Econ 452 Section 9 - STATA

Connor Cole

November 9, 2015

Monte Carlo Simulations in STATA

NOTE: This material on Monte Carlo simulations is meant to elaborate on what was discussed in class and is completely optional.

Last week we saw a few examples of what we call Monte Carlo simulations in statistics to examine the properties of consistency, unbiasedness, finite sample distributions of $\hat{\beta}$ with normal homoskedastic errors, and asymptotic limiting distributions of $\hat{\beta}$ with either normal or non-normal homosekdastic errors. In this section, we will do a simple Monte Carlo example to look at the performance of 95% confidence intervals. Although will not be asked to do any Monte Carlo simulations for this class, they are a very handy tool to have up your sleeve for modeling purposes and a great way of looking at the performance of different estimators.

Monte Carlo simulations work by repeatedly simulating data created under some known process and then applying some estimation procedure and seeing how the estimation procedure works over many different datasets. Then, since we know the 'truth' of the data process as we're the ones who created it, we can stack up our estimates against the truth and compute bias, mean squared error, the distribution of estimators, standard errors, and other statistics that evaluate the performance of the estimator.

In order to set up a Monte Carlo, we first define some process that we want to do within a program. So, we type `program` and then add a program name, and we type `end` several lines down to indicate the 'end' of the program.

Then, within the program, we set up a process that creates data and then run some estimation process on it and save the output. Setting up the data is just a straightforward extension of the tools for generating random variables we used last time. Saving the results from the output are a bit more tricky, and involve using the post-estimation tools for saving scalars that we briefly covered before in another optional section. So, for example, to create a program titled "babymontecarlo" that generates 1000 observations with the data generating process $y = 2x + \epsilon$ where both x and ϵ are standard normal and then estimates the model using a regression and saves the estimated coefficient and standard error on x, I would type:

```
program babymontecarlo
```

```
clear

set obs 1000

generate x=rnormal(0,1)

generate epsilon=rnormal(0,1)

generate y = 2*x+epsilon

regress y x

scalar b1= _b[x]

scalar se1= _se[x]

end
```

Note that `clear` and `set obs 1000` are included at the beginning of the program in order to clear out the previous data and run the simulation again.

After having defined this routine in a program, we are ready to do a Monte Carlo simulation. Generally, before any simulation like this, we would set the seed to be a certain amount so the result could be easily replicated by someone trying to rederive the same results. In order to run this simulation 1000 times and generate 1000 outcomes, we would type the following commands after having defined the program `babymontecarlo`:

```
set seed 12345

simulate b=b1 se=se1, reps(1000):  babymontecarlo
```

Note that the above command saves the scalars we saved in the program as `b` and `se`. We could have chosen any other different name for these objects. This process saves two columns with 1000 observations each titled `b` and `se` that record the output from each simulation. Unless we add another output scalar above, the simulation will not save additional information.

Underlying this process is just a series of loops and random number generation problems. The `simulate` command just offers a faster way of running this process repeatedly.

The **`Preserve`** and **`Restore`** Commands

Sometimes, we want to perform operations on the data that involve dropping observations, averaging across observations, or some other process that would fundamentally alter the dataset. Since STATA does not save changes to data unless we tell STATA to save and replace the data, we could just drop all observations after we run such processes and reopen the dataset. Sometimes,

however, a faster way of doing the same operation would be to save the current dataset using the `preserve` command, perform the operations we want, and the reopen the old dataset by just typing in `restore`. For example, if we wanted to drop observations that take on a certain value of `VariableName1` and then return to the original dataset after having done work on the subsample, we could type:

```
preserve

drop if VariableName1==3

....

restore
```

Then, we return the original dataset in the end as if nothing has happened.