Simulations in STATA

Dr. Ani Dasgupta
MMA and BU[1]

1. Why Do Simulations?

When you can run simulations with your software package, econometrics tends to come alive. One can illustrate the Central Limit Theorem, demonstrate the unbiasedness of the least squares estimator, estimate probabilities of events which are too difficult to calculate analytically and this is where it is really important, when it becomes too hard to theoretically demonstrate that one estimator is 'better' than another (in whatever sense you wish to use the term), you can use simulations to make your point.

2. Pseudo-random Numbers

At the heart of simulations is random number generation, or more accurately, pseudo-random number generation. Pseudo-random numbers are "mock" uniform(0,1) random numbers. The prefix 'pseudo' emphasizes the fact that really, computer-generated random numbers are not random at all – they are generated by well-defined deterministic algorithms, although the stream of numbers thus generated appear random. One of the more common methods of generating these pseudo random numbers is the so-called LCG: linear congruential generator. Here is an example of the LCG scheme:

$x_{n+1} = (69069\ x_n + 1234567) \bmod 2^{32}$

where u mod v is the remainder when u is divided by v. Given a starting number, which is called the *seed*, this scheme can recursively generate a stream of numbers. In general, LCG schemes look like

$x_{n+1} = (A * x_n + B) \bmod M$

where A, B and M are certain integers and this can generate pseudo-random numbers between 0 and M. Upon dividing the x's by M, one can obtain uniform (0,1) numbers.

The building blocks for generating any random number obeying any distribution are uniform (0, 1) random numbers. In STATA, the command

gen x = uniform( )

[1] These manuals have been prepared for teaching Ec508, a Masters econometrics course at Boston University Economics department. Feel free to use it for your own purpose, but do not circulate or use for wider dissemination without permission.

or
gen x = runiform( )

will generate a uniform(0, 1) variate called x (notice that there is nothing inside the parens). If you have 100 observations in your datafile, you will get 100 i.i.d. uniform (0, 1) numbers. If you are starting afresh, with no dataset in memory, you should first issue the command:

set obs 100

so that STATA knows how many observations of random numbers you need. Now, once, you have uniform(0, 1) numbers, you can easily generate uniform (a, b) for any given a and b (How?)

One advantage of doing simulations with random numbers on a computer is that I can replicate your results exactly. For this to happen, before you use the *gen x = uniform( )* You have to set the seed by issuing a command like

set seed 1234567

Now, during my analysis, if I set the same seed, I should get exactly the same set of random numbers that you did.

3. Generating Random Numbers Other Than Uniform (Using Built-in Functions)

STATA can also generate other types of random numbers. Discrete distributions from which you can draw includes binomial, hypergeometric, negative binomial and poisson. Continuous distributions you can draw from includes normal, gamma, chi-squared, t and beta distributions. You should be familiar with these distributions from Ec507; if you are not, look up your favorite graduate statistics text or look up the "handbook" posted here: www.stat.rice.edu/~dobelman/textfiles/**Distributions**Handbook.pdf

It is important for the applied data analyst to have some idea of what are the properties of standard distributions and which distribution is used for modeling what kind of phenomena or variables and why.

Next to uniformly distributed random numbers, the most important random numbers are normal random numbers. One way to obtain 100 normal(0,1) numbers, one can issue the command (after the "*set obs 100*" statement) and store it in a variable called z is:

gen z = rnormal()

(An alternative is the simpler command *drawnorm z*)

If you wish to generate not standard normal numbers, but say, normal numbers with mean 5 and variance 9, you will issue the command

gen z = rnormal(5, 3)

Thus, when STATA sees two arguments inside an rnormal(), it knows that the first argument is the mean and the second variance. Actually we did not need this nicety. For, once you have normal (0, 1) numbers, you should have no problem generating normal (μ, σ^2) (How?)

To generate a gamma variate with shape parameter 2 and scale parameter 7, you will use

gen w = rgamma(2, 7)

As you can imagine, the rbeta(a, b) function will generate a beta variate with the parameters a and b etc. etc.

4. Generating Random Numbers without Built-in Functions

There are many distributions of interest to econometricians for which STATA does not have a built-in random number generator function. The F distribution is an example. If you can write down the inverse of the cumulative distribution function (cdf) for such a distribution, you can generate random numbers from that said distribution by making use of the following fact:

Fact: *Suppose F() is a continuous monotonically increasing cdf. Then* $x = F^{-1}(u)$ *has the cdf F if u is uniformly distributed.*

Hence, an alternative command for generating a standard normal variate is:

gen z = invnorm(uniform())

The invnorm() function is the inverse of the cdf of standard normal. Since the F distribution is very frequently used by statisticians/econometricians, STATA has built-in commands for its density function, distribution function, upper tail function (i.e. 1 – cdf()), and inverses of these two latter functions. Check them out. Same goes for t and chi-squared distributions.

Thus, to generate one hundred F-distributed random variables with 5 and 7 degrees of freedom, I can issue the commands:

set obs 100
gen x = invF(5, 7, uniform( ))

How does one generate draws from a multivariate normal distributions where the different variates are correlated? Let us say you want 10 observations on a multinormal 2-vector with means 3 and -2; variances 4 and 1 and covariance -1. Then you issue the following commands.

set obs 10
mat Sigma = [4, -1\-1, 1]
drawnorm x1 x2, means(3, -2) cov(Sigma)

You will now have 10 observations on two variables x_1 and x_2 as desired.

If you know the correlation matrix and the standard deviations, instead of computing the variance-covariance matrix, you can enter the information directly as follows. Let us say, you need 30 observations on x_1, x_2, x_3 where the three variables have means 2, 5 and –7, standard deviations 6, 20 and 8 with the correlation matrix given by

$$\begin{bmatrix} 1 & .4 & -.8 \\ .4 & 1 & 0 \\ -.8 & 0 & 1 \end{bmatrix}$$

Note that the above is the correlation matrix, not the covariance matrix. We can then generate our desired variables issuing the following commands:

set obs 30
mat C = [1, .4, -.8\ .4, 1, 0\ -.8, 0, 1]
drawnorm x1 x2 x3, means(2, 5, -7) sds(6, 20, 8) corr(C)

and you will have 30 observations on a random 3-vector.

In the next handout, we will learn about programming in STATA, and then we will be able to examone the sampling distribution of such things as the least squares estimator.

Exercise: How will you generate 100 observations on a random variable that takes the value 1 with probability .3, the value 2 with probability .2 and the value 3 with probability .5?

5. Running Simulations in STATA

A simulation typically consists of the following steps:

Step 1. Decide on sample size (say n) and replication number (say m).

Step 2. Initialize a matrix of size m x p with zeros. Here p is the number of objects of interest on which you wish to obtain simulation results.

Step 3. Create a loop (typically using “forvalues”) that will run m times and during each iteration will

a. Generate a new sample of size n according to some given DGP.
b. Run the command or econometric procedure
c. During the i-th iteration will store the p objects of interest in the i-th row of the matrix you initialized.
d. Drop the current variables.

Step 4. Use the matrix of the stored info directly or, convert them into variables and then create histograms, produce descriptive statistics, run tests etc.

As an example, I display a do-file that demonstrates the Central Limit Theorem effect:

```
clear
quietly{
set more off
local m = 500 // rep size
local n = 250 // sample size
set matsize `m' // needed if dealing with matrices larger than Stata's default
mat xmean = J(`m', 1, 0) // matrix that will hold objects of interest
tempvar x smean standardized_mean // defining temporary variables
tempname xmean // defining a temporary scalar
set obs `n'
set seed 547698321

//The loop starts. In each iteration we will create a sample of size n
//from a chi-squared distribution with 5 degrees of freedom. We will take
// the sample mean xbar and create n^.5 (xbar - mu) / sigma where mu
//is the mean of the chi-squared distribution and sigma is its standard
//deviation. According to CLT, this object is supposed to behave like a
//standard normal variate for large n.
forvalues i = 1(1)`m' {
set obs `n'
gen x = invchi2(5,uniform())
summarize x
mat xmean[`i',1] = (((`n')^.5) * (r(mean)- 5))/(10^.5)
drop _all //dropping the current sample before exiting loop
}
```

```
svmat xmean, name(smean) //converting a stored object into a variable
rename smean1 standardized_mean //renaming it for labeling purposes
histogram standardized_mean, bin(20)normal //drawing a histogram
}
swilk standardized_mean //performing Shapiro-Wilk test of univariate normality
```

6. The Simulate Command

STATA has a built-in command that makes it fairly straightforward to run simple simulations. Its syntax goes something like…

simulate object(s)_of_interest, reps(# of reps) : myprog

To use the command, you need a program called say, myprog that will generate objects of interest. The program will generate a new dataset and run a procedure which can be a built-in command or an r-class command that you have written. If it is built-in command, objects_of_interest can be anything that is stored in the ereturn or return list. In particular, if it is estimation command, and you do not specify the objects of interest, STATA will run simulations on all estimated coefficients; else it will run simulations on all objects stored in r().

If the procedure is your own procedure or you are interested in something that is not naturally stored after some built-in-command is executed, you need to write an r-class program, which will return your object of interest. The following do-file illustrates this by running a simulation on the t-statistics created on one of the coefficients in some regression.

```
set obs 100
forvalues i = 1/10{
capture drop x`i'
gen x`i' = rnormal()
}

capture program drop myprog
program define myprog, rclass
capture drop y
capture drop ep
capture mat drop V
capture mat drop b
gen ep = 10*rnormal()
gen y = x1 + 2*x2 + 3*x3 + 4*x4 + ep
reg y x1-x4
mat V = e(V)
mat b = e(b)
return scalar myt = b[1,1] / (V[1,1])^.5
```

end

simulate myt = r(myt), reps(10): myprog

Exercise: Answer all the “hows” and “why”s in this tutorial.