Introduction to Eviews programming:

Nico Katzke (BER) *nfkatzke@gmail.com*

In this short (and by no means comprehensive) tutorial I intend to introduce certain concepts of programming in Eviews to you. Most of these we will be using either in this course or in financial econometrics. Keep this handy when we do programming in Eviews.

I regard the following as an essential (and more detailed) guide to coding in Eviews:

https://sites.google.com/site/davesmant/courses/working-with-eviews/intro-eviews-programming

Eviews code is more similar to very early coding structures (even resembles bash commands in Linux) than programs like R or Python. Nonetheless, it offers a good introduction to coding principles, especially if you are new to coding, which you might find useful as you go through this document

Most of the models that we fit will be of the "drop-down" list type, but we will at times need to program it ourselves, as this provides us with more flexibility.

To start, consider the following Objects in the main screen:

At the top of the screen we have the command window, of which I am sure most of you are by now aware.

If you want to create a numeric variable, type in the command window: series x = (whatever)

Typing in this box and pressing OK yields your intended result. When we, however, want to run a larger set of commands – we need to use the program option. *File / New / Program.*

Now you have a window in which you can type large sets of coding, while at the end you can hit *Run*.

Do the following as an exercise:

Create a Workfile with the following dates and frequency:

Monthly, from 1990m01 --> 2013m01

Then Copy and Paste the following into your Program window:

.....

```
'whatever follows the ' at the beginning of a line, tells Eviews what follows is text. Thereafter we can type anything and it won't affect our results.
series u=0.5*nrnd
```

```
smpl @first @first+2  'Starting a line with smpl implies we are trimming the sample. Here from first +3 → last obs
'We first set the values of all the newly created series equal to 0: remember otherwise it would be NA, making it impossible for Eviews to run the program.
series y1=0
series t=0
series s=0
series i=0
series c1=0
smpl @first +3 @last
t = 0.005*@trend + 0.007*@trend^2 + u '@trend is simply the time trend (i.e. each time period is an integer: 1,2,3….
s= sin(1.4*@trend)+u
i= 0.97*i(-1) +u
c1= sin(0.15*@trend)+0.2*u
y1 = 0.2 + 7*s + 0.52*t + 3*i + 15*c1+ u
'now click on Run at the top left.
```

.....

Now you will see in your workfile a bunch of series has now been created.

If you want to then make some transformations, you can then use the command window at the top.

E.g., a typical command would be:

```
series dly = dlog(y1)
```

<u>The following will be useful to remember when programming:</u>

Dates:

If you want to change the dates in your active sample, type the following:

Smpl @first 2010m01	(first date – 2010m01)
Smpl 2008Q1 @last	(2008m01 – the last date)
Smpl @all	(entire sample)
smpl 1995-01-01 @last	(daily)

Math operators:

Expression	Operator Description
+	add, x+y, adds the contents of X and Y
–	subtract, x–y, subtracts the contents of Y from X
*	multiply, x*y, multiplies the contents of X by Y
/	divide, x/y, divides the contents of X by Y
^	raise to the power, $x^{\wedge}y$, raises X to the power of Y
>	greater than, x>y, takes the value 1 if X exceeds Y, and 0 otherwise
<	less than, x<y, takes the value 1 if Y exceeds X, and 0 otherwise
=	equal to, x=y, takes the value 1 if X and Y are equal, and 0 otherwise
<>	not equal to, x<>y, takes the value 1 if X and Y are not equal, and 0 if they are equal
<=	less than or equal to, x<=y, takes the value 1 if X does not exceed Y, and 0 otherwise
>=	greater than or equal to, x>=y, takes the value 1 if Y does not exceed X, and 0 otherwise
and	logical and, x and y, takes the value 1 if both X and Y are nonzero, and 0 otherwise
or	logical or, x or y, takes the value 1 if either X or Y is nonzero, and 0 otherwise

Descriptive functions:

@abs(x) → absolute value of x

@ exp(x) → $exponential\ value$

@ceiling @floor → smallest (largest)integer no less (greater)than x. @ceiling$(4.3) = 5$

@recode(statement, b, c) → Returns b if statement is true, otherwise it returns c.

@inv(x) → reciprocal

@nan(x, y) → returns x if x <> NA, and y if x = NA.

** This we do to avoid Eviews failing to model our request based on NA datapoints.

@round(c) = rounds x to the nearest integer.

Time series functions:

$X(-k)$ → kth lag of X

$d(X) = First\ difference;\quad d(x,n) = n^{th} difference\ of\ x$

$genr\ dlx = dlog(X)$ → often used in time series econometrics

$dlog(X, 1,4)$ → dlog of X with seasonal difference at 4thlag.

$@pch$ → one period percentage change $\left(\frac{X_t}{X_{t-1}} - 1\right)$

$@pcha$ → one period percentage change $\left(\frac{X_t}{X_{t-1}} - 1\right)$: $annualized$

Control variables:

These are especially important when programming to make life easier by storing and naming values. It can be described as follows:

Control variables we use in place of numerical values in programming code. Once it has a value assigned, it can be used anywhere in the code where you would ordinarily use a value.

The naming convention is to use a "!" mark, after which follows the name used. Think of it as a **placeholder:**

E.g.: !x !pi !time.

Thereafter we use an equation to specify the values, as:

!ar = 3 !time = 5

Once the value is assigned, it can be used in expressions throughout our coding.

E.g.: !time = !time + 3 smpl 1990q1 + !x 2012q1 log(y(- !lag))

Note that such control variables are deleted after you press run, and as such is used only when programming...

String variables

A variable with text values are called *string variables*. A variable with such values should be assigned using quotation marks: "". The variable itself should be named using %.

If we want to use string variables which again only exist in the programming code – we type it as: %eqname = "equation1"

%MaxValue = "3"

For the last example, @val will be useful, as it **converts** a **string** representation of a number **into a numeric value** that can be used as a number in other functions...

Thus eviews will return the string number in quotations "string number" as the number that the strings represents for use perhaps in another equation.

Thus from above: @val(%maxarterm) = 3

@str(num) does the converse – creates a string or "word" from a value.

Thus e.g. if you want to create a matrix that provides a value as a word, e.g.: Maximum level, type: @str(@max(x))

An example:

%maxar = "3"

!maxAR = @val(%maxar)

‘and then we can use it later e.g. as:

%ar = “0”

for !i = 0 to !maxAR

...

The obvious question now is why go through this effort and not just type the max ar we want (i.e. type 3) i.s.o. typing %maxar = “3” ?!!!

Simply, if you have a large piece of coding and want to change the max ar term from 3 to 4 to see what happens to the output, you’d have to go change all the 3’s to 4’s in your coding. Instead you can just change one value: retype it as: %maxar = “4”.

This will save you lots of time and make using your coding easier for others.

We can also combine several string variables to make a sentence / statement.

e.g.: %Exch = “Dollar / Pound”

%freqmon = “monthly”

%exchtmon = %exch + %freqmon

Thus %exchmon will return: “Dollar / Pound monthly”.

Typically such variables are used in tables / matrices:

e.g.: ertable (1,1) = %exchmon

Here is an example of how we can use the strong variables in designing commands for Eviews when programming:

!repeats = 500

%source = “ draws from normal”

%method = “cauchy ”

%process = @str(!repeats) + %source + %method + “distribution”

So if we want to enter in our matrix the method, we type:

table(1,1) = %process

and then the entry in (1,1) of the table would be:

500 draws from normal Cauchy distribution

We can also use curly brackets brackets { } around string variables to use it in equations when we want to use what is referred to...

Suppose we have a variable GDP.

Suppose then in our coding we may use the same procedure on several target variables, so that we want to save the effort of having to reprogram the process for each, that we rather type it as:

%target = "GDP"

Then if we wanted to use GDP, we cannot specify it as:

Series y =%target

Because Eviews will then use the string variable "GDP" and not the variable it is referring to (the actual variable GDP).

To do this, (and think of it as diverting a call on your cellphone...) use { }.

Series y = {%target}, which will be interpreted as setting y = GDP variable.

And then we can create, e.g., an equation with it looking as follows (**and notice the convention for creating an equation named eq01**):

Equation eq01. ls {%target} c dlog{%target} (-1)

This would create an equation named eq01 which, under the current specification, is:

ls GDP c dlog(GDP(-1))

To now repeat this ls on some other target all you need to do is change the string:

%target = "Inv"

You can also use references within references (quite "inception" like!), such as:

%var = "x"

%use = "var"

Then series {{%use}} = 0

Will be similar to typing: series x = 0.

Another example - figure out what will happen below:

%x = dlog(income)

%lagseries = "x c x(-1) x(-2) x(-3)"

equation eq01. Ls ({%lagseries})

show eq01.

By pressing OK --> Eviews will run a regression and open the results for the LS equation specified.

Creating a matrix, vector, table:

To create a 3X3 **matrix** called Mat type:

Matrix (3, 3) Mat

To create a **vector** having 5 values and which is called x, type:

Coef(5) x

To make a 3X3 table called tab, type:

table (3, 3) tab.

To set the first entry of the Tab equal to the F stat of the ls equation specified above, type:

Tab(1, 1) = eq01. @f.

Statements of intent:

If statements use: if, else, then, and / or pretty much like you would in normal sentences of speach. Just remember to end your command by using: endif.

It is thus of the form:

If ... and ... or ... else ... (and any other legitimate conjuncture you'd want to use) ... **endif**

Consider the following example:

If %x = dlog

then

series income = dlog(income)

endif

Can you figure out the meaning?

Or consider the next example:

%target = income

!k = 100 000

If @val({%target}) > 1*!k

and @val({%target}) <2*!k

then tab(1,1) = medium

else tab(1,1) = “”

endif

This means that if the income level is between 100 000 and 200 000 the first entry of the table should show “medium”, and if its not it should remain blank.

A shorthand formula for an if that has one condition and two scenarios for true and false (similar to that used in excel) is @**recode**:

Thus we could type: @recode(condtion to hold, “value if true”, “value if false”)

Thus for the above if it were only to get the answer (and not to tabulate it) it would be: @recode(@val({%target}), “medium”, “”)

Loops

Programs can also contain “loops” that repeat many times over.

For Loop

This allows the user to repeat a set of commands for different values of a control / string variable.

It is of the form: **for** ..... **next (with pretty much anything in between)**

The program could also run for different values of a control variable implying you not needing to re-specify all the equations for every different value. We then need to specify a **starting value** for the variable, followed by the word **to** and then an end value. You could also use the step function to specify how much the control variable should change.

Now, suppose we have a series, **mktcap**, that includes all the firms on the JSE’s Market Capitalization (measure of the respective businesses’ size).

Suppose that we are then interested in creating 3 series that splits all these factors into Cap(Small), Cap(Med), Cap(Large).

The factors should be as follows:

0 – R2000 000 = CapSmall

R2000 000 – R4 000 000 = Med Cap

R4000 000 – R6 000 000 = Large Cap

```
%2 = "sml"
%4 = "med"
%6 = "lrg"
For !k=2 to 6 step 2
Series cap{%{!k}} = (1000000*({!k} - 2)) < mktcap < (1000000*({!k}))
next
```

Running 10 000 regressions:

In Eviews, Commands can also be written in "object form." In a program, use of the object form is often the easiest way to set options that would normally be set by making choices in dialog boxes. Sometimes you don't want all the output from each command in a program. Suppose you want to run (e.g. for a monte carlo experiment) 10 000 regressions, saving only one coefficient from each regression to be used downstream, ignoring all the other output such as R^2 and other statistics. Now, if we did the following:

```
for !i = 1 to 10000
ls y c x
next
```

Eviews will create 1000 different windows (it would essentially crash). **This is sub-optimal.**

This program:

```
equation eq
for !i = 1 to 10000
eq.ls y c x
next
```

Will now use the **object form of the equation** and create a single object with a single, re-used window.

This can now be used to extract the coefficient or statistic of choice, and saved at each iteration (try saving the first coefficient at each step yourself. NB to remember – you need to create an object to save it in!)

To conduct a simple rolling regression analysis, use the loop:

```
For !period = 4 to 24 step 4
Smpl 1995m04 + !period  1999m01 + (!period + 8)
Equation eq0{!period}.ls dlog(y1) c ar(1)
                Next
```

This concludes our introduction into the programming capabilities of Eviews.

This should be kept as a handy go-to manual for when we use programs to run in our models.

Let's consider the following coding experiment.

Suppose we want to find the optimal lag length for the ARIMA model of GDP.

We thus want to program the optimal lags for the AR and MA terms in terms of either the AIC, SBC or HQIC.

Work through the code (Eviewscodearima.prg) that I shared.

Remember to specify your own max lags, and also edit it to conform to your data series and to give the output to the criterion of your choice.