An Introduction to Simulation Using SAS

This chapter provides an introduction to simulation using the SAS System. Examples are provided of both techniques using DATA steps and standard procedures and techniques based on SAS/IML software. Some examples use entirely simulated data, and some illustrate techniques useful with real study data. This is definitely not a comprehensive survey, just an introduction to some concepts and techniques.

Definitions

Simulation is "the process of using a mathematical model that mimics a real-world situation to conduct experiments in order to describe, explain, illustrate, and predict the behavior of that situation. It includes the process of simulating data based on an assumed model." (Munsaka and Carniello, p. 2)

In the pure statistics world, simulation can be used to investigate the properties of estimators and hypothesis tests. In an applied clinical trials world, simulation can be used to simulate expected data, model enrollment, assess sensitivity to missing data, estimate needed sample sizes, investigate the probability of study results, and so forth. SAS can be used for any of these purposes.

Brief definitions of complex simulation terms that you might run into include

Bootstrap: Produce a sample, re-sample it many times (simple random sampling with replacement), analyze each iteration, and base a population estimate (of standard error, confidence interval, etc.) on the compiled sample estimates.

Jackknife: From an original sample, produce re-samples not by random sampling but by systematically dropping subsets of the original data. Again, analyze each iteration and base a population estimate on the compiled sample estimates. Typically used to estimate bias.

Randomization or perturbation test: Involves using real data but exchanging the labels on the data points multiple times, analyzing each shuffled group, and analyzing the compiled results. The point is to see how extreme the results from the actual data arrangement are when compared with the results from many alternative arrangements of the data.

Most of what we will do in this course is simpler types of simulation.

Topics covered in these notes, mostly via examples:

1. Simulating univariate data
    a. Discrete
    b. Continuous
    c. Simulating a coin toss (extended univariate example)
2. Simulating realistic data for a hypothetical study/clinical trial
3. Simulating realistic data based on certain inter-variable correlations
4. Simulating data to estimate sampling distributions (Monte Carlo, non-bootstrap)
5. Bootstrap-based simulation for a statistical study
6. A simple way to do a randomization or perturbation test
7. Some general simulation principles and strategies

Note that I have attended a lot of Masters presentations over the years where students have generated / simulated data with known properties in order to evaluate the quality of estimators.

1 Simulating univariate data

Simulating data means generating a random sample from a distribution with known properties. A general form for simulating univariate data in a DATA step is the following:

```
* general form for simulating univariate data in a DATA step;
%LET N = 100;                  * size of sample;
DATA mysample(KEEP=x);         * the x's will be the generated values;
      CALL STREAMINIT(67787); * initialize random number stream;
      p = 1/2;                 * set distrib-specific parameters before loop;
      DO i = 1 TO &N;
            x = RAND("Bernoulli", p); * call RAND for desired distribution;
            OUTPUT;
      END;
RUN;

* check generated data in some reasonable way;
title 'Continuous check';
proc means data=mysample;
    var x;
run;
proc univariate data=mysample noprint;
    histogram x;
run;

title 'Discrete check';
proc freq data=mysample;
    tables x;
run;
proc sgplot data=mysample;
    vbar x;
run;
```

This form works for both continuous and discrete data types, and of course you can generate and output data from more than one distribution in a single DO loop. Look up the RAND function for the distributions available for sampling and their associated parameters.
https://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a001466748.htm

Note that these notes use the RAND function in place of older functions such as UNIFORM (or RANUNI) and NORMAL (or RANNOR). Those earlier functions still work, but the RAND function uses a newer random number generator that has better statistical properties. For most applications, either the old or new functions will work fine.

1a Simulating discrete univariate data

In the code shown above, sampling from the BERNOULLI distribution produced x's that were either 0 or 1, each with a probability of ½ given our value of parameter p. Other discrete distributions that can be sampled include BINOMIAL (specify probability of success and number of trials, returns number of successes), GEOMETRIC (specify probability, returns number of trials until a success), POISSON, and discrete UNIFORM.

```
%LET N = 100;                  * size of sample;
DATA binomial(KEEP=x);         * the x's will be the generated values;
      CALL STREAMINIT(3021);   * initialize random number stream;
      p = 1/2;                 * set distrib-specific parameters before loop;
      t = 20;
      DO i = 1 TO &N;
            x = RAND("BINOMIAL", p, t);
            OUTPUT;
      END;
RUN;

title 'Check binomial sampling (number of successes) for reasonableness';
proc freq data=binomial;
      tables x;
run;
title;
```

```
%LET N = 100;                  * size of sample;
DATA die6(KEEP=x);             * the x's will be the generated values;
      CALL STREAMINIT(55631);  * initialize random number stream;
      k=6;                     * set distrib-specific parameters before loop;
      DO i = 1 TO &N;
            x = CEIL(k * RAND('UNIFORM'));
            OUTPUT;
      END;
RUN;

title 'Check rolls of a fair six-sided die';
proc freq data=die6;
      tables x;
run;
title;
```

Another very useful discrete distribution is TABLE, where SAS chooses from a table of values, each with a probability that you specify. The returned value is an integer between 1 and the number of probabilities. A nice application is to use these returned values as pointers to an array, the values of which become the values of a variable (see the section below where data is generated for a hypothetical clinical trial).

```
%LET N = 100;
DATA table(KEEP=x);
    CALL STREAMINIT(67787);
    DO i = 1 TO &N;
        x = RAND("TABLE", 0.1,0.7,0.2);  * specified probs must sum to 1;
        OUTPUT;
    END;
RUN;

title 'Check sampling from a specified table';
proc freq data=table;
    tables x / nocum;
run;
title;
```

1b Simulating continuous univariate data

Continuous distributions that can be sampled include NORMAL, UNIFORM, EXPONENTIAL, WEIBULL, BETA, and GAMMA (and many others).

First let's sample from the NORMAL distribution, here with mean 2 and standard deviation 1.

```
%LET N = 100;                  * size of sample;
DATA normal(KEEP=x);           * the x's will be the generated values;
      CALL STREAMINIT(55631); * initialize random number stream;
      mu=2;                    * set distrib-specific parameters before loop;
      sigma=1;
      DO i = 1 TO &N;
            x = RAND('NORMAL',mu,sigma);
            OUTPUT;
      END;
RUN;

title 'Check normal sample (mean of 2, std of 1)';
proc means data=normal;
      var x;
run;
proc univariate data=normal noprint;
      histogram x;
run;
title;
```

By default the UNIFORM distribution generates values between 0 and 1. To generate values uniformly distributed between certain values a and b, you can use code such as the following:

```
%LET N = 100;                        * size of sample;
DATA uniform(KEEP=x);                * the x's will be the generated values;
      CALL STREAMINIT(55631);        * initialize random number stream;
      a=5;
      b=10;
      DO i = 1 TO &N;
            x = a + ((b-a)*RAND('UNIFORM'));
            OUTPUT;
      END;
RUN;

title 'Check uniform distribution between 5 and 10';
proc means data=uniform;
      var x;
run;
title;
```

1c Using PROC IML to simulate both continuous and discrete univariate data

You can also use PROC IML to simulate univariate data. The general form is as follows:

```
* general form for simulating univariate data with SAS/IML;
%LET N=100;                    * sample size;
PROC IML;
      CALL RANDSEED(89111);    * use any seed here;
      x=j(1,&n);               * allocate row vector of desired size;
      CALL RANDGEN(x, 'DistributionName', param1, param2, ...);
```

Distribution name possibilities and parameters are the same as for the DATA step's RAND function. For example, the code below would generate 50 values from a BINOMIAL distribution with probability of success=1/2 and 10 attempts as well as 50 values from a NORMAL distribution with mu 0 and standard deviation 2.

```
%LET N=50;
PROC IML;
      CALL RANDSEED(28712);    * use any seed here;
      bin=j(1,&n);             * allocate row vectors of desired size;
      norm=j(1,&n);
      p=1/2; t=10; mu=0; std=2;      * set parameters;

      CALL RANDGEN(bin,'BINOMIAL',p,t);   * generate and print values;
      CALL RANDGEN(norm,'NORMAL',mu,std);
      PRINT bin;
      PRINT norm;

      * check sample from normal distrib with mean 0, standard deviation 2;
      *RESET PRINT;
      RESET NOPRINT;
      mean_norm = norm[,:];
      PRINT 'NORM mean' mean_norm;

      norm_cent = norm-mean_norm;
      norm_ss = norm_cent[,##];
      std_norm=SQRT(norm_ss/(NCOL(norm)-1));
      PRINT 'NORM standard deviation' std_norm;

      * check sample from binomial distribution (p=1/2, 10 tries);
      RESET PRINT;
      cats=UNIQUE(bin);              * makes row vector of bin's unique vals;
      cats_c=CHAR(cats,2);           * converts these values to character;
                                     * so can be used as row headings;
      counts=J(NCOL(cats),1,0);      * make row vector of 0's of size needed;
      RESET NOPRINT;
      DO i=1 TO NCOL(cats);          * operate loop for each unique bin value;
            idx=LOC(bin=cats[i]);    * LOC returns all indices;
                                     * satisfying the condition;
            counts[i]=NCOL(idx);     * count number of cols returned by LOC;
      END;

      PRINT counts[ROWNAME=cats_c];
```

Concerning seed values for random number generation: CALL STREAMINIT (in DATA step) and CALL RANDSEED (in IML) are used to set the seed value for subsequent RAND or RANDGEN function calls. The seed value controls the sequence of random numbers generated, and the same sequence will be produced every time that seed is used. This is GOOD for reproducibility of results. These calls need to be made only once per step; subsequent calls are ignored by SAS.

1d Simulating a coin toss (extended univariate example)

Coin tossing provides a neat simple simulation example.

```
* simulating a coin toss - relative freq of heads as # of tosses increases;
* with thanks to Bailer, Statistical Programming in SAS (p. 241);
DATA cointoss;
      RETAIN num_heads 0;
      CALL STREAMINIT(440289);
      DO itoss=1 TO 1000;
            outcome=RAND('BERNOULLI',0.50);
            num_heads=num_heads + (outcome=1);
            probability_heads=num_heads/itoss;
            OUTPUT;
      END;
RUN;
PROC SGPLOT DATA=cointoss;
      SERIES X=itoss Y=probability_heads / LINEATTRS=(THICKNESS=2);
      YAXIS LABEL="Estimated Pr(HEADS)" VALUES=(0.25 TO 0.60 BY 0.05);
      XAXIS LABEL="Number of simulated data points" VALUES=(0 TO 1000 BY 50);
      REFLINE 0.50 / AXIS=Y;
RUN;
```

Here is an equivalent example done with PROC IML. If you use the same seed, the plots are exactly the same. If you change the initial seed, the trajectory is different but in the end there is still convergence toward 0.50. (Note: the video shows an inefficient version of this code.)

```
/* make sure code works for 10 coin flips */
%let n=10;
proc iml;
    call randseed(440289);

    * flip all coins and generate row of outcomes;
    outcomes = j(1,&n);  /* j(&n,1) for column */
    call randgen(outcomes,'BERNOULLI',0.50);
    print outcomes;

    * produce cumulative row;
    cum = j(1,&n);
    do i=1 to &n;
        if i=1 then cum[1,i] = outcomes[1,i] ;
        else cum[i] = cum[1,i-1] + outcomes[1,i] ;
    end;
    print cum;

    * compute proportion heads;
    prop = j(1,&n);
    do i=1 to &n;
        prop[i] = cum[i] / i;
    end;
    print prop;
```

```
    * concatenate all, transposing into columns and preceeding with a column
counter;
    all = (1:&n)` || outcomes` || cum` || prop`;
    print all;
quit;


/* now run for all 1000 tosses, output to SAS data set, and plot */
%let n=1000;
proc iml;
    call randseed(440289);

    * flip all coins and generate row of outcomes;
    outcomes = j(1,&n);  /* j(&n,1) for column */
    call randgen(outcomes,'BERNOULLI',0.50);
    *print outcomes;

    * produce cumulative row;
    cum = j(1,&n);
    do i=1 to &n;
        if i=1 then cum[1,i] = outcomes[1,i] ;
        else cum[i] = cum[1,i-1] + outcomes[1,i] ;
    end;
    *print cum;

    * compute proportion heads;
    prop = j(1,&n);
    do i=1 to &n;
        prop[i] = cum[i] / i;
    end;
    *print prop;

    * concatenate all, transposing into columns and preceeding with a column
counter;
    all = (1:&n)` || outcomes` || cum` || prop`;
    * print all;

    create PropHeadsIML var {is outcomes numheadscum probability_heads};
    append from all;
    close PropHeadsIML;
quit;

title 'Coin toss simulation with PROC IML';
proc sgplot data=PropHeadsIML;
    series x=is y=probability_heads / lineattrs=(thickness=2);
    yaxis label="Estimated Pr(HEADS)" values=(0.25 to 0.60 by 0.05);
    xaxis label="Number of simulated data points" values=(0 to 1000 by 50);
    refline 0.50 / axis=y;
run;
```

2 Simulating realistic data for a hypothetical study/clinical trial

Imagine being in a situation where a study's time frame is short and you need to be able to produce some important tables very soon after the study's data is ready. In this type of situation, it would be nice to have fake data of the same form as the real data in order to develop your table code ahead of time. Then when the real data is in, you can simply change a libref and rerun your programs, and the reports will be ready.

Data "of the same form" means that all variables in the fake data have the same names, types, labels, and rough magnitudes as the real data will have. You should know this information if the study has been well planned. Let's imagine a study and generate some fake data to use for preliminary table code development.

Study hypothesis: Taking baby aspirin daily in the evening lowers systolic blood pressure in middle-aged African-American men and women. BMI (measured at baseline only) is considered to be a possible confounder.

Study details: 300 participants, evenly divided between men and women between the ages of 45 and 65 (maximum age at baseline is 64). Half are randomized to take baby aspirin and half are randomized to placebo. The study duration for each participant is 9 months, with the baseline visit at month 0 and intermediate visits at months 3 and 6. At each visit, the participant's blood pressure is measured and any adverse events are noted.

```
data baseline (keep=ID Gender Age Height Weight BMI Sys_BP BaseDate Visit
Treatment)
     visits    (keep=ID Date Visit Sys_BP Efficacy)
     aes       (keep=ID Visit AE Severity)
     ;

    call streaminit(396805);

    * temporary arrays to provide values of aes and their severity;
    array aelist  [5] $16 _temporary_
('HEADACHE','ITCHING','RASH','NAUSEA','FLUSH');
    array sevlist [3] $16 _temporary_ ('MILD','MODERATE','SEVERE');
    array efflist [4] $16 _temporary_ ('NONE','POOR','MODERATE','GOOD');

    * first possible date for randomization was July 1, 2013;
    studystartdate='01JUL2013'd;

    * set needed parameters (this degree of detail not needed for this
application!);
    BMImuM=26;
    BMImuF=25.5;
```

```
    BMIsigma=5.0;

    HTmuM=1.78; * height in meters;
    HTmuF=1.63;
    HTsigmaM=.072;
    HTsigmaF=.071;

    SBPmuYM=139;
    SBPsigmaYM=22.5;
    SBPmuOM=148;
    SBPsigmaOM=25.5;

    SBPmuYF=148;
    SBPsigmaYF=30;
    SBPmuOF=156;
    SBPsigmaOF=27.5;

    * loop through each participant;
    do ID=1001 to 1300;

        * assign baseline characteristics by gender and age;

        Age = INT(45 + (65-45)*RAND('UNIFORM'));

        Visit=0;

        Treatment=0;

        if rand('UNIFORM')>.5 then do;

            Gender = 'M';

            BMI = round(RAND('NORMAL',BMImuM,BMIsigma),.01);

            Height = round(RAND('NORMAL',HTmuM,HTsigmaM),.01);

            if 45<=age<=54 then
                Sys_BP=RAND('NORMAL',SBPmuYM,SBPsigmaYM);   *SBP really
skewed right but use NORMAL for now;
            else
                sys_bp=RAND('NORMAL',SBPmuOM,SBPsigmaOM);


            if rand('UNIFORM') < .5 then
                Treatment=1;

        end;

        else do;

            gender = 'F';

            BMI = round(RAND('NORMAL',BMImuF,BMIsigma),.01);

            height = round(RAND('NORMAL',HTmuF,HTsigmaF),.01);
```

```
            if 45<=age<=54 then
                sys_bp=RAND('NORMAL',SBPmuYF,SBPsigmaYF);
            else
                sys_bp=RAND('NORMAL',SBPmuOF,SBPsigmaOF);

            if rand('UNIFORM') < .5 then
                Treatment=1;

        end;

        Weight = ROUND(BMI*(height**2),.1);

        sys_bp = ROUND(sys_bp,.1);

        BaseDate=studystartdate + int(RAND('UNIFORM')*5);
        format basedate mmddyy10.;

        output baseline;

        * for each participant, generate number of visits;

        NVisits = RAND('TABLE',1/8,1/4,5/8);

        * loop through visits for participant;

        format Date mmddyy10.;

        do visit=1 to nvisits;

            Date = basedate + (visit*90);

            sys_bp = sys_bp - int(RAND('UNIFORM')*6);

            Efficacy = efflist[RAND('TABLE',.1,.1,.4,.4)];

            output visits;

            * create a max of 2 AEs for each participant (usually none);
            NAES = RAND('TABLE',.7,.25,.05)-1;

            * loop through aes;
            do i=1 to naes;

                Severity = sevlist[RAND('TABLE',.4,.4,.2)];

                AE = aelist[RAND('TABLE',.1,.3,.4,.05,.15)];

                output aes;

            end;    /* ends do i=1 to naes; */

        end;    /* ends do visit=1 to nvisits; */

    end;    /* ends do ID=1001 to 1300; */

run;
```

```
* check baseline variables;
title 'Check baseline variables';
proc means data=baseline n nmiss mean std min max;
    class gender;
    var age bmi height weight sys_bp;
run;
proc freq data=baseline;
    tables gender*treatment;
run;

title2 'Baseline Date';
proc tabulate data=baseline;
    class visit;
    var basedate;
    tables visit, basedate*(MIN MAX)*f=date10.;
run;

title2 'BMI';
proc sgpanel data=baseline;
    panelby gender;
    histogram bmi;
run;
title2 'Height';
proc sgpanel data=baseline;
    panelby gender;
    histogram height;
run;
title2 'Systolic Blood Pressure';
proc sgpanel data=baseline;
    panelby gender;
    histogram sys_bp;
run;

proc format;
    value agecat 45-54='Y' 55-64='O';
run;
proc means data=baseline n nmiss mean std min max;
    class gender age;
    var sys_bp;
    format age agecat.;
run;


* check aes;
title 'Check AE variables';
proc freq data=aes;
    tables visit*ae*severity / list missing;
run;


* check visit variables;
title 'Check visit variables';
proc freq data=visits;
    tables visit visit*efficacy;
run;
```

```
proc means data=visits n nmiss mean std min max;
    class visit;
    var sys_bp;
run;
proc means data=visits n nmiss mean std min max;
    class efficacy;
    var sys_bp;
run;

title2 'Visit Dates';
proc tabulate data=visits;
    class visit;
    var date;
    tables visit, date*(MIN MAX)*f=date10.;
run;
```

3 Simulating realistic data based on certain inter-variable correlations

The technique illustrated above is fine for certain applications. You could even build in relationships among variables using IF/THEN logic and varying parameters. For building in more sophisticated relationships, you might want to base your simulation on a covariance matrix. In fact, if you want to simulate data that has a covariance structure similar to existing data, you are in luck: SAS has built-in facilities that make this very simple. IML's RandNormal function and SAS/STAT's PROC SIMNORMAL are illustrated below.

This code shows simulating data that has a covariance structure like the one among variables calories, totalfat, and cholesterol in the candy data set from BIOS511. You can use PROC CORR to generate means and a covariance matrix to feed into either the RandNormal function or PROC SIMNORMAL. Those two tools do the simulation work, and in either case you can check the output data set using PROC CORR to make sure it has the intended characteristics.

```
ods graphics on;
title 'Starting data set';
proc corr data=bios511.candy cov outp=outcov plots=matrix(histogram);
    where servings=1;
    var calories carbohydrate sugars;
run;

* use IML to generate data with this same covariance structure;
%let n=1000;
proc iml;
```

```
    use outcov;
    read all var {calories carbohydrate sugars} where(_type_='MEAN') into
means;
    read all var {calories carbohydrate sugars} where(_type_='COV')  into
covmat;
    print means covmat;

    call randseed(5883);
    x = RandNormal(&n, means, covmat);

    SampleMean=mean(x);
    SampleCov=cov(x);

    c={"calories" "carbohydrate" "sugars"};
    print (x[1:5,]) [label="First 5 Obs: MV Normal"][colname=c];
    print SampleMean[colname=c];
    print SampleCov[colname=c rowname=c];

    create MVN from x[colname=c];
    append from x;
    close MVN;
quit;

title 'Data simulated with PROC IML (RandNormal function)';
proc corr data=MVN cov plots(maxpoints=NONE)=matrix(histogram);
    var calories carbohydrate sugars;
run;

* use PROC SIMNORMAL to do the same;
proc simnormal data=outcov out=simnorm seed=5883 numreal=1000;
    var calories carbohydrate sugars;
run;

title 'Data simulated with PROC SIMNORMAL';
proc corr data=simnorm cov plots(maxpoints=NONE)=matrix(histogram);
    var calories carbohydrate sugars;
run;
```

Note: If you need to generate multivariate data that mirror sample characteristics but the variables can't be considered multivariate normal, I recommend consulting chapter 4 of the book SAS for Monte Carlo Studies: A Guide for Quantitative Researchers.

4 Simulating data to estimate sampling distributions (Monte Carlo, non-bootstrap)

The information above shows how to simulate a random sample of N observations from a variety of distributions. This section shows how to simulate many random samples and use them to estimate the distribution of sample statistics for the source population of the samples. Such statistics could include p-values, standard errors, and confidence intervals. These types of simulations are often called *Monte Carlo* studies.

The general plan for conducting a Monte Carlo simulation in SAS is the following:

1. Simulate data with known properties: *m* samples of *N* observations apiece. Each sample must have an identifier.
2. Run an appropriate procedure by the sample identifier to compute statistics for each sample, saving the results in an output data set.
3. Use the output data set to compute summary statistics and produce graphs to answer questions about hypothesis tests, confidence intervals, and so forth.

The plan above could be considered pseudo-code as a starting point for writing a program to do such a simulation. If you are writing a complex program, it can be very helpful to write out a plan for your program as a series of comments. Then you fill in the sections with appropriate code. Test each section as you go along. This fulfills three very important coding principles: (1) Think before you code. (2) Document your code well. (3) Develop complex code in pieces and test each piece as you go along.

Here's code for a Monte Carlo simulation of the sampling distribution of the mean from the UNIFORM distribution, where 10 is the size of each sample. This code can be used as a template for more complex simulations following the same pattern.

```
/* 0. Specify sample sizes (using macro variables is a good practice) */
%LET N=10;          /* N=size of each sample */
%LET m=1000;        /* m=number of samples */

/* 1. Simulate data with the DATA step */
DATA SimUni;
      CALL STREAMINIT(172635);

      DO SampleID=1 TO &m;
            DO i=1 TO &N;
                  x=RAND('UNIFORM');  /* change this for your application */
                  OUTPUT;
            END;
      END;
RUN;
```

```
/* 2. Compute statistics for each sample */
PROC MEANS DATA=SimUni NOPRINT;
      BY SampleID;
      VAR x;
      OUTPUT OUT=OutStatsUni MEAN=SampleMean;
RUN;

/* 3. Analyze the samples -> approx sampling distribution of statistic */
TITLE 'Sample distribution of Mean generated from UNIFORM distribution';
PROC MEANS DATA=OutStatsUni N MEAN STD P5 MEDIAN P95;
      VAR SampleMean;
RUN;
PROC UNIVARIATE DATA=OutStatsUni NOPRINT;
      HISTOGRAM SampleMean / NORMAL;
RUN;
TITLE;

/* Tip:  if you don't know a distribution a priori, you can */
/*       request a non-parametric look at the distribution  */
/*       using a KERNEL smooth                              */
/* In this case, the distribution looks very normal.        */
PROC SGPLOT DATA=OutStatsUni;
      HISTOGRAM SampleMean;
      DENSITY SampleMean / TYPE=KERNEL LEGENDLABEL="Mean";
RUN;
```

Once we have such a distribution, we can use it to answer questions about individual samples. For example, say you drew a new sample of 10 points from the UNIFORM distribution and wondered about the probability that the mean of this sample would be less than 0.4.

```
/* What is the probability that the mean of a new sample of 10 points */
/* from the UNIFORM distribution will have a mean < 0.4?              */
DATA TestNewSamp;
      SET OutStatsUni;
      SmallishMean = (SampleMean < 0.4);
RUN;

TITLE 'Probability of Mean < 0.4';
PROC FREQ DATA=TestNewSamp;
      TABLES SmallishMean / NOCUM;
RUN;
TITLE;
```

You find that the probability of a sample mean less than 0.4 is about 15% (that is, not unlikely).

If you took BIOS 511 in fall 2013, take a look at section s of Chapter 10. There, Jackie provided code for a simulation of TYPE I error. You'll see that this code follows the template for Monte Carlo simulation described above.

You can also use IML to perform this type of simulation, also following a pattern.

```
/* Use IML to find the sampling distribution of the mean */
%LET N=10;         /* N=size of each sample */
%LET m=1000;       /* m=number of samples */
PROC IML;
      CALL RANDSEED(172635);
      x = J(&m,&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. Analyze the distribution       */
      StdDev = STD(s);
      CALL QNTL(q,s,{0.05 0.95});
      PRINT Mean StdDev (q`) [COLNAME={"5th Pctl" "95th Pctl"}];

      /* compute proportion of statistics less than 0.4 */
      /* (works because (s < 0.4) is either 0 or 1)     */
      Prob = MEAN(s < 0.4);
      PRINT Prob[FORMAT=PERCENT7.2];
```

Yea! Using the same seed, IML yields the same results as our DATA step-based simulation.

Note: If you review old material about doing simulations with SAS, you might see macro code in which each call of the macro generates one sample, finds statistics for the sample, and appends those statistics to a data set. After all the macro calls, the accumulated data is analyzed as in step #3 above. With that older simulation technique, steps #1 and #2 are MUCH LESS efficient than doing things with BY processing as illustrated in these notes. Instead of one DATA step and one PROC call no matter what *m* is, as shown in these notes, you would have *m* DATA steps and *m* procedure calls! Yikes!

4a Graphical extension of a Monte Carlo simulation

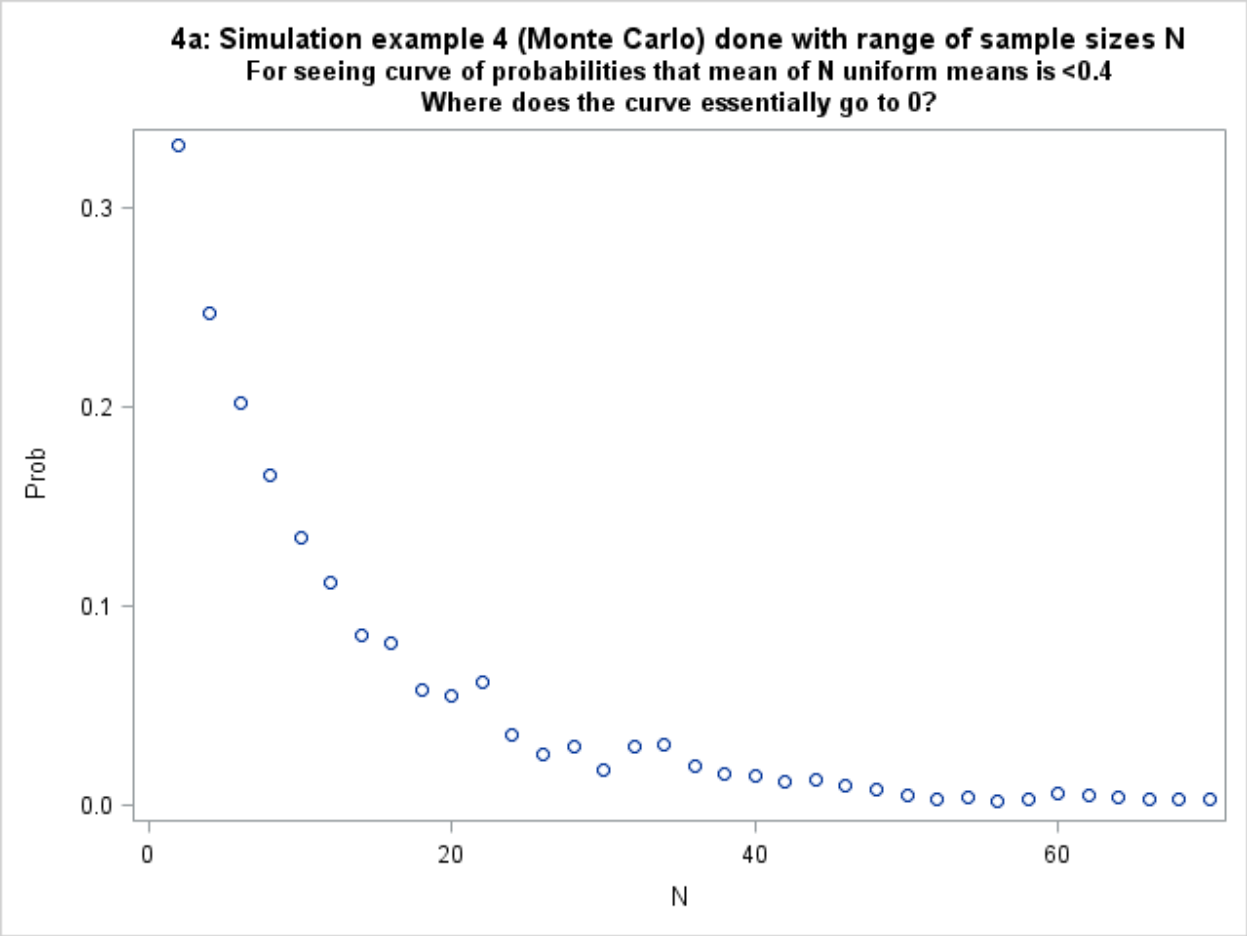
What if you were using Monte Carlo simulation to decide on the "best" N, or at least an adequate N, for your particular application? You might run the simulation with a series of different N's, graph the results, and use the graph to see where the estimate stabilizes. The optimal N would be near that stabilization point.

As an example, let's run the IML code from Example 4 with N ranging from 2 to 70 by 2, each time with m=1000, and plot the resulting Prob values. As you can imagine, once N gets to be a certain size there will be almost no chance that the mean of 1000 samples of that size from the uniform distribution will be less than 0.4, so our curve should approach 0 at some point. What will N be at that point?

Note that this example uses IML rather than the data step implementation of Example 4 since IML allows us to easily concatenate the iteration results in the same step.

```
%let m=1000;
PROC IML;
    CALL RANDSEED(172635);

    do N=2 to 70 by 2;
        x = J(&m,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. Analyze the distribution       */
        StdDev = STD(s);
        CALL QNTL(q,s,{0.05 0.95});
        PRINT Mean StdDev (q`) [COLNAME={"5th Pctl" "95th Pctl"}];

        /* compute proportion of statistics less than 0.4 */
        /* (works because (s < 0.4) is either 0 or 1)     */
        Prob = MEAN(s < 0.4);
        PRINT Prob[FORMAT=PERCENT7.2];

        both = N || Prob;   * for our graph, need to save both N and Prob;
        all = all // both;  * stack results;
    end;

    c = {"N" "Prob"};
    create forP from all[colname=c]; * write results to a data set;
    append from all;
    close forP;

quit;
```

```
title1 '4a: Simulation example 4 (Monte Carlo) done with range of sample sizes N';
title2 'For seeing curve of probabilities that mean of N uniform means is <0.4';
title3 'Where does the curve essentially go to 0?';
proc sgplot data=forP;
    scatter x=N y=Prob;
run;
```



The curve seems to start sitting very close to 0 when N is somewhere between 50 and 60.

5 Bootstrap-based simulation for a statistical study

The section above describes simulation done by drawing random samples from a *model* of a population. In contrast, bootstrap samples are created by sampling with replacement *directly from the data*, sometimes called resampling. Besides this difference, the simulation techniques are similar. The purpose of bootstrapping is typically to approximate the sampling distribution of a statistic when no parametric statistical model of the data is known.

Compared with the template presented in the section directly above, bootstrapping differs mainly in step 1, where instead of generating data with a DATA step, we sample from existing data with PROC SURVEYSELECT.

Here's an example of using bootstrapping to estimate skewness and kurtosis for the population of bream in a lake in Finland, using data set sashelp.fish distributed by SAS in SAS 9.3 and later releases.

```
* make and check out bream data set;
DATA bream(KEEP=Weight);
      SET sashelp.fish(WHERE=(lowcase(species)="bream" AND
^MISSING(weight)));
run;
PROC MEANS DATA=bream NOLABELS N MEAN STD SKEWNESS KURTOSIS;
      VAR weight;
RUN;
PROC UNIVARIATE NOPRINT DATA=bream;
      HISTOGRAM weight;
RUN;

/* 0. Specify sample sizes (using macro variables is a good practice) */
%LET m=5000;      /* m=number of samples */

/* 1. Resample with PROC SURVEYSELECT */
PROC SURVEYSELECT DATA=bream NOPRINT SEED=2162
      OUT=BootSS(RENAME=(Replicate=SampleID))
      METHOD=URS SAMPRATE=1
      REPS=&m OUTHITS;
RUN;

/* 2. Compute statistics for each sample */
PROC MEANS DATA=BootSS NOPRINT;
      BY SampleID;
      VAR Weight;
      OUTPUT OUT=OutStats SKEW=Skewness KURT=Kurtosis;
RUN;

/* 3. Analyze the samples */
TITLE 'Descriptive Statistics for Bootstrap Distribution';
PROC MEANS DATA=OutStats N MEAN STD P5 MEDIAN P95;
```

```
      VAR Skewness Kurtosis;
RUN;
PROC UNIVARIATE DATA=OutStats NOPRINT;
      HISTOGRAM Skewness;
      HISTOGRAM Kurtosis;
RUN;
TITLE;
```

Let's look more closely at the step #1 code:

```
PROC SURVEYSELECT DATA=bream NOPRINT SEED=2162
      OUT=BootSS(RENAME=(Replicate=SampleID))
      METHOD=URS SAMPRATE=1
      REPS=&m OUTHITS;
RUN;
```

- The RENAME is simply for consistency with the section above. *Replicate* is the default name for the replicate identifier to be used as a BY variable.
- METHOD=URS requests unrestricted random sampling, which means sampling with replacement and with equal probability.
- SAMPRATE=1 requests that each resample be the same size as the original data set. Because of this option, the &n sample size parameter used in the previous section is not needed here.
- REPS=&m specifies how many resamples to generate.
- Because we are sampling with replacement, any particular resample is likely to contain multiple copies of some observations. The OUTHITS option specifies that all multiples are to be included in the output data set. If you are operating under restricted conditions and would like to produce a smaller output data set, you might want to omit this option so that only one copy of duplicates is included for any particular replicate. This is safe because the NumberHits variable automatically included in the output data set tells how many times the observation was selected, and this variable can be used as a FREQ variable in many procedures. Here's how the code above would change if you omit OUTHITS:

```
/* 1. Resample with PROC SURVEYSELECT */
PROC SURVEYSELECT DATA=bream NOPRINT SEED=2162
      OUT=BootSS(RENAME=(Replicate=SampleID))
      METHOD=URS SAMPRATE=1
      REPS=&m;
RUN;

/* 2. Compute statistics for each sample */
PROC MEANS DATA=BootSS NOPRINT;
      BY SampleID;
      VAR Weight;
      FREQ NumberHits;
      OUTPUT OUT=OutStats SKEW=Skewness KURT=Kurtosis;
```

```
RUN;
```

The size of data set BootSS goes down from 170000 observation (5000x34) to 108359 when duplicates are eliminated, with this particular seed, but the OutStats data sets are identical.

The note at the end of section 4 also applies to bootstrapping: avoid the inefficient technique of using a macro for looping instead of making one long data set of all the resampled data and then taking advantage of BY processing.

6 A simple way to do a randomization or perturbation test

Sometimes we are interested in the likelihood of the results we actually get when conducting a study, compared with all possible results. This question might come up especially when the distribution of our data is unusual and underlying model assumptions are not met. We can use a kind of resampling called a randomization test or perturbation test to assess the likelihood of our actual results, given all possible outcomes.

For an illustration of a perturbation test we will use the preemies data set from BIOS 511 and look at how gestational age (variable ga) is related to sex. In the sex variable in this data set, 0 signals a boy and 1 signals a girl. Some fundamental displays of these variables and our basic model can be produced with this code:

```
data preem;
      set bios511.preemies(keep=sex ga);
run;

proc univariate noprint data=preem;
      histogram ga;
run;

proc sgplot data=preem;
      vbox ga / category=sex;
run;

proc glm data=preem;
      class sex;
      model ga=sex;
run; quit;
```

This model produces on overall p-value of 0.28, indicating that gestational age among these infants is not related to sex [note: this really makes sense, given that the main point of the study behind this data set was to look at how birth weight is related to gestational age and sex]. Still, we can use this model to show how to conduct a perturbation test.

To do this we are going to use a macro provided by David Cassell, as presented in his paper A Randomization-test Wrapper for SAS PROCs. The clever idea coded in this pair of macros is that we don't need to perturb all of the independent variables, which is hard to do when you have more than one. Rather, we can simply permute the values of the dependent variable and associate those shuffled values with multiple copies of the existing rows of independent variable values.

Cassell's wrapper consists of two macros, %rand_gen to be called before we run our model and %rand_anl to be called after. As you'll see, the model to be run here is slightly different from the one shown above. Here are the two macros.

```
%macro rand_gen(
    indata=_last_,
    outdata=outrand,
    depvar=Y,
    numreps=1000,
    seed=0);

/* get size of input data set into macro variable &numrecs */
proc sql noprint;
    select count(*) into :numrecs from &indata;
quit;

/* generate &numreps random numbers for each record, so
   records can be randomly sorted within each replicate */
data __temp_1;
    set &indata;
    call streaminit(&seed);
    do replicate=1 to &numreps;
        rand_dep = RAND('UNIFORM');
        output;
    end;
run;

proc sort data=__temp_1;
    by replicate rand_dep;
run;

/* Now append the new re-orderings to the original data set.
   Label the original as Replicate=0, so the &RAND_ANL macro
   will be able to pick out the correct p-value.  Then use
   the ordering of __counter within each replicate to
   write the original values of &depvar, thus creating a
   randomization of the dependent variables in every
   replicate. */
data &outdata;
    array deplist{ &numrecs } _temporary_;
    set &indata(in=in_orig)
        __temp_1(drop=rand_dep);

    if in_orig then do;
        replicate=0;
        deplist(_n_)=&depvar;
    end;
    else &depvar = deplist{ 1 + mod(_n_,&numrecs) };
run;

%mend rand_gen;

%macro rand_anl(
    randdata=outrand,
    where=,
```

```
    testprob=probf,
    testlabel=F test,);

data _null_;
    retain pvalue numsig numtot 0;
    set &randdata end=endofile;

    %if "&where" ne ""
        %then where &where %str(;);
    if replicate=0 then pvalue=&testprob;
    else do;
        numtot+1;
        numsig + ( &testprob < pvalue );
    end;

    if endofile then do;
        ratio=numsig/numtot;
        put numsig= numtot= ;
        put "Randomization test for &testlabel"
        %if "&where" ne "" %then " where &where";
          " has significance level of "
          ratio 6.4;
    end;
run;

%mend rand_anl;
```

And here is how we can call these macros and model for our example:

```
options mprint;
%rand_gen(indata=preem, outdata=outrand, depvar=ga, numreps=100, seed=6644)

ods listing close;
ods html close;
ods output overallanova=glmanova1;
proc glm data=outrand;
    by replicate;
    class sex;
    model ga=sex;
run; quit;
ods output close;
ods listing;
ods html;

%rand_anl(randdata=glmanova1,
          where= %str(source='Model'),
          testprob=probF,
          testlabel=Model F test )
```

To understand what is going on, it helps to run the %rand_gen macro call and then look at the intermediate and output data sets: intermediate data set work.__temp_1 and output data set work.outrand (specified with the outdata parameter). You'll see that outrand consists of 101 copies of the original observations (original + 100 replications),

but with sex and ga matched up in a different order from the original. As a partial check to see if these reshuffles are in good shape, you could run code like this:

```
proc freq data=outrand;
    tables replicate*sex / list missing;
run;
proc means data=outrand;
    class replicate;
    var ga;
run;
```

Note that the GLM call uses data set outrand (produced by the rand_gen macro) and is done by replicate. Also, it includes an ODS OUTPUT statement to write overall model test information to an output data set (I had to use ODS TRACE to determine the correct data set to save). Otherwise, this model is identical to our original GLM run on the preem data set. We turn off both the listing and HTML destinations to avoid clogging up our SAS session since all of the results we care about are in the glmanova1 output data set produce with ODS OUTPUT. Recall that we can't use NOPRINT in the PROC GLM call because this makes ODS non-functional.

Finally, we call the rand_anl macro to process the output data set glmanova1 and write a note about findings to the SAS log. To write a rand_anl call that would work correctly, I had to examine the OverallAnova output table from GLM: which records contained the p-values of interest (the ones with Model='Source') and which variable contained those p-values (the one named ProbF). If you look at the rand_anl macro, you'll see that what the code there does is quite simple: it counts how many p-values are lower than the original one, divides that count by the number of replicates, and presents that ratio as part of an informative message that uses text provided as parameter values in the rand_anl call.

7 Some general simulation principles and strategies

For efficient and effective simulations with SAS (and often otherwise), here are some good guidelines to follow.

1. Plan before you begin – clarify your goal, select a suitable technique, and so forth.
2. Check the distribution of any simulated data with appropriate summary and/or graphical techniques to make sure it is as you intended.
3. When using regular SAS for a simulation, take advantage of BY processing (use in place of macro loops).
4. When performing a simulation with IML, use the J function to allocate a matrix of the appropriate size to store the simulated data before calling RANDGEN. That way one RANDGEN call can generate all needed values. As the simulation runs, existing values will be replaced rather than new matrices having to be created.
5. Suppress all output generation (listing, HTML) during the BY processing step. If you are not using ODS, implement a procedure's NOPRINT option (if available). NOPRINT options and ODS are incompatible, so if you are using ODS to produce an output dataset, use ODS CLOSE statements in place of NOPRINT.
   Programming tip: For total ODS results suppression, consider this type of sequence.

```
%macro ODSOff;
ods graphics off;
ods exclude all;
ods noresults;
%mend;

%macro ODSOn;
ods graphics on;
ods exclude none;
ods results;
%mend;

/* sample usage */
%ODSOff
proc means data=SimNormal;
      by SampleID;
      var x;
      ods output Summary=SummOut;
run;
%ODSOn
```

6. In the course of some simulations, the SAS log can fill up with SAS notes. To prevent this, and if you are sure that the notes are uninformative, you can include in your code *options nonotes;*. After the procedure run is complete, re-enable notes using *options notes;*.
7. Save intermediate results for later analysis, especially if the simulation takes a long time to run.
8. Begin with a small number of iterations while you are debugging your simulation. When you are sure that everything works, do your final run using the full N. The initial small runs might also help you anticipate how long your full run will take.
9. If you keep the results of different runs done with different seeds (which could be interesting and informative to compare), make sure to note the seed used with each run.
10. Document your code!

General references

Rick Wicklin's book Simulating Data with SAS

John Bailer's book Statistical Programming in SAS (includes more complicated simulation examples than the Wicklin book and is about much more than simulation)

David Cassell - Don't Be Loopy: Re-Sampling and Simulation the SAS Way – a great primer about doing all kinds of resampling-based simulations with SAS – bootstrapping, case resampling and resampling residuals, the jackknife, randomization tests (permutation tests), and cross-validation
http://www2.sas.com/proceedings/forum2007/183-2007.pdf

David Cassell – A Randomization-test Wrapper for SAS PROCs
http://www2.sas.com/proceedings/sugi27/p251-27.pdf

Marie Davidian's handout on simulation studies in statistics
http://www4.stat.ncsu.edu/~davidian/st810a/simulation_handout.pdf

Rick Wicklin blog post - The essential guide to bootstrapping in SAS
https://blogs.sas.com/content/iml/2018/12/12/essential-guide-bootstrapping-sas.html

SAS Studio's Combinatorics and Probability Task list includes some fun simulation tasks that you can play with and see the generated code.