Econ 452 Section 8 - STATA

Connor Cole

October 26, 2015

Random Numbers in STATA

Sometimes you may want to generate draws of numbers from certain statistical distributions, either to create Monte Carlo simulations or create subsamples from your data. Of course, there can be no true 'random' number generator in a computer program, but statistical packages have come up with ways of simulating randomness by using modular arithmetic. In many ways, this apparent limitation of computer programs is in fact a strength, as it allows us to replicate results predictably.

Before generating random numbers, we always set a 'seed,' or a starting place for our draws from a statistical distribution. I always choose 12345, or the code on my luggage. We do this by typing:

```
set seed 12345
```

Then, we can create a new variable that creates a random draw from a particular distribution for each row in our dataset. As previously, STATA uses a particular formatting for these calculations, and we can see all possible distributions we can create random variables from by typing `help statistical functions`. For now, let's just continue with the t distribution. So, if we wanted to create a new random variable with draws from a t distribution with x degrees of freedom, we type:

```
gen randomt = rt(x)
```

Note that if we reset the seed and created another set of draws from this distribution, this new set of draws would be exactly the same as the first.

Looping Commands in STATA

Sometimes, we want to run commands in STATA that loop over certain values of variables. The `bysort` command we discussed earlier is one way of accomplishing this goal. Sometimes, however, the `bysort` command is cumbersome, and there are additional commands that we want to run for

each subgroup that we'd like to add in. There are, in general, two types of commands we can use, the `forvalues` command and the `foreach` command.

The `forvalues` command

STATA's equivalent of a for loop is the `forvalues` command. This command is useful when we want to perform some interaction that involves moving over different interger values in some sequence. To use this command, we type `forvalues` and then we type some statement describing the object we want to cycle through different values, for example a variable named `i`, and then a range of values we want object `i` to take on with the gap between different values in parentheses. Then, in brackets, we type the commands we want to apply to the data referencing the object that takes on different values as `` `i' `` For example, let's say that we had some variable that takes on values between 5 and 11 and we wanted to regress two variables on each other for the odd numbers and then summarize the covariates for the subgroups of consideration. Then, we would type:

```
forvalues i=5(2)11 {

display "VariableName1 Has Value "`i'"

regress VariableName2 VariableName3 if VariableName1==`i'

summarize VariableName3 if VariableName1==`i'

}
```

It's good practice to include the first statement in the bracket in `forvalues` loops, as the command does not specify what value object `i` takes on otherwise, and it can make reading log files much easier. Note that we reference object `i` in the brackets as `` `i' ``, even when using the object in some text statement.

The `foreach` command

The `forvalues` command is useful for operations involving sequential numbers. However, sometimes we want to run loops over non-sequential values, or run loops over objects other than sequences of numbers. The `foreach` command is built for these other kinds of loops. To use this command, type `foreach` and, as previously, specify the object that we would like to loop over, for example `i`. Now, we choose what type of object we would like to loop over. If we loop over some generic list of names or numbers, then we can simply type `in` and then list the individual values or text strings we want to loop over. The syntax from there out is similar to the previous set-up we used for the `forvalues` command. For example, if we wanted to run the previous regression when VariableName1 takes on values 6, 7 and 10, we would type:

```
foreach i in 6 7 10 {
```

```
display "VariableName1 Has Value "`i'"

regress VariableName2 VariableName3 if VariableName1==`i'

summarize VariableName3 if VariableName1==`i'

}
```

The `foreach` command is much more flexible than the `forvalues` command, as we are able to loop over other objects besides numbers. For example, we would use a `foreach` command to loop over different covariates we could type:

```
foreach var in VariableName2 VariableName3 {

display "Covariate is `var'"

regress `var'

}
```

The type of list we are using here is a generic list in that it can take on any form. However, there are more efficient ways of coding `foreach` loops over different types of lists that would allow us to use shortcuts without having to spell out each individual element of the list over which we are looping. These different types of lists that we may use are described in the STATA help manual, but they will not be necessary for the sorts of problems we might do in the homework.