**Advanced time-series analysis (University of Lund, Economic History Department)**

30 Jan-3 February and 26-30 March 2012

**Lecture 3 Monte Carlo simulations and Bootstrapping.**

***3.a. What is a Monte Carlo simulation?***

Imagine you have a truffle-seeking pig. You would like to know how good she is. You cannot just let her loose in the forest and count the number of truffles he found, since you do not know how rich in truffles the area was. You need a controlled experiment.

So you choose an area, rail it off, and you randomly bury truffles. Now, you can let the pig loose, and you can count how many of the truffles were found. But one experience is not enough: after all, what if your pig was just too lucky or was disturbed by a squirrel? So you should repeat the experiment as many times as possible. As the number of experiments goes toward infinity, the hit ratio of your pig that you calculated from the experiments will get closer and closer to the real hit probability of the pig (law of large numbers…).

This is a Monte Carlo simulation.

Monte Carlo experiments are based on repeated random sampling. It was first used in Los Alamos National Laboratory by John von Neuman, Stanislaw Ulam and Nicholas Metropolis, the first two were mathematicians and the third one was nuclear physicist, to find out how far neutrons would travel in different materials (important for radiation shielding). The codename for the then secret method was Monte Carlo.

We use Monte Carlo experiments in cases when we know the random process behind a phenomenon, but an analytical solution would be far too difficult to achieve.

**3.b A simple example: the critical values for a DF unit-root test**

We know that if the null-hypothesis of the DF test is correct, the distribution of the coefficients will not follow the student's t distribution. As a consequence, you will need different critical values. One way to have an adjusted table is to design an MC experiment!

We are going to assume that the DGP is the following:

$y_t = 1 + y_{t-1} + \varepsilon_t$ where $\varepsilon_t \sim WN(0,1)$ and $y_0 = 1$

So what we are going to do is the following:

We choose sample size, let us say 100. Then we will use a random number generator to create a series based on the above DGP. This is basically a sample. Then we run a basic DF test:

$\Delta y_t = \alpha_0 + \alpha_1 y_{t-1} + u_t$ and we calculate the test statistics as $\alpha_1$ divided by the standard error of it. WE store this statistics.

Then we redo this experiment 9999 times (so we have 10000 experiments in total) and store all the test statistics. Our estimate for the critical value of the unit root test statistics is going to be the lower fifth percentile (in case of a one-tailed test).

*the code in Eviews*

```
wfcreate dfsimu u 1 10000 'we create the workfile called dfsimu with 10000 observations
series testresults 'this is the variable where we store our testresults for final analysis

scalar fivepercentcrit 'this is where we store our estimate of the 5% critical value

for !i=1 to 10000 'we will repeat this 1000 times

series y=0 ' we create the series y with the starting value 0
smpl 2 101 'no we need to reset the sample to 2-101
y=1+y(-1)+nrnd ' we generate the random series according our assumptions

equation eq.ls d(y) c y(-1)  'we run the test regression on the generated 100 observations
smpl @all 'we reset the sample to the 1 10000

testresults(!i)=eq.@tstats(2) 'save the test stat

next 'return to the start of the for loop

fivepercentcrit=@quantile(testresults,.05)
```

By changing the parameters you can come to a table of your own:

Simulated critical values of a DF test (10000 experiments)

| level of significance (one-tailed) | 10% | 5% | 1% |
|---|---|---|---|
| n=100 | | | |
| without constant | -1.599 | -1.950 | -2.566 |
| with constant | -1.466 | -1.871 | -2.547 |
| n=50 | | | |
| without constant | -1.603 | -1.966 | -2.573 |
| with constant | -1.507 | -1.898 | -2.696 |

**3.c. Storage and hunger in an ancient society**

There is a debate on the ability to have long-term storage of grain in pre-industrial societies. The historical evidence is sporadic and not decisive. McCloskey and Nash[1] utilized a simple probabilistic approach to find out whether the storage was. They assumed that grain production followed a normal probability distribution with mean 100 and standard deviation 35 and found out that this leads to very long waiting times between famines. Van Leeuwen et al (2011)[2] continued in their

[1] McCloskey, D. N. and Nash, J. (1984) Corn at interest: the extent and cost of grain storage in medieval England, American Economic Review, 74, 1, 174–87

[2] van Leeuwen, B., Földvári, P. and Pirngruber, R. (2011) Markets in pre-industrial societies: storage in Hellenistic Babylonia in the medieval English mirror. Journal of Global History (2011) 6, pp. 169–193

footsteps and after revising the parameters of the distribution of production and used a Monte Carlo simulation to find out which type of inter-annual grain storage was feasible.

Obviously, without storage you can easily calculate the chance of a hunger. If the production is normally distributed with mean 100 and std. dev. 35, and the average yield is just 40% higher than the famine limit (so the famine limit is 60). All you need is to look it up in a standard normal probability distribution table how likely it is that you obtain a value equal or lower than 60.

Since $z = \frac{60-100}{35} = -1.142$, we are interested in the p-value of the standard normal distribution at z, which is about 0.127. So with this the chance of a hunger would 12.7%. The average waiting time between two famines is $\frac{1-0.127}{0.127} = 6.87$ so roughly 7 years. This is not far from the picture suggested by historical records.

If we assume that there was storage we cannot do this easily, so we need a simulation.

We assume the following rules:

1. If the production is above the famine limit, 0<c<1 share of the output is stored. This is our carryover parameter. The storage cannot be negative. Also you will not store if storage would reduce your consumption below the limit.

2. If the production is below or equal the famine limit, agents will consume as much from the storage that their consumption just exceeds the famine limit.

3. Storage is assumed not to decay over time (this is a simplification).

Obviously, the process is serially correlated so any outcome depends on the history of the process. What we do here is that we simulate 1000 periods 1000 times.

```
'wfcreate storage u 1 1001
series q 'output
series cons 'consumption
series storage=0 'storage starting level
scalar co=0.05 'carryover
series famine 'here we store if there was a famine, 1 means famine 0 otherwise
scalar limit=70 'famine limit
series chfamine 'chance of famine
scalar  waiting 'number of periods between two famines
for !j=1 to 1000
for !i=2 to 1001

q(!i)=35*@nrnd+100 'output in period i is simulated as a normally distributed proces

' if there is no famine, you will store some of the harvest and consume the rest
if q(!i)>limit*(1+co) then
storage(!i)=storage(!i-1)+co*q(!i)
cons(!i)=q(!i)*(1-co)
endif

' if you have just enough you will not store this obviously never happens since we have anormal random variable
and the chance that is takes the value 60 is null. Still, for theoretical consistency I include this here.
if  q(!i)=limit*(1+co) then
storage(!i)=storage(!i-1)
cons(!i)=q(!i)
endif

' if your output was not enough then you will use up the storage provided it is enough
```

```
if q(!i)<limit*(1+co) and storage(!i-1)>=(limit-q(!i)) then
  storage(!i)=storage(!i-1)-(limit-q(!i))
  cons(!i)=limit
endif

' if storage is not enough you use up as much as possible
if q(!i)<limit*(1+co) and storage(!i-1)>0 and storage(!i-1)<(limit-q(!i)) then
  storage(!i)=0
  cons(!i)=q(!i)+storage(!i-1)
endif

'if you do not have storage at all, you obviously cannot do anything
if q(!i)<limit*(1+co) and storage(!i-1)=0  then
  storage(!i)=0
  cons(!i)=q(!i)
  endif

next

smpl 2 1001
famine=cons<limit

smpl 1 1001
chfamine(!j)=@mean(famine)
next

scalar waiting=1/@mean(chfamine)-1
```

I am sure that you could find an even more elegant (=parsimonious) way to write the code but this will do for now. Again, you can create a table:

**Average waiting time between famines under different carryovers**

| carryover | 0% | 1% | 2% | 5% |
|---|---|---|---|---|
| famine limit=60 | 6.9 | 11.8 | 37.2 | 469 |
| famine limit=70 | 4.1 | 5.4 | 7.8 | 123 |

Note: output is assume to be a normally distributed random variable with mean 100 and standard deviation 35

Obviously, the 5% carryover can be rejected, as it would have basically eradicated society wide hunger. Depending on what we believe about the productivity of pre-industrial agriculture, however, some minor storage could have been possible. In the paper we used a production with much lower standard deviation, based on new data. This enabled us to reject the possibility of any systematic storage.

### *3.d. Bootstrapping*

Unlike in case of an MC experiment, we use the bootstrapping when we do not know the Data Generating Process, i.e., we have no idea about the underlying distribution of the data. Since we do not know the population you will create samples using your sample only by drawing with replacement. For this reason we call this procedure resampling. It is important that the new samples should have the same size and the original sample!

For an example we take the data of Austrian conscripts height by Komlos.

Looking at the sample, we estimate the mean at 167.8 cm. it is quite obvious that the heights are not normally distributed. This is not surprising, the data is truncated: recruitment boards anly registered those with a minimum posture.

You can also run a regression:

Dependent Variable: HEIGHTCM
Method: Least Squares
Date: 03/26/12 Time: 19:55
Sample: 1 145802
Included observations: 145802

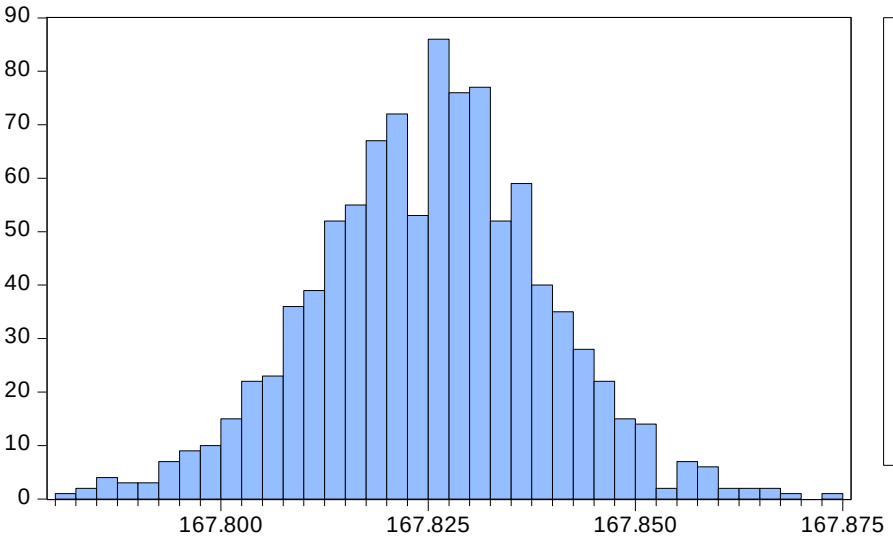
| Variable | Coefficient | Std. Error | t-Statistic | Prob. |
|---|---|---|---|---|
| C | 167.8250 | 0.013812 | 12150.37 | 0.0000 |

| | | | |
|---|---|---|---|
| R-squared | 0.000000 | Mean dependent var | 167.8250 |
| Adjusted R-squared | 0.000000 | S.D. dependent var | 5.274106 |
| S.E. of regression | 5.274106 | Akaike info criterion | 6.163502 |
| Sum squared resid | 4055630. | Schwarz criterion | 6.163570 |
| Log likelihood | -449324.5 | Hannan-Quinn criter. | 6.163523 |
| Durbin-Watson stat | 1.161986 | | |

Alternatively you can have an estimate by bootstrap:

```
 matrix height=@convert(heightcm) 'here I have to convert the heights to a vector
series mean
for !i=1 to 1000
matrix bs=@resample(height) 'resampling with replacement
mtos(bs,h) 'conversion of the vector to a variable
mean(!i)=@mean(h)
next
```

mean.hist




Series: MEAN
Sample 1 145802
Observations 1000

| | |
|---|---|
| Mean | 167.8252 |
| Median | 167.8259 |
| Maximum | 167.8739 |
| Minimum | 167.7812 |
| Std. Dev. | 0.014023 |
| Skewness | -0.045728 |
| Kurtosis | 3.282167 |
| Jarque-Bera | 3.665934 |
| Probability | 0.159938 |

We get about the same mean and standard deviation as in case of a regression with constant. This is not surprising: whatever distribution the data has the sample mean is going to be an unbiased estimator of the expected value of the population. The distribution of the parameter estimate is going to converge to normal. So we did not gain anything from this exercise except that you now can do a bootstrap (actually bootstrapping is rarely very useful).

Let us use bootstrapping in a regression. We regress height on age:

Dependent Variable: HEIGHTCM
Method: Least Squares
Date: 03/26/12 Time: 20:07
Sample: 1 145802
Included observations: 145802

| Variable | Coefficient | Std. Error | t-Statistic | Prob. |
|---|---|---|---|---|
| C | 163.7536 | 0.051930 | 3153.365 | 0.0000 |
| AGE | 0.164741 | 0.002029 | 81.19722 | 0.0000 |

| | | | |
|---|---|---|---|
| R-squared | 0.043263 | Mean dependent var | 167.8250 |
| Adjusted R-squared | 0.043257 | S.D. dependent var | 5.274106 |
| S.E. of regression | 5.158776 | Akaike info criterion | 6.119289 |
| Sum squared resid | 3880171. | Schwarz criterion | 6.119425 |
| Log likelihood | -446100.3 | Hannan-Quinn criter. | 6.119330 |
| F-statistic | 6592.988 | Durbin-Watson stat | 1.214570 |
| Prob(F-statistic) | 0.000000 | | |

So, on average if a recruit was one year older it was 16,5 mm taller in the sample. Our 95% confidence interval for this coefficient would be:

$$0.164 \pm 0.002 \cdot Z_{0.025} = 0.164 \pm 0.002 \cdot 1.96 = 0.16008 - 0.16792$$

I used standard normal distribution instead of a Student's t because at high degrees of freedom (it is now 145800) it converges to standard normal.

How would you do the same with bootstrapping?

1. You run the regression
2. You save the residual and the fitted value
3. You resample the residual and add it to the fitted value. This will be you dependent variable in step 4.
4. You rerun the regression with the variable and store the coefficient
5. You repeat steps 3 and 4 many hundred or thousand times.
6. You take the 2.5 and 97.5 percentiles of the bootstrapped coefficients. These are the limits of your confidence interval.

The Eviews code is:

```
equation eq.ls heightcm c age 'I run the regression
eq.makeresids res 'I store the residuals
series yfit=heightcm-res 'I store fitted values

vector resm=@convert(res) 'here I have to convert the residuals to a vector
series coeff
for !i=1 to 10000
matrix bsres=@resample(resm) 'resampling with replacement
mtos(bsres,residual) 'conversion of the vector to a variable
equation eq.ls yfit+residual(1) c age 'I rerun the regression on the bootstrapped dependent var
coeff(!i)=eq.@coefs(2) ' I save the coefficient
next
```

```
scalar lower=@quantile(coeff,0.025)
scalar upper=@quantile(coeff, 0.975)
```

After 1000 repeated resampling, I got 0.1608-0.1686 as confidence interval.

After 10000 repetitions I obtain 0.1607-0.1687, so having much more resampling did not improve the results much.