NAME_* variable indicates the order of the rows of the covariance matrix. To check that the simulation produced the correct error structure, use the CORR procedure with the COV option.

Here is an example that uses a 3x3 unstructured covariance matrix:

```
  data UNCov;
     input  _type_  $ _name_ $ x1 x2 x3;
     datalines;
  COV X1 4 2 3
  COV X2 2 2 1
  COV X3 3 1 5
  MEAN .  0 0 0
  ;
  run;

  proc simnormal data=UNCov(type=cov) out=simdata numr=2000 seed=6512345;
     var x1 x2 x3;
     run;

  proc corr data=simdata cov;
     var x1 x2 x3;
  run;
```

Now that you know how to simulate correlated errors for a repeated measures model, here is a full example. Suppose that you have a treatment effect with three levels. A pool of 200 subjects is randomly assigned to the treatment levels. Also, assume that the covariance structure for the R-side errors is unstructured. The R-side errors are generated by using PROC IML:

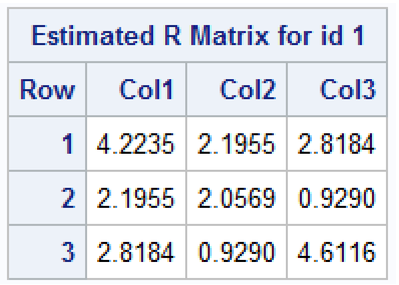
```
proc iml;
   cov={4 2 3,
        2 2 1,
        3 1 5};
   mean={0 0 0};
   n=200;
   call randseed(61345);
   z=RandNormal(n,mean,Cov);
   create UNErrors from z;
   append from z;
quit;
```

You can add in the treatment effect by using a SAS DATA step and then run the associated mixed model:

```
data test;
    retain id 0;
    array e{3} col1-col3;
    array TRTEffect{3} trt1-trt3 (10 15 12);
     set UNErrors;
     call streaminit(561469);
     id+1;
     trt=rand("integer",1,3);
     do i= 1 to 3;
        y=5 + TRTEffect{trt} + e{i};
        output;
     end;
     keep id trt y;
run;

/* run the model */
proc mixed data=test;
    class id trt;
    model y= trt/ s ddfm=kr;
    repeated / type=un subject=id r;
    lsmeans trt;
run;
```

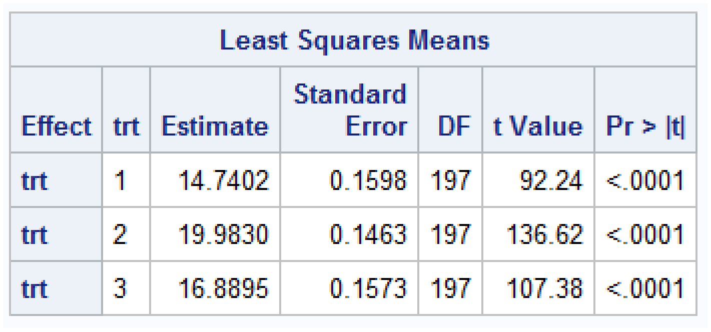
The PROC MIXED output shows that the simulation produced the results desired. Output 2 shows the estimated covariance matrix for the R-side errors, and Output 3 shows the LS-means for the treatment levels.



Estimated R Matrix for id 1

Row	Col1	Col2	Col3
1	4.2235	2.1955	2.8184
2	2.1955	2.0569	0.9290
3	2.8184	0.9290	4.6116

Output 2. Estimated Covariance Matrix



Least Squares Means						
Effect	trt	Estimate	Standard Error	DF	t Value	Pr > |t|
trt	1	14.7402	0.1598	197	92.24	<.0001
trt	2	19.9830	0.1463	197	136.62	<.0001
trt	3	16.8895	0.1573	197	107.38	<.0001

Output 3. LS-means for Treatment Levels

If the R-side covariance structure is AR(1), then those errors can be generated by PROC IML. The AR(1) structure requires a residual variance and a correlation parameter. The covariances are generated using a power structure based on the residual variance, correlation, and number of time points (five, in this example):

```
proc iml worksize=100;
   resid=2;
   rho=.8;
   m=5;
   cov=j(m,m,1);
   do i=1 to m;
      do j=1 to m;
         cov[i,j]=resid*rho**abs(i-j);
      end;
   end;
   mean={0 0 0 0 0};
   n=200;
   call randseed(61345);
   z=RandNormal(n,mean,Cov);
   create AR1Errors from z;
   append from z;
quit;
```

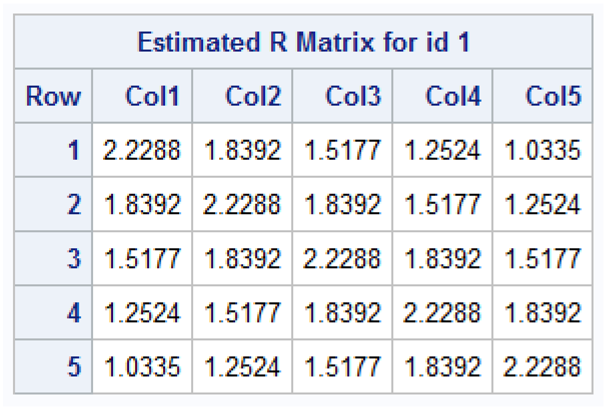
You can add the treatment effect and estimate the mixed model by using the approach that was in the previous SAS DATA step example:

```
  data test;
     retain id 0;
     array e{5} col1-col5;
     array TRTEffect{3} trt1-trt3 (10 15 12);
     set AR1Errors;
     call streaminit(561469);
     id+1;
     trt=rand("integer",1,3);
     do i= 1 to 5;
        y=5 + TRTEffect{trt} + e{i};
        output;
     end;
     keep id trt y;
  run;
```

(code continued)

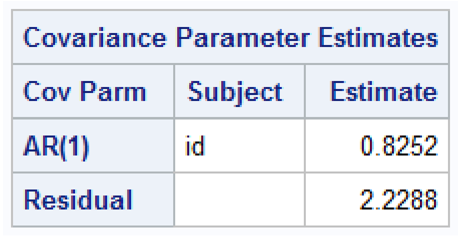
```
proc mixed data=test;
   class id trt;
   model y= trt/ s ddfm=kr;
   repeated / type=ar(1) subject=id r;
   lsmeans trt;
run;
```

Output 4 shows the estimated covariance matrix for the R-side errors for this AR(1) structure. It is easy to see the decay in the covariance over the five levels of the repeated effect. Output 5 shows the estimated covariance parameters for this model, matching the simulated parameters from the PROC IML code.



Estimated R Matrix for id 1					
Row	Col1	Col2	Col3	Col4	Col5
1	2.2288	1.8392	1.5177	1.2524	1.0335
2	1.8392	2.2288	1.8392	1.5177	1.2524
3	1.5177	1.8392	2.2288	1.8392	1.5177
4	1.2524	1.5177	1.8392	2.2288	1.8392
5	1.0335	1.2524	1.5177	1.8392	2.2288

Output 4. Estimated Covariance Matrix for R-side Errors



Covariance Parameter Estimates		
Cov Parm	Subject	Estimate
AR(1)	id	0.8252
Residual		2.2288

Output 5. Estimated Covariance Parameters

HIERARCHICAL LINEAR MODELS: ADDING LAYERS OF CORRELATION

Hierarchical linear models, or HLMs, are very popular in the social sciences. These models have layers of correlated errors. An easy-to-understand example of an HLM comes from education-related data. When you collect test scores for students, you can expect that students with the same teachers have correlated scores, at least more so than students who had different teachers. Additional correlation could be expected at the school level, district level, and perhaps even state level. This correlation structure can be accommodated through a series of RANDOM statements in PROC MIXED. One RANDOM statement is used for each level in the hierarchy:

```
random int / subject=state;
random int / subject=district(state);
random int / subject=school(district state);
random int / subject=teacher(school district state);
```

The first RANDOM statement correlates all the students' scores from the same state. The second RANDOM statement further correlates the students' scores from the same district. The additional RANDOM statements add the layers of correlation from scores from the same school and teacher.

Fortunately, simulating data for this type of model is easy to do in the SAS DATA step. A series of DO loops quickly and efficiently produces the data that you need. For example, a three-level hierarchical structure has 10 reps for the outermost layer, four reps for each of those 10 reps for the middle layer, and two reps for each of the four middle reps for the innermost layer. In terms of the education model, this structure could represent 10 school districts, with four schools in each, and two students within each school. The students are measured over five time periods. Also, a covariate is measured at each level of the hierarchy.

The fixed-effect structure of the model is an intercept and a slope on the time effect. The hierarchical structure of the model adjusts these intercepts and slopes for each level of the hierarchy, adding a random adjustment to each intercept and slope for each hierarchical level. The code below generates data for this simulation:

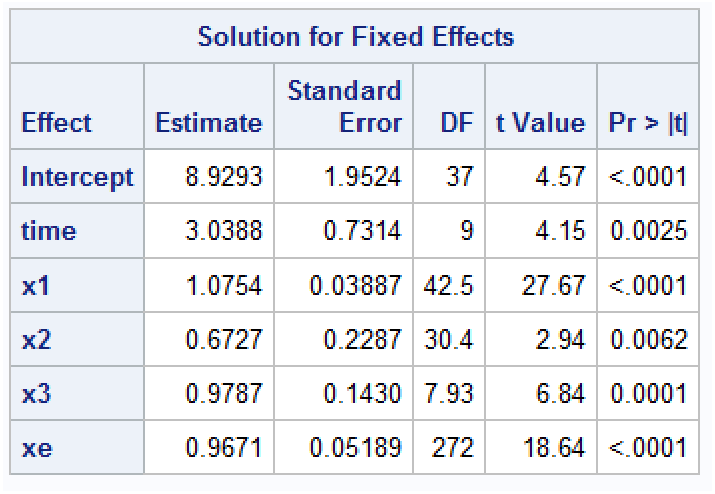
```
data hlm3;
    nlevel3=10;
    nlevel2=4;
    nlevel1=2;
    call streaminit(92821);
    do level3=1 to nlevel3;
       r_slp3=rand("normal",0,2);    /* variance of slopes 4 */
       r_int3=rand("normal",0,1.5);  /* variance of intercepts 2.25 */
       x3=rand("uniform",0,10);
       do level2=1 to nlevel2;
          r_slp2=rand("normal",0,1);  /* variance of slopes 1 */
          r_int2=rand("normal",0,2);  /* variance of intercepts 4 */
          x2=rand("uniform",5,10);
           do level1=1 to nlevel1;
              r_slp1=rand("normal",0,1.2);
              r_int1=rand("normal",0,.7);
              x1=rand("uniform",-5,5);
              do time=1 to 5;
                 xe=rand("uniform",0,2);
                 e=rand("normal",0,.5);
                 y=(5 + r_int1 + r_int2 + r_int3) + (2 + r_slp1 + r_slp2
                 + r_slp3)*time + (xe + x1 + x2 + x3) + e;
                 output;
              end;
           end;
        end;
    end;
    keep level3 level2 level1 time x1 x2 x3 xe y
run;

proc mixed data=hlm3 namelen=32;
    class level3 level2 level1;
    model y=time x1 x2 x3 xe / s ddfm=kr;
    random int time / subject=level3;
    random int time / subject=level2(level3);
    random int time / subject=level1(level2 level3);
run;
```

For each LEVEL3 value, an adjustment to the slope on time and intercept is generated. The process repeats for each of the LEVEL2 and LEVEL1 values. Notice that the realization for the adjustment to the slope and intercept for each value of LEVEL3 is the same for all observations that share that LEVEL3 value. That structure is key to simulating this data correctly.

The line that creates the response variable shows how the adjustments to the slope on time and intercept for the model come together. The DDFM=KR model option gives the best degrees of freedom (DF) calculation for this model type. However, this calculation is useful only for small data sets. Larger simulations need to use the DDFM=BW or DDFM=CONTAIN model options to facilitate a quicker run time when estimating the model.

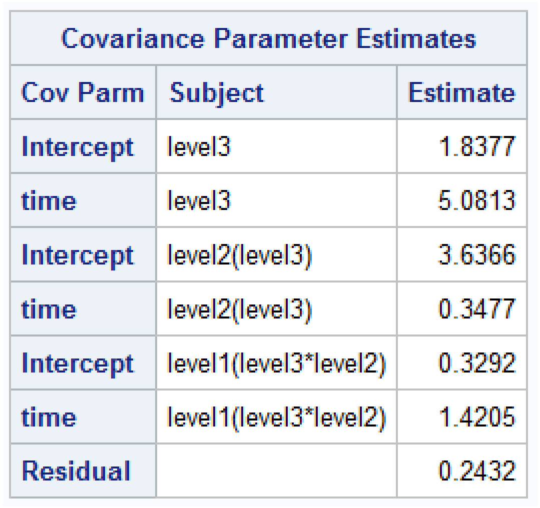
The PROC MIXED output (see Output 6, Output 7, and Output 8) shows that the parameter estimates for the fixed and random effects are close to the simulated values. These complicated covariance structures require large sample sizes in order to accurately match the simulated values with a good degree of precision. Another takeaway from the output is to watch the change in the DF for the fixed-effect tests. There is much more precision available for results measured at the residual level than those results measured at LEVEL3 (the X3 covariate, for example). Also notice that the residual and LEVEL1 covariance estimates more closely match their simulated values.



Solution for Fixed Effects

Effect	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	8.9293	1.9524	37	4.57	<.0001
time	3.0388	0.7314	9	4.15	0.0025
x1	1.0754	0.03887	42.5	27.67	<.0001
x2	0.6727	0.2287	30.4	2.94	0.0062
x3	0.9787	0.1430	7.93	6.84	0.0001
xe	0.9671	0.05189	272	18.64	<.0001

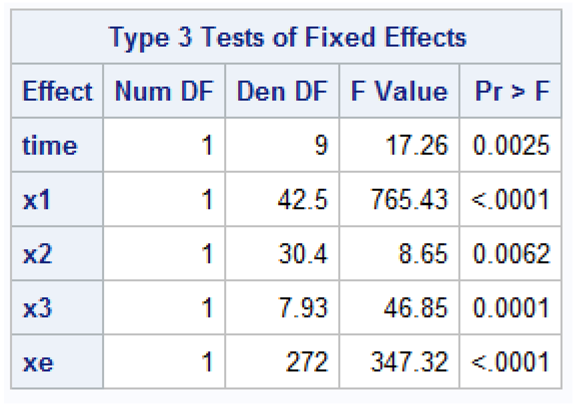
Output 6. Fixed-Effect Parameter Estimates for the HLM



Covariance Parameter Estimates

Cov Parm	Subject	Estimate
Intercept	level3	1.8377
time	level3	5.0813
Intercept	level2(level3)	3.6366
time	level2(level3)	0.3477
Intercept	level1(level3*level2)	0.3292
time	level1(level3*level2)	1.4205
Residual		0.2432

Output 7. Covariance Parameter Estimates for the HLM



Type 3 Tests of Fixed Effects				
Effect	Num DF	Den DF	F Value	Pr > F
time	1	9	17.26	0.0025
x1	1	42.5	765.43	<.0001
x2	1	30.4	8.65	0.0062
x3	1	7.93	46.85	0.0001
xe	1	272	347.32	<.0001

Output 8. Tests of Fixed-Effect Covariates

A LOGISTIC REGRESSION MODEL WITH A RANDOM EFFECT

Simulating data for a generalized linear model adds an extra layer of complexity. A linear predictor is simulated, like what is done for a linear regression model. The LINK= function is applied to this linear predictor, transforming the linear predictor to the scaled response for the generalized linear model.

A logistic regression model shows how this works. Suppose that you want to use PROC GLIMMIX to estimate the model, as shown here:

```
proc glimmix data=test;
   class trt grp;
   model y=trt x1 x2 / link=logit dist=binomial s;
   random int / subject=grp;
   lsmeans trt / at x1=0 at x2=0;
   nloptions tech=nrridg;
run;
```

Assume that X1 and X2 are uniform covariates. Also, the treatment effect has three levels, and there are 200 levels in the random GRP model effect. There are 20 observations for each GRP level.

The code for this simulation is as follows:

```
data test;
   call streaminit(25345278);
   do grp=1 to 200;
      rgrp=rand('normal',0,.7);
      do i=1 to 20;
         x1=rand('uniform');
         x2=rand('uniform');
         trt=rand('integer',1,3);
         logit=-2 + 2*x1 + x2 + (trt-2) + rgrp;
         p=exp(-logit)/(1+exp(-logit));
         if rand('uniform')>p then y=1; else y=0;
         output;
      end;
    end;
    keep trt grp x1 x2 y;
run;
```

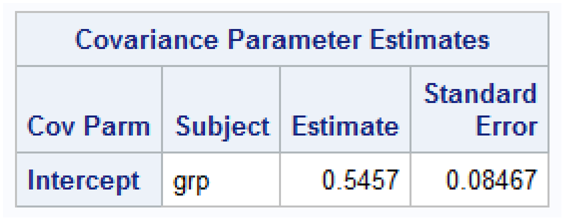
The simulation uses a variance of 0.49 for the random GRP effect. Slopes of 2 and 1 are used for the two uniform covariates, X1 and X2. The treatment effect is simulated randomly as an integer between 1 and 3, resulting in an unbalanced design. With a generalized linear model, even more simulation replications are needed to validate that the simulation is working correctly. The 200 levels for the GRP effect should enable validation of the results.

Rather than use an array to enter the simulated values for the treatment effect levels, use an algebraic calculation to provide the simulated values. Here, the overall intercept is -2, and the intercept is adjusted for each treatment level with the calculation (*trt – 2*). With the GLM parameterization of the treatment effect (setting the intercept to the last treatment level and comparing all other levels to the last level), the expected simulated parameter estimates for the treatment levels are given in Table 2.

Treatment Effect Level	**Parameter Estimate Calculation (LS-means)**	**Parameter Estimate**
1	-2 + (1 – 2) = -3	-3 – (-1) = -2
2	-2 + (2 – 2) = -2	-2 – (-1) = -1
3	-2 + (3 – 2) = -1	-1 – (-1) = 0

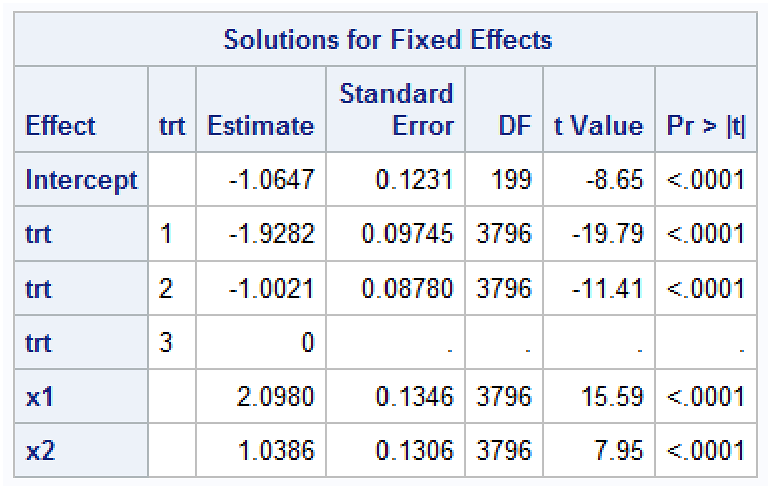
Table 2. LS-means and Parameter Estimates for the Logistic Regression Model

Output 9, Output 10, and Output 11 show that the simulation is working. The estimated results from PROC GLIMMIX are close to the simulated values. If a greater degree of validation is required, then increase both the sample sizes for the GRP effect and the number of observations within each GRP level.



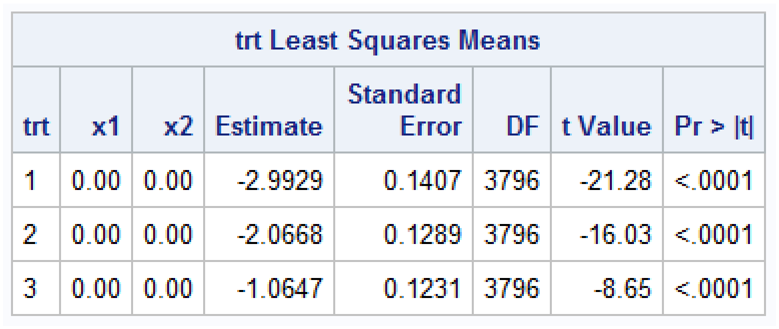
Covariance Parameter Estimates			
Cov Parm	Subject	Estimate	Standard Error
Intercept	grp	0.5457	0.08467

Output 9. Estimated Covariance for the GRP Effect



Solutions for Fixed Effects						
Effect	trt	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept		-1.0647	0.1231	199	-8.65	<.0001
trt	1	-1.9282	0.09745	3796	-19.79	<.0001
trt	2	-1.0021	0.08780	3796	-11.41	<.0001
trt	3	0	.	.	.	.
x1		2.0980	0.1346	3796	15.59	<.0001
x2		1.0386	0.1306	3796	7.95	<.0001

Output 10. Estimated Fixed-Effect Parameters



trt Least Squares Means							
trt	x1	x2	Estimate	Standard Error	DF	t Value	Pr > |t|
1	0.00	0.00	-2.9929	0.1407	3796	-21.28	<.0001
2	0.00	0.00	-2.0668	0.1289	3796	-16.03	<.0001
3	0.00	0.00	-1.0647	0.1231	3796	-8.65	<.0001

Output 11. LS-means for Treatment Levels

A LOGISTIC REGRESSION MODEL WITH TWO RANDOM EFFECTS

The complexity of the simulation code increases when you add a second random effect to the model. Suppose that the model now has two random effects, *A* and *B*, with 30 and 20 levels, respectively. Nested DO loops cannot be used to generate the levels for these effects. The random realizations, or the draw from the associated normal distribution, need to be done outside of the overall loop used for the simulation. If nested DO loops are used, and the loop for *B* is nested in the loop for *A*, then a different realization for the levels of *B* would be generated for each level of *A*. Those realizations correspond to generating a random effect of *B* nested in *A*, not for a main effect of *B*.

Arrays specify the variables to hold the random draw from the normal distributions. In the code below, the variance of the *A* effects is 4 and the variance for *B* is 0.49. When those realizations are created, they are used in the main loop of the simulation as part of the calculation for the linear predictor for the logistic regression model:

```
data test;
   call streaminit(25345278);
   array AEffect{30} AEffect1-AEffect30;
   array BEffect{20} BEffect1-BEffect20;
   do a=1 to 30;
      AEffect{a}=rand('normal',0,2);
   end;
   do b=1 to 20;
      BEffect{b}=rand('normal',0,.7);
   end;
   do i=1 to 25000;
      x1=rand('uniform');
      x2=rand('uniform');
      trt=rand('integer',1,3));
      a=rand('integer',1,30);
      b=rand('integer',1,20);
      logit=-2 + 2*x1 + x2 + (trt-2) + AEffect{a} + BEffect{b};
      p=exp(-logit)/(1+exp(-logit));
      if rand('uniform')>p then y=1; else y=0;
      output;
    end;
    keep trt a b x1 x2 y;
run;
```

```
proc glimmix data=test;
   class a b trt;
   model y=trt x1 x2 / link=logit dist=binomial s;
   random int / subject=a;
   random int / subject=b;
   nloptions tech=nrridg;
run;
```

ZERO-INFLATED POISSON MODELS

Zero-inflated Poisson models are popular in the field of medicine. They arise from data where the response follows a Poisson distribution, with an excess of zeroes in the data. Further, the process that gives rise to these extra zeroes is deemed to be independent of the Poisson process that defines the rest of the data.

For example, a response variable *Y* follows a Poisson distribution with a mean parameter that depends on two covariates, X1 and X2. There are extra zero values in the response that researchers view as a valid issue with the data. That spike is also assumed to be a function of the two covariates.

Data for this model can be created with the following simulation. The covariates X1 and X2 are generated from the standard normal and uniform distributions. The mean of the Poisson is generated from the two covariates. The probability of the response coming from the zero spike is generated from a logit transform, and that probability depends only on the second covariate:

```
data zip;
    call streaminit(576134);
    do i=1 to 10000;
       x1=rand("normal");
       x2=rand("uniform",1,3);
       mu=exp(2 + .3*x1 - .2*x2);
       y=rand("poisson",mu);
       pzero=cdf('logistic',1+x2*2);
       if rand("uniform")<pzero then y=0;
       output;
     end;
run;

proc genmod data=zip;
   model y=x1 x2 / dist=zip;
   zeromodel x1 x2;
run;
```

The GENMOD procedure is used to estimate the model. (The COUNTREG procedure in SAS/ETS® software also estimates zero-inflated models.) Output 12 and Output 13 show that the estimated parameter values closely match the simulated values. Note that the estimate for X1 in the zero-inflated piece of the model is close to zero. That result is expected, because the zero-inflated probability in the simulation did not depend on X1.

Analysis Of Maximum Likelihood Parameter Estimates							
Parameter	DF	Estimate	Standard Error	Wald 95% Confidence Limits		Wald Chi-Square	Pr > ChiSq
Intercept	1	2.0677	0.1575	1.7591	2.3764	172.40	<.0001
x1	1	0.2934	0.0374	0.2201	0.3667	61.51	<.0001
x2	1	-0.2786	0.1092	-0.4926	-0.0646	6.51	0.0107
Scale	0	1.0000	0.0000	1.0000	1.0000		

Output 12. Estimated Parameters for the Poisson Model

Analysis Of Maximum Likelihood Zero Inflation Parameter Estimates							
Parameter	DF	Estimate	Standard Error	Wald 95% Confidence Limits		Wald Chi-Square	Pr > ChiSq
Intercept	1	0.7085	0.3363	0.0494	1.3676	4.44	0.0351
x1	1	-0.0398	0.0891	-0.2144	0.1349	0.20	0.6554
x2	1	2.1521	0.2245	1.7119	2.5922	91.85	<.0001

Output 13. Estimated Zero-Inflated Parameters

A NONLINEAR MODEL: A WEIBULL DISTRIBUTION WITH CENSORING

The NLMIXED procedure is a very flexible procedure for fitting nonlinear models that involve random effects. The procedure can also work with any custom log likelihood that can be programmed using SAS DATA step code.

The Weibull distribution is often used to model survival data. One common aspect of survival data is censoring. Suppose that you have a survival time, days, that follows a Weibull distribution. The shape parameter is a constant, but the scale parameter is a function of a covariate X and the random GROUP effect.

Each level of the random GROUP effect has a realization from a standard normal distribution associated with it (*U*, in the following code). The scale parameter, BETA, is a function of the covariate X and the random effect realization *U*.

This model is complicated and requires a lot of replications in the data to get results in the simulation that match the simulated parameters. You should use 5,000 groups with 50 observations for each group to ensure that the simulation results are close to the estimated parameters.

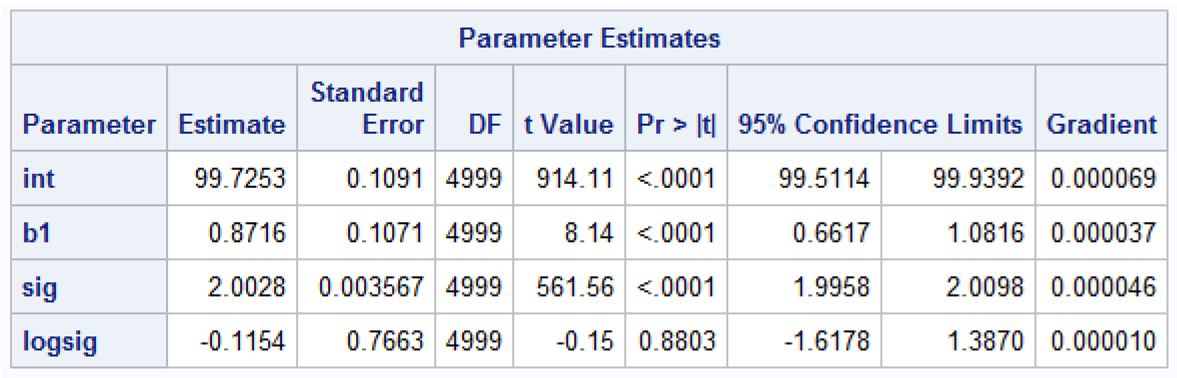
Nonlinear models estimated with PROC NLMIXED are very sensitive to a good choice of starting values. The starting values chosen here are very close to the actual values to improve convergence. The censoring time is generated from a uniform distribution, and if the censoring time is less than the generated survival time, then that observation is considered censored.

There are several distributions built in to PROC NLMIXED, but Weibull is not one of them. The log likelihood for this Weibull distribution is programmed in two pieces: one for observations that are censored and one for observations that are not censored. The following code generates the data and estimates the model:

```
data test;
   call streaminit(7235765);
   do group=1 to 5000;
      u=rand('normal');
      do rep=1 to 50;
         x=rand('normal');
         beta=100 + x + u;
         sig=2;
         days=rand('weibull',sig,beta);
         cens=0;
         days_t=rand('uniform',100,200);
         if days_t<days then do;
            cens=1;
            days=days_t;
         end;
         output;
       end;
    end;
    keep group x days cens;
run;

proc nlmixed data=test gconv=0 fconv=0;
    parms int=110 b1=2 sig=1 logsig=0;
    beta=int + b1*x + u;
    ll=-(days/beta)**sig;
    if cens=0 then ll=ll + log(sig) - sig*log(beta) + (sig-1)*log(days);
    model days ~ general(ll);
    random u ~ normal(0,exp(2*logsig)) subject=group;
    estimate 'sig2' exp(2*logsig);
run;
```

The GCONV= and FCONV= options force PROC NLMIXED to find the best solution available. Output 14 shows the estimated parameters for this model.



Parameter Estimates								
Parameter	Estimate	Standard Error	DF	t Value	Pr > |t|	95% Confidence Limits		Gradient
int	99.7253	0.1091	4999	914.11	<.0001	99.5114	99.9392	0.000069
b1	0.8716	0.1071	4999	8.14	<.0001	0.6617	1.0816	0.000037
sig	2.0028	0.003567	4999	561.56	<.0001	1.9958	2.0098	0.000046
logsig	-0.1154	0.7663	4999	-0.15	0.8803	-1.6178	1.3870	0.000010

Output 14. Estimated Parameters for the Weibull Survival Model

CONCLUSION

This paper has shown the power of both the SAS DATA step and SAS/IML software for simulating data for complex models. Although DATA step code can be easier to understand, SAS/IML code is more efficient and saves time when you create data sets for large simulations. Simulation data sets can get very big, especially when they store multiple replications for a Monte Carlo study. To manage memory issues with large simulations, remove unnecessary temporary variables by using the KEEP statement or the KEEP= data set option. Many of the complex linear models discussed here require large sample sizes to show that the simulation code is generating the data required for the associated model. Adjust the sample size until you are confident that the simulation is programmed correctly.

ACKNOWLEDGMENTS

The authors are indebted to Rick Wicklin for his comments that improved the quality of this paper.

REFERENCES

SAS Institute Inc. 2019a. *SAS® 9.4 Functions and CALL Routines: Reference, Fifth Edition*. Cary, NC: SAS Institute Inc. Available at **go.documentation.sas.com/?docsetId=lefunctionsref&docsetTarget=titlepage.htm&docsetVersion=9.4&locale=en.**

SAS Institute Inc. 2019b. *SAS/STAT 15.1 User's Guide*. Cary, NC: SAS Institute Inc. Available at **go.documentation.sas.com/?docsetId=statug&docsetTarget=titlepage.htm&docsetVersion=15.1&locale=en.**

Stroup, Walter W. 1989. "Use of Mixed Model Procedure to Analyze Spatially Correlated Data: An Example Applied to a Line-Source Sprinkler Irrigation Experiment." In *Applications of Mixed Models in Agriculture and Related Disciplines*, 104–122. Southern Cooperative Series Bulletin No. 343. Baton Rouge: Louisiana Agricultural Experiment Station.

Stroup, Walter W., George A. Milliken, Elizabeth A. Claassen, and Russell D. Wolfinger. 2018. *SAS® for Mixed Models: Introduction and Basic Applications*. Cary, NC: SAS Institute Inc.

Wicklin, Rick. 2013. *Simulating Data with SAS®*. Cary, NC: SAS Institute Inc.

CONTACT INFORMATION

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

Phil Gibbs
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Email: **support@sas.com**
Web: **support.sas.com/en/support-home.html**

Kathleen Kiernan
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Email: **support@sas.com**
Web: **support.sas.com/en/support-home.html**