Programming and Post-Estimation

• Bootstrapping

• Monte Carlo

• Post-Estimation Simulation (Clarify)

• Extending Clarify to Other Models
  - Censored Probit Example
What is Bootstrapping?

- A computer-simulated nonparametric technique for making inferences about a population parameter based on sample statistics.
- If the sample is a good approximation of the population, the sampling distribution of interest can be estimated by generating a large number of new samples from the original.
- Useful when no analytic formula for the sampling distribution is available.
1. Obtain a Sample from the population of interest. Call this $\mathbf{x} = (x_1, x_2, \ldots, x_n)$.

2. Re-sample based on $\mathbf{x}$ by randomly sampling with replacement from it.

3. Generate many such samples, $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^B$ – each of length $n$.

4. Estimate the desired parameter in each sample, $s(\mathbf{x}^1), s(\mathbf{x}^2), \ldots, s(\mathbf{x}^B)$.

5. For instance the bootstrap estimate of the standard error is the standard deviation of the bootstrap replications.

$$S\hat{E}_B = \sqrt{\sum_{b=1}^{B} [s(\mathbf{x}^B) - \frac{1}{B} \sum_{b=1}^{B} s(\mathbf{x}^b)]^2 / (B - 1)}$$
Example: Standard Error of a Sample Mean Canned in Stata

```
use I:\general\PRISM Programming\auto.dta", clear
 Type: sum mpg
```

```
<table>
<thead>
<tr>
<th>Variable</th>
<th>Obs</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpg</td>
<td>74</td>
<td>21.2973</td>
<td>6.785503</td>
<td>12</td>
<td>41</td>
</tr>
</tbody>
</table>
```
Example: Standard Error of a Sample Mean Canned in Stata

```
bootstrap "sum mpg" r(mean), reps(1000)
```

\[ \bar{x}^B - \bar{x} \]

\[ \approx 21.2973 \pm 1.96 \times 0.6790806 \]
Example: Difference of Medians Test

<table>
<thead>
<tr>
<th>Percentiles</th>
<th>Smallest</th>
<th>Obe</th>
<th>Sum of Ugt.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1%</td>
<td>143</td>
<td>147</td>
<td>24</td>
</tr>
<tr>
<td>10%</td>
<td>157</td>
<td>154</td>
<td>24</td>
</tr>
<tr>
<td>25%</td>
<td>170</td>
<td>154</td>
<td>74</td>
</tr>
<tr>
<td>50%</td>
<td>192.5</td>
<td>Mean</td>
<td>189.7324</td>
</tr>
<tr>
<td>75%</td>
<td>204</td>
<td>Std. Dev.</td>
<td>22.26624</td>
</tr>
<tr>
<td>90%</td>
<td>218</td>
<td>Variance</td>
<td>495.2829</td>
</tr>
<tr>
<td>99%</td>
<td>223</td>
<td>Skewness</td>
<td>-0.0490946</td>
</tr>
<tr>
<td></td>
<td>233</td>
<td>Kurtosis</td>
<td>2.04158</td>
</tr>
</tbody>
</table>

return list

```
return list
```

```
sum length, detail
```

```
<table>
<thead>
<tr>
<th>Percentiles</th>
<th>Smallest</th>
<th>Obe</th>
<th>Sum of Ugt.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1%</td>
<td>143</td>
<td>147</td>
<td>24</td>
</tr>
<tr>
<td>10%</td>
<td>157</td>
<td>154</td>
<td>24</td>
</tr>
<tr>
<td>25%</td>
<td>170</td>
<td>154</td>
<td>74</td>
</tr>
<tr>
<td>50%</td>
<td>192.5</td>
<td>Mean</td>
<td>189.7324</td>
</tr>
<tr>
<td>75%</td>
<td>204</td>
<td>Std. Dev.</td>
<td>22.26624</td>
</tr>
<tr>
<td>90%</td>
<td>218</td>
<td>Variance</td>
<td>495.2829</td>
</tr>
<tr>
<td>99%</td>
<td>223</td>
<td>Skewness</td>
<td>-0.0490946</td>
</tr>
<tr>
<td></td>
<td>233</td>
<td>Kurtosis</td>
<td>2.04158</td>
</tr>
</tbody>
</table>

```

types:

```
types:
```
Example: Difference of Medians Test
Example: Difference of Medians Test

```
program mymedian, rclass
    quietly summarize length, detail
    return scalar `median1' = r(p50)
    quietly summarize displacement, detail
    return scalar `median2' = r(p50)
    return scalar diff = return(median1) - return(median2)
end
```
Example: Difference of Medians Test

```stata
do "C:\DOCU\1\ADMIN\1\LOCAL\1\Temp\STD01000000.tnp"

program mmedian, rclass
    quietly summarize length, detail
    return scalar median = r(p50)
    quietly summarize displacement, detail
    return scalar median2 = r(p50)
end

bootstrap "mmedian" r(diff), reps(1000)
```

The medians are not very different.
What is Monte Carlo Simulation?

- Uses the observation of random samples from known populations of simulated data to track the behavior of a statistic.
- If the sampling distribution of a statistic is the density function of values it could take on in a given population, then its estimate is the relative frequency distribution of the values that were actually observed in many samples drawn from that population.
- Since we can generate the population to have any characteristics we wish, Monte Carlos are very flexible.
How do I do it?

1. Determine what the “population” is.
2. Sample from the “population”
3. Calculate the estimator of interest \( \hat{\theta} \). Save this value.
4. Repeat steps 2 and 3 many times.
5. Construct the frequency distribution of the \( \hat{\theta} \) values. This is the Monte Carlo estimate of the sampling distribution of \( \theta \) under the conditions you specified for the population.
An Example: Coin Toss

- If you toss a fair coin 10 times, what is the probability of obtaining exactly 3 heads?
- By the binomial probability distribution:

\[
\frac{10!}{3!(10-3)!} 0.5^3 \times 0.5^7 \approx 0.1172
\]

However, if we didn’t know how to use the binomial distribution, we could toss a fair coin 10 times in a repeated number of trials and simulate this probability.
Coin Toss Monte Carlo
Coin Toss Monte Carlo

```stata
set more off
set obs 10
gen counter=0
local i=1
while `i' <=10000 {
    quietly gen heads=0
    quietly replace heads = heads + int(uniform()*2)
    quietly egen sum = sum(heads)
    quietly replace counter = counter+1 if sum==3
    drop heads sum
disp `i'
    local i=`i'+1
}
```
Coin Toss Monte Carlo

```
dc "C:\DATA\Coin\Monte Carlo\" prob = counter/10000
sum prob

end of do-file

. gen prob = counter/10000
. sum prob

Variable | Obs | Mean | Std. Dev. | Min | Max
---------|-----|------|-----------|-----|-----
prob     | 10  | .1143| 0         | .1143| .1143
```
Programming the Coin Toss Monte Carlo

- Allows us to automate the experiment, controlling:
  - The number of tosses
    (e.g. 10, or something else)
  - The number of trials
    (e.g. 10,000 or something else)
  - The number of successes
    (e.g. 3 or something else)
Coin Toss Program
program coinflip

set more off
set obs 2
gen counter 0
local i=1
while `i' < `3' {
    quietly gen heads=0
    quietly replace heads = heads + int(uniform()*2)
    quietly egen sum = sum(heads)
    quietly replace counter = counter+1 if sum==1
    drop heads sum
    disp `i'
    local i=`i'+1
}

local i=`i'+1

gen prob = counter/`3'
sum prob

end
Coin Toss Program

```
do "C:\DOCUME~1\Kevin\LOCALS\1\Temp\STD00000000.tmp"

program coinflip
1.
2.        set more off
3.        gen counter=0
4.        local i=1
5.        while `i' <= `3' {
6.            quietly gen heads=0
7.            quietly replace heads - heads + int(uniform()*2)
8.            gen sum - sum(heads)
9.            quietly replace counter = counter+1 if sum==`1'
10.        drop heads sum
11.        disp `i'
12.        local i=`i+1'
13.        }
14.        gen prob = counter/3
15.        sum prob

end of do-file

clear
```

```
coinflip 3 10 10000
```
Coin Toss Program
Another Monte Carlo Application:
The Logic of Post Estimation Simulation

So, you estimate a model... and you want to say something substantive with quantities of interest:

Predicted or Expected Values of DV = $X_\mu \hat{\beta}$

First Differences = $X_{+\sigma} \hat{\beta} - X_\mu \hat{\beta}$

The problem is that our $\hat{\beta}$s are uncertain!

The solution is we know how uncertain.
Monte Carlo Simulation of Parameters

In order to capture the uncertainty, we draw simulated $\hat{\beta}_s$ from the multivariate* normal distribution.

Then we use these simulated parameters to calculate many draws of the same quantity of interest.

\[
\text{Standard Deviation} = \hat{\sigma}_1
\]
Simulating Quantities of Interest

In practice...

\[ Y_i \sim f(\theta_i, \alpha), \quad \theta_i = g(X_i, \beta) \]
\[ Y_i \sim N(\mu_i, \sigma^2), \quad \mu_i = g(X_i, \beta) = \beta_0 + X_{i1} \beta_1 + X_{i2} \beta_2 + \cdots \]

\[ \hat{\gamma} = \begin{bmatrix} \hat{\beta}_1 \\ \hat{\beta}_2 \\ \vdots \\ \hat{\alpha}_1 \end{bmatrix} \quad \hat{V}(\hat{\gamma}) = \begin{bmatrix} v_{\hat{\beta}_{11}} & v_{\hat{\beta}_{12}} & \cdots & v_{\hat{\beta}_{1\alpha}} \\ v_{\hat{\beta}_{21}} & v_{\hat{\beta}_{22}} & \cdots & v_{\hat{\beta}_{2\alpha}} \\ \vdots & \vdots & \ddots & \vdots \\ v_{\hat{\alpha}_1} & v_{\hat{\alpha}_2} & \cdots & v_{\hat{\alpha}} \end{bmatrix} \]

we simulate parameters with M draws from the multivariate normal distribution...

\[ \tilde{\gamma} \sim N(\hat{\gamma}, \hat{V}) \]

- Choose a starting scenario, \( X_c \).
- Draw one value of \( \tilde{\gamma} \), and compute \( \tilde{\theta}_c = g(X_c, \tilde{\beta}) \).
- Simulate the outcome \( \tilde{Y}_c \), by taking a random draw from \( f(\tilde{\theta}_c, \tilde{\alpha}) \).
- Repeat M times to get the distribution of \( Y_c \).
Clarify (King et. al. A JPS 1999)

- **estsimp** – estimates the model and simulates the parameters
  - This command **must** precede your regression command
  - e.g.: `estsimp logit y x1 x2 x3 x4`
  - This will save simulated $\beta$s to your dataset!

- **setx** – sets the values for the IVs (the Xs)
  - Used after model estimation to set values of the Xs
  - e.g.: `setx x1 mean x2 p20 x3 .4 x4[16], nocwdel`
  - functions = mean|median|min|max|p#|math|#|‘macro’|varname[#]
  - reset values by re-issuing the command, e.g.: `setx x1 median`

- **simqi** – simulates the quantities of interest
  - Automates the simulation of quantities of interest for the X values you just set.
  - e.g.: `simqi, prval(1)`
  - e.g.: `simqi, fd(prval(1)) changex(x4 p25 p75)`
You Can Use Clarify, but you Don’t have to.

Models Currently Supported by Clarify

- regress
- logit
- probit
- ologit
-oprobit
- mlogit
- poisson
- nbreg
- sureg
- weibull

But, you really don’t need Clarify to do this, so you can simulate quantities of interest for any model!

- Easy to simulate parameters because Stata saves them after estimation!
- **Program** the correct link function yourself!
An Example: The Censored Probit Model

Selection Equation:
\[ y_{1j} = z_j \gamma + u_{2j} \]

Outcome Equation:
\[ y_{2j} = x_j \beta + u_{1j} \]

Where:
\[ u_1 \sim N(0,1) \]
\[ u_2 \sim N(0,1) \]
\[ \text{corr}(u_1, u_2) = \rho \]

/*Programming Step One: Estimate Model*/
heckprob y2 x1 x2 x3 x4, sel(y1 = z1 z2 z3 z4) robust
An Example:
The Censored Probit Model

Simulate the model parameters by drawing from the multivariate normal distribution.

Note: there are 11 - 4 Xs, 4 Zs, 2 constants, and ρ (the correlation between the errors).

/* Programming Step Two: 
   Draw \( \beta \) from multivariate normal, mean \( \hat{\beta} \) and Covariance Matrix \( \hat{\Sigma} \). */

matrix params = e(b)
matrix P = e(V)
drawnorm b1-b11,
   means(params) cov(P)
double
An Example: The Censored Probit Model

Stata estimates the hyperbolic arctangent of $\rho$, so we must simulate to get the actual $\rho$.  

/ *Programming Step Three: Generate Simulated Rho*/

```stata
gen simrho = (exp(2*b11) - 1)/(exp(2*b11) + 1)
```
An Example: The Censored Probit Model

Initiate a looping structure to generate \( m \) (in this case 1,000) simulated first differences for the effect of \( x_1 \) on \( y_2 \) comparing when \( x_1 \) is at its mean (the base model) to when \( x_1 \) is at a value two standard deviations (denoted \( \_m2sd \) below its mean.

/*ProgramStep Four: The Loop*/
local i = 1
/*A. Generate variables that will be used to fill in a cell of Substantive Table*/
generate base_y2=.
generate x1_m2sd=.
while `i' <= 1000 {
/*B. Generate \( z_\gamma \) for the selection equation.*/
quietly generate select = b6[`i'] + (b7[`i']*z1) + (b8[`i']*z2) + (b9[`i']*z3) + (b10[`i']*z4)
/*C. Generate \( x_b \) for the outcome equation.*/
quietly generate outcome = b1[`i'] + (b2[`i']*x1) + (b3[`i']*x2) + (b4[`i']*x3) + (b5[`i']*x4)
An Example:
The Censored Probit Model

This is the meat of the simulation. The first three commands generate the probability of being selected and experiencing the outcome \( p_{11} \) for the base model. For the censored probit, this probability is (Greene 2000, 857):

\[
\Phi_2[\beta \tilde{x}, \gamma \tilde{z}, \rho]
\]

/* Generate first difference*/
quietly generate p_11 =
    binorm(outcome,select,simrho)
quietly summarize p_11, meanonly
quietly replace base_y2=r(mean) in `i'
quietly generate x1_m2sd=outcome -
    (b1[`i'}*x1) + (b1[`i']*-0.2)
quietly generate p11_x1_m2sd =
    binorm(x1_m2sd,select,simrho)
quietly summarize p_x1_m2sd,
    meanonly
quietly replace x1_m2sd=r(mean) in `i'
An Example: The Censored Probit Model

To get the other 999 we drop the three variables we just generated and repeat the loop until `i' = 1,000.

When we're done with the 1,000 simulations, we can use the centile command to get the relevant distributions. To do this for each variable in the model, we would embed this loop within a larger looping structure.

/*Step 5: Do the Loop Again*/

```
drop select outcome p_11 x1_m2sd p11_x1_m2sd
disp `i'
local i=`i'+1
}
```

```
centile base_y2 x1_m2sd, centile(2.5 50 97.5)
```