

# Chapter 1


## Introduction to EViews 6.0

EViews is a simple, interactive econometrics package which proves many tools used in econometrics. It provides users with several convenient ways of performing analysis including a Windows and a command line interfaces. Many operations that can be implemented using menus may also be entered into the command window, or placed in programs for batch processing. The possibility of using interactive features like windows, buttons and menus makes EViews a user-friendly software.


In this chapter we briefly introduce you main features of the language, will show you the use of some important commands which will be used further in this textbook. We will start with the interactive Windows interface and then go into more detailed description about the EViews' batch processing language and advanced programming features.

SIMPLY CLEVER

ŠKODA



**We will turn your CV into an opportunity of a lifetime**



Do you like cars? Would you like to be a part of a successful brand? We will appreciate and reward both your enthusiasm and talent. Send us your CV. You will be surprised where it can take you.

Send us your CV on [www.employerforlife.com](http://www.employerforlife.com)

## 1.1 Workfiles in EViews

EViews' design allows you to work with various types of data in an intuitive and convenient way. We start with the basic concepts of how to working with datasets using workfiles, and describing simple methods to get you started on creating and working with workfiles in EViews.

In the majority of cases you start your work in EViews with a workfile – a container for EViews objects. Before you perform any tasks with EViews' objects you first have to either create a new workfile or to load an existing workfile from the disc.

In order to create a new workfile you need to provide and information about its structure. Select *File/New/Workfile* from the main menu to open the *Workfile Create* dialog. On the left side of the dialog is a combo box for describing the underlying structure of your dataset. You have to choose between three options regarding the structure of your data – the *Dated - regular frequency*, the *Unstructured*, and the *Balanced Panel* settings. *Dated - regular frequency* is normally used to work with a simple time series data, *Balanced Panel* is used for a simple panel dataset and *Unstructured* options is used for all other cases.

For the *Dated - regular frequency*, you may choose among the following options: Annual, Semi-annual, Quarterly, Monthly, Weekly, Daily - 5 day week, Daily - 7 day week and Integer date. EViews will also ask you to enter a Start date and End date for your workfile. When you click on OK, EViews will create a regular frequency workfile with the specified number of observations and the associated identifiers.

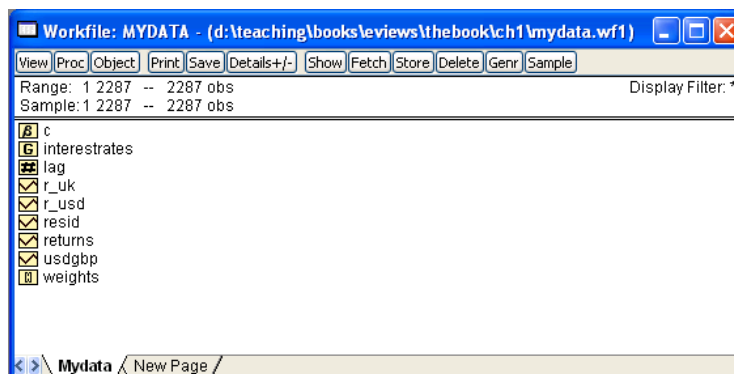
The *Unstructured data* simply uses integer identifiers instead of date identifiers. You would use this type of workfile while performing a crosssectional analysis. Under this option you would only need to enter the number of observations.

The *Balanced Panel* entry provides a method of describing a regular frequency panel data structure. Panel data is the term that we use to refer to data containing observations with both a group (cross-section) and time series identifiers. This entry may be used when you wish to create a balanced structure in which every crosssection follows the same regular frequency with the same date observations. Under this option you should specify a desired Frequency, a Start and End date, and Number of cross sections.

Another method of creating an EViews workfile is to open a non-EViews data source and to read the data into an new EViews workfile. To open a foreign data source, first select *File/Open/Foreign Data as Workfile*. First, EViews will open a series of dialogs asking you to describe and select data to be read. The data will be read into the new workfile, which will be resized to fit. If there is a single date series

in the data, EViews will attempt to restructure the workfile using the date series. A typical workfile view is given in Figure 1.1.

**Figure 1.1:** Workfile in EViews



Workfiles contain the EViews objects and provide you an access to your data and tools for working with this data.

Below the titlebar of a workfile is a button bar that provides you with easy access to some useful workfile operations. These buttons are simply shortcuts to items that may be accessed from the main EViews menu. Below the toolbar are two lines of status information where EViews displays the range of the workfile, the current sample (the range of observations that are to be used in calculations), and the display filter (rule used in choosing a subset of objects to display in the workfile window). You may change the range, sample, and filter by double clicking on these labels and entering the relevant information in the dialog boxes. The contents of your workfile page is provided in in the workfile directory. You can find there all named objects, sorted by name, with an icon showing the object type.

Push the *Save* button on the workfile toolbar to save a copy of the workfile on disk. You can also save a file using the *File/ Save As* or *File/Save* choices from the main menu. By default, EViews will save your data in the EViews workfile format, the extension ".wf1". You may also choose to save the data in your workfile in a foreign data format by selecting a different format in the combo box.

When you click on the *Save* button, EViews will display a dialog showing the current global default options for saving the data in your workfile. You should choose between saving your series data in either *Single precision* or *Double precision*. Single precision will create smaller files on disk, but saves the data with fewer digits of accuracy (7 versus 16). You may also choose to save your data in compressed or non-compressed form.

## 1.2 Objects

All information in EViews is stored in objects. Each object consists of a collection of information related to a particular area of analysis. For example, a series object is a collection of information related to a set of observations on a particular variable. An equation object is a collection of information related to the relationship between a collection of variables. Together with the data information, EViews also associates procedures which can be used to process the data. For example, an equation object contains all of the information relevant to an estimated relationship, you can examine results, perform hypothesis and specification tests, or generate forecasts at any time. Managing your work is simplified since only a single object is used to work with an entire collection of data and results.

Each object contains various types of information. For example, series, matrix, vector, and scalar objects contain numeric data while equations and systems contain complete information about the specification of the equation or system, the estimation results. Graphs and tables contain numeric, text, and formatting information. Since objects contain various kinds of data, you will work with different objects in different ways.

I joined MITAS because  
I wanted **real responsibility**

The Graduate Programme  
for Engineers and Geoscientists  
[www.discovermitas.com](http://www.discovermitas.com)



**Month 16**

I was a construction supervisor in the North Sea advising and helping foremen solve problems

Real work  
International opportunities  
Three work placements







EViews provides you with different tools for each object. These tools are views and procedures which often display tables or graphs in the object's window. Using procedures you can create new objects. For example, equation objects contain procedures for generating new series containing the residuals, fitted values, or forecasts from the estimated equation. You select procedures from the *Proc* menu and views from the *View* on the object's toolbar or from the EViews main menu.

There are a number of different types of objects, each of which serves a unique function. Most objects are represented by a unique icon which is displayed in the workfile window. The basic object icons are:

**Figure 1.2:** Object Icons



In order to create an object, create or loaded a workfile first and then select *Object/New Object* from the main menu. You will see the *New Object* dialog box where you can click on the type of object you want to create. For some object types, a second dialog box will open prompting you to describe your object in more detail. For example, if you select *Equation*, you will see a dialog box prompting you for additional information.

Once you have selected your object, you can open it by double clicking anywhere in the highlighted area. If you double click on a single selected object, you will open an object window. If you select multiple graphs or series and double click, a pop-up menu appears, giving you the option of creating and opening new objects (group, equation, VAR, graph) or displaying each of the selected objects in its own window. Note that if you select multiple graphs and double click or select *View/Open as One Window*, all of the graphs will be merged into a single graph and displayed in a single window. Other multiple item selections are not valid, and will either issue an error or will simply not respond when you double click. When you open an object, EViews will display the view that was displayed the last time the object was opened (if an object has never been opened, EViews will use a default view). The exception to this general rule is for those views that require significant computational time. In this latter case, the current view will revert to the default.

An alternative method of selecting and opening objects is to "show" the item. Click on the *Show* button on the toolbar, or select *Quick/Show* from the menu and

type in the object name or names. Showing an object works exactly as if you first selected the object or objects, and then opened your selection.

**Object windows** are the windows that are displayed when you open an object or object container. An object's window will contain either a view of the object, or the results of an object procedure. One of the more important features of EViews is that you can display object windows for a number of items at the same time.

Let us look again at a typical object window:

**Figure 1.3:** Object Window in EViews

RETURNS	
Last updated: 09/02/08 - 14:58	
Modified: 1 2287 // returns = dlog(usdgbp)	
1	NA
2	0.006779
3	-0.002640
4	0.000991
5	-0.004135
6	0.003804
7	0.010674
8	-0.001308
9	-0.009532
10	-0.009123
11	-0.002169
12	0.003834
13	

Here, we see the series window for RETURNS. At the top of the window there is a toolbar containing a number of buttons that provide easy access to frequently used menu items. These toolbars will vary across objects. There are several buttons that are found on all object toolbars:

- *View* button lets you change the view that is displayed in the object window. The available choices will differ, depending upon the object type.
- *Proc* button provides access to a menu of procedures that are available for the object.
- *Object* button lets you manage your objects. You can store the object on disk, name, delete, copy, or print the object.
- *Print* button lets you print the current view of the object (the window contents).
- *Name* button allows you to name or rename the object.

- *Freeze* button creates a new object graph, table, or text object out of the current view.

There are two distinct methods of duplicating the information in an object: copying and freezing. If you select *Object/Copy* from the menu, EViews will create a new untitled object containing an exact copy of the original object. By exact copy, we mean that the new object duplicates all the features of the original (except for the name). It contains all of the views and procedures of the original object and can be used in future analyses just like the original object. You may also copy an object from the workfile window. Simply highlight the object and click on *Object/Copy Selected* or right mouse click and select *Object/Copy*, then specify the destination name for the object.

The second method of copying information from an object is to freeze a view of the object. If you click *Object/Freeze Output* or press the *Freeze* button on the object's toolbar, a table or graph object is created that duplicates the current view of the original object. Freezing the view makes a copy of the view and turns it into an independent object that will remain even if you delete the original object. A frozen view shows a snapshot of the object at the moment you pushed the button. The primary feature of freezing an object is that the tables and graphs created by freezing may be edited for presentations or reports. Frozen views do not change when the workfile sample or data change.

**ie business school**

#1 EUROPEAN BUSINESS SCHOOL  
FINANCIAL TIMES 2013

**#gobeyond**

**MASTER IN MANAGEMENT**

**Because achieving your dreams is your greatest challenge.** IE Business School's Master in Management taught in English, Spanish or bilingually, trains young high performance professionals at the beginning of their career through an innovative and stimulating program that will help them reach their full potential.

- Choose your area of specialization.
- Customize your master through the different options offered.
- Global Immersion Weeks in locations such as London, Silicon Valley or Shanghai.

*Because you change, we change with you.*

www.ie.edu/master-management | mim.admissions@ie.edu |

To delete an object or objects from your workfile, select the object or objects in the workfile directory and click *Delete* or *Object/Delete Selected* on the workfile toolbar.

### Series

An series object contains a set of observations on a numeric variable. Associated with each observation in the series is a date or observation label. Note that the series object may only be used to hold numeric data. If you wish to work with alphanumeric data, you should employ *alpha* series.

You can create a numeric series by selecting *Object/New Object* from the menu, and then to select *Series*. EViews will open a spreadsheet view of the new series object with all of the observations containing "NA" (the missing value). You may then edit or use expressions to assign values for the series. A second method of creating a series is to generate the series using mathematical expressions. Click on *Quick/Generate Series* in the main EViews menu, and enter an expression defining the series.

Lastly, you may create the series by entering a series command in the command window. Entering an expression of the form:

```
series returns=expression
```

creates a series with the name `returns` and assigns the expression to each observation.

You can edit individual values of the data in a series. First, open the spreadsheet view of the series. Next, make certain that the spreadsheet window is in edit mode (you can use the *Edit +/-* button on the toolbar to toggle between edit mode and protected mode). To change the value for an observation, select the cell, type in the value, and press ENTER.

You can also insert and delete observations in the series. First, click on the cell where you want the new observation to appear. Next, right click and select *Insert Obs* or *Delete Obs* from the menu. You will see a dialog asking how many observations you wish to insert or delete at the current position and whether you wish to insert observations in the selected series or in all of the series in the group. If you choose to insert a single observation, EViews will insert a missing value at the appropriate position and push all of the observations down so that the last observation will be lost from the workfile. If you wish to preserve this observation, you will have to expand the workfile before inserting observations. If you choose to delete an observation, all of the remaining observations will move up, so that you will have a missing value at the end of the workfile range.

### Groups



A group is a list of series names that provides simultaneous access to all of the elements in the list. With a group, you can refer to sets of variables using a single name. Thus, a set of variables may be analyzed using the group object, rather than each one of the individual series. Once a group is defined, you can use the group name in many places to refer to all of the series contained in the group. You would normally create groups of series when you wish to analyze or examine multiple series at the same time. For example, groups are used in computing correlation matrices, testing for cointegration and estimating a VAR or VEC, and graphing series against one another.

There are several ways to create a group. Perhaps the easiest method is to select *Object/New Object* from the main menu or workfile toolbar, click on *Group*. You should enter the names of the series to be included in the group, separated by spaces, and then click OK. A group window will open showing a spreadsheet view of the group.

If you apply an operation to a group, EViews will automatically evaluate the expressions for each observation and display the results as if they were an ordinary series.



"I studied English for 16 years but...  
...I finally learned to speak it in just six lessons"  
Jane, Chinese architect

ENGLISH OUT THERE

Click to hear me talking before and after my unique course download

An equivalent method of creating a group is to select *Quick/Show*, or to click on the *Show* button on the workfile toolbar, and then to enter the list of series, groups and series expressions to be included in the group. You can also create an empty group that may be used for entering new data from the keyboard or pasting data copied from another Windows program.

### Samples

One of the most important concepts in EViews is the sample of observations. The sample is the set of observations in the workfile used for performing statistical procedures. Samples may be specified using ranges of observations and "if conditions" that observations must satisfy to be included. For example, you can tell EViews that you want to work with observations from 1973M1 to 1990M12 and 1995M1 to 2006M12. Or you may want to work with data from 1973M1 to 1978M12 where observations in the *Returns* series are positive. When you create a workfile, the workfile sample is set initially to be the entire range of the workfile. The workfile sample tells EViews what set of observations you wish to use for subsequent operations. You can always determine the current workfile sample of observations by looking at the top of your workfile window. Here the MYDATA workfile consists of 408 observations from January 1973 to December 2006. The current workfile sample uses a subset of those 72 observations between 1973M01 and 1978M12 for which the value of the *Returns* series is positive.

There are four ways to set the workfile sample: you may click on the *Sample* button in the workfile toolbar, you may double click on the sample string display in the workfile window, you can select *Proc/Set Sample* from the main workfile menu, or you may enter a `smpl` command in the command window.

EViews provides special keywords that may make entering sample date pairs easier. First, you can use the keyword `@all`, to refer to the entire workfile range. In the workfile above, entering `@all` in the dialog is equivalent to typing "1973M12006M12". Furthermore, you may use `@first` and `@last` to refer to the first and last observation in the workfile. Thus, the three sample specifications for the above workfile:

`@all`

`@first 2006m12`

`19733m1 @last`

are identical. <sup>1</sup>

---

<sup>1</sup>You may use the IEEE standard format, "YYYY-MM-DD", which uses a four-digit year, followed by a dash, a two-digit month, a second dash, and a two-digit day. The presence of a dash in the format means that you must enclose the date in quotes for EViews to accept this format. For example: "1991-01-03" "1995-07-05" will always be interpreted as January 3, 1991 and July

## Sample Commands

EViews allows you to add conditions to the sample specification. In this case the sample is the intersection of the set of observations defined by the range pairs in the upper window and the set of observations defined by the if conditions. This can be done by typing the expression:

```
smpl 1973m1 1978m12 if returns>0
```

in the command window. You should see the sample change in the workfile window.

Sample range elements may contain mathematical expressions to create date offsets. This feature can be particularly useful in setting up a fixed width window of observations. For example, in the regular frequency monthly workfile above, the sample string: 1973m1 1973m1+11 defines a sample that includes the 12 observations in the calendar year beginning in 1973M1. The offsets are perhaps most useful when combined with the special keywords to trim observations from the beginning or end of the sample. For example, to drop the first observation in your sample, you may use the sample statement:

```
smpl @first+1 @last
```

Accordingly, the following commands generate a cumulative returns series from the price levels one:

```
smpl @first @first
```

```
series returns = 0
```

```
smpl @first+1 @last
```

```
returns = returns(-1) + log(price) - log(price(-1))
```

The first two commands initialize the cumulative returns series at 0, the last two commands compute them recursively all remaining dates. Later we will see how sample offsets can be used to perform the rolling window estimation.

EViews provides you with a method of saving sample information in an object which can then be referred to by name. To create a sample object, select *Object/New Object* from the main menu or the workfile toolbar. When the *New Object* dialog appears, select *Sample* and, optionally provide a name. Click on OK and EViews will open the sample object specification dialog which you should fill out. The sample object now appears in the workfile directory with a double-arrow icon. To declare a sample object using a command, simply issue the **sample** declaration, followed by the name to be given to the sample object, and then the sample string:

---

5, 1995.

```
sample mysample 1973m1 1978m12 if returns>0
```

EViews will create the sample object `MYSAMPLE` which will use observations between 1973:01 and 1978:12, where the cumulative returns are positive.

You may use a previously defined sample object directly to set the workfile sample. Simply open a sample object by double clicking on the name or icon. You can set the workfile sample using the sample object, by entering the `smpl` command, followed by the sample object name. For example, the command:

```
smpl mysample
```

will set the workfile sample according to the rules contained in the sample object `MYSAMPLE`.

## 1.3 Eviews Functions

### 1.3.1 Operators

All of the operators described below may be used in expressions involving series and scalar values. When applied to a series expression, the operation is performed for each observation in the current sample.

Excellent Economics and Business programmes at:

 **university of groningen**



**“The perfect start of a successful, international career.”**

**CLICK HERE**  
to discover why both socially and academically the University of Groningen is one of the best places for a student to be

[www.rug.nl/feb/education](http://www.rug.nl/feb/education)

**Table 1.1: 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, $x^y$ , 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

### 1.3.2 Basic Mathematical Functions

The following functions perform basic mathematical operations. When applied to a series, they return a value for every observation in the current sample. When applied to a matrix object, they return a value for every element of the matrix object.

**Table 1.2: Mathematical Functions**

Function	Function Description
@abs(x)	absolute value @abs(-3)=3
@ceiling(x)	smallest integer not less than X, @ceiling(2.34)=3
@exp(x)	exponential, @exp(1)=2.71813
@floor(x)	largest integer not greater than X, @floor(1.23)=1
@iff(s,x,y)	returns X if condition S is true; otherwise returns Y
@inv(x)	reciprocal, @inv(2)=0.5 (For series or scalars only)
@log(x)	natural logarithm, @log(2)=0.693...
@log10(x)	base-10 logarithm
@logx(x,b)	base-b logarithm
@nan(x,y)	returns X if X<> NA, and Y if X=NA
@round(x)	rounds to the nearest integer @round(-97.5)=-98, @round(3.5)=4
@sqrt(x)	square root, @sqrt(9)=3

**Time Series Functions** The following functions facilitate working with time series data.

### 1.3.3 Statistical functions

These functions compute descriptive statistics for a specified sample, excluding missing values if necessary. The default sample is the current workfile sample. If you are performing these computations on a series and placing the results into a series,

**Table 1.3: Time Series Functions**

Function	Function Description
(-k)	k-lag operator
(+k)	k-lead operator
d(x)	first difference
d(x,n)	n-th order difference
d(x,n,s)	n-th order difference with a seasonal difference at S
dlog(x)	first difference of the logarithm
dlog(x,n)	n-th order difference of the logarithm
dlog(x,n,s)	n-th order difference of the logarithm with a seasonal difference at S

you can specify a sample as the last argument of the descriptive statistic function, either as a string (in double quotes) or using the name of a sample object.

### Statistical Functions

Function	Function Description
@cor(x,y[,s])	correlation between X and Y
@cov(x,y[,s])	covariance between X and Y
@inner(x,y[,s])	inner product of X and Y
@obs(x[,s])	number of non-missing observations for X in the current sample
@nas(x[,s])	number of missing observations for X in the current sample
@mean(x[,s])	average of the values in X
@median(x[,s])	computes the median of the X
@min(x[,s])	minimum of the values in X
@max(x[,s])	maximum of the values in X
@quantile(x,q[,s])	q-th quantile of the series X
@ranks(x[,o,t,s])	rank the ranking of each observation in X. The order of ranking is set using o: "a" (ascending - default) or "d" (descending). Ties are broken according to the setting of t: "i" (ignore), "f" (first), "l" (last), "a" (average - default), "r" randomize
@stdev(x[,s])	standard deviation of the values in X
@var(x[,s])	variance of the values in X
@skew(x[,s])	skewness of values in X
@kurt(x[,s])	kurtosis of values in X
@sum(x[,s])	sum of the values in X
@prod(x[,s])	product of the values in X
@sumsq(x[,s])	sum of the squares of the values in X
@cumsum(x[,s])	sum of the values in X from the start of the sample to the current observation
@cumprod(x[,s])	product of the values in X from the start of the sample to the current observation
@cummean(x[,s])	mean of the values in X from the start of the sample to the current observation
@cumstdev(x[,s])	standard deviation of the values in X from the start of the sample to the current observation
@cumvar(x[,s])	variance of the values in X from the start of the sample to the current observation
@cumsumsq(x[,s])	sum-of-squares of the values in X from the start of the sample to the current observation
@movsum(x,n)	n-period backward moving sum of X for the current and previous n-1 observations
@movav(x,n)	n-period backward moving average of X for the current and previous n-1 observations
@movstddev(x,n)	n-period backward moving standard deviation of X for the current and previous n-1 observations
@movvar(x,n)	n-period backward moving variance of X for the current and previous n-1 observations
@movcov(x,y,n)	n-period backwards moving covariance between X and Y of the current and previous n-1 observations
@movcor(x,y,n)	n-period backwards moving correlation between X and Y of the current and previous n-1 observations
@movsumsq(x,n)	n-period backwards sum-of-squares of X for the current and previous observations

### 1.3.4 Statistical Distribution Functions

The following set of functions gives you a possibility to compute and use within your analysis values of density functions, cumulative distribution, quantile functions, and random number generators for a variety of statistical distributions.

**Table 1.4: Statistical Distribution Functions**

This tables provides cumulative, density, quantile functions and the random number generator functions respectively for the following distributions

Distribution	Function Description
Beta $\beta(a, b)$	@cbeta(x,a,b), @dbeta(x,a,b), @qbeta(p,a,b), @rbeta(a,b)
Binomial $B(n, p)$	@cbinom(x,n,p), @dbinom(x,n,p), @qbinom(s,n,p), @rbinom(n,p)
Chi-square $\chi^2(v)$	@cchisq(x,v), @dchisq(x,v), @qchisq(p,v), @rchisq(v)
Exponential $E(m)$	@cexp(x,m), @dexp(x,m), @qexp(p,m), @rexp(m)
F-distribution $F(v1, v2)$	@cfdist(x,v1,v2), @dfdist(x,v1,v2), @qfdist(p,v1,v2), @rfdist(v1,v1)
Gamma $\Gamma(b, r)$	@cgamma(x,b,r), @dgamma(x,b,r), @qgamma(p,b,r), @rgamma(b,r)
Laplace	@claplace(x), @dlaplace(x), @qlaplace(x), @rlaplace
Log-normal $LN(m, s)$	@clognorm(x,m,s), @dlognorm(x,m,s), @qlognorm(p,m,s), @rlognorm(m,s)
Negative Binomial $NB(n, p)$	@cnegbin(x,n,p), @dnegbin(x,n,p), @qnegbin(s,n,p), @rnebin(n,p)
Normal $N(0, 1)$	@cnorm(x), @dnorm(x), @qnorm(p), @rnorm, nrnd
Poisson $P(m)$	@cpoisson(x,m), @dpoisson(x,m), @qpoisson(p,m), @rpoisson(m)
Pareto	@cpareto(x,k,a), @dpareto(x,k,a), @qpareto(p,k,a), @rpareto(k,a)
Student t-distribution $t(v)$	@ctdist(x,v), @dtdist(x,v), @qtdist(p,v), @rtdist(v)
Uniform $U(a, b)$	@cumif(x,a,b), @dunif(x,a,b), @qunif(p,a,b), @runif(a,b), rnd
Weibull $W(m, a)$	@cweib(x,m,a), @dweib(x,m,a), @qweib(p,m,a), @rweib(m,a)

## LIGS University

based in Hawaii, USA

is currently enrolling in the  
Interactive Online **BBA, MBA, MSc,**  
**DBA and PhD** programs:

- ▶ enroll **by October 31st, 2014** and
- ▶ **save up to 11%** on the tuition!
- ▶ pay in 10 installments / 2 years
- ▶ Interactive **Online education**
- ▶ visit [www.ligsuniversity.com](http://www.ligsuniversity.com) to find out more!

**Note:** LIGS University is not accredited by any nationally recognized accrediting agency listed by the US Secretary of Education. [More info here.](#)



## 1.4 Programming in Eviews

On addition to the interactive part of the EViews, where you use the menu commands, windows and graphical interface, you can use programming language to perform your analysis. There are two ways of using the EViews batch language – either enter and edit commands in the command window, or create programs. A program is simply a text file containing EViews commands. Each command in the program will be executed in the order that it appears in the program. Using programs allows you to use looping, conditioning and subroutine processing.

In order to create a program file in EViews, select *File/New/Program* from the main menu. EViews will open an untitled program window where you can enter your commands. You can save the program by clicking on the *Save* or *Save As* button. EViews will add the extension ".PRG" to the name you provide.

To load a program previously saved on disk, click on *File/Open/Program*, navigate to the appropriate directory, and click on the desired name. Alternatively, from the command line, you may type `open` followed by the full program name, including the file extension ".prg". If necessary, include the full path to the file. The entire name should be enclosed in quotations if necessary.

A program consists of a one or more lines of text. Since each line of a program corresponds to a single EViews command, simply enter the text for each command and terminate the line by pressing the **Enter** key.

There are several ways to execute a program. The easiest method is to execute your program by pushing the *Run* button on a program window. The *Run* dialog opens, where you can enter the program name and supply arguments. You may use the radio buttons to choose between *Verbose* and *Quiet* modes. In *verbose* mode, EViews sends messages to the status line and continuously updates the workfile window as objects are created and deleted. *Quiet* mode suppresses these updates, reducing the time spent writing to the screen.

By default, when EViews encounters an error, it will immediately terminate the program and display a message. If you enter a number into the *Maximum errors before halting* field, EViews will continue to execute the program until the maximum number of errors is reached (unless there is a serious error occurred).

You may also execute a program by entering the `run` command, followed by the name of the program file:

```
run mysp500 or run c:\eviews\myprog
```

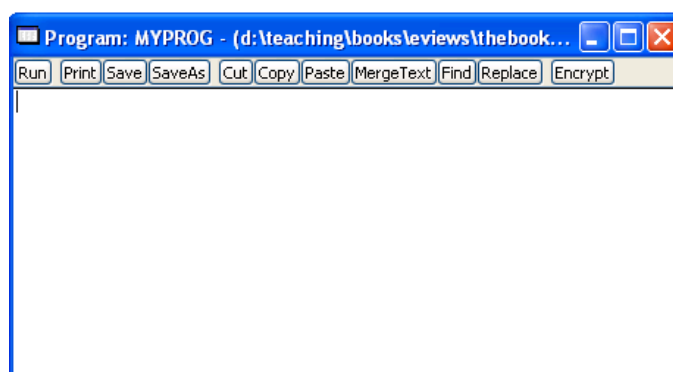
### Simple Programs

The simplest program is just a list of commands. Execution of the program is equivalent to typing the commands one by one into the command window. En-



tering commands in the program file has the advantage that you can save the set of commands for later use, and execute the program repeatedly, making minor modifications each time. Let us look at a simple example. Create a new program by typing program MYPROG in the command window. In the program window that opens for MYPROG, we are going to enter the commands to create a workfile, run a regression, compute residuals and a forecast, make a plot of the forecast, and save the results.

**Figure 1.4:** Program Window



### 1.4.1 Program Variables

**Control variables** are variables that you can use in place of numerical values in your EViews programs. Once a control variable is assigned a value, you can use it anywhere in a program that you would normally use a number. The name of a control variable starts with an "!" mark. After the "!", the name should be a legal EViews name of 15 characters or fewer. Examples of control variable names are: !q  
!1 !time

You do not need to declare control variables before you refer to them, though you must assign them a value before use. Control variables are assigned in the usual way, with the control variable name on the left of an "=" sign and a numerical value or expression on the right. For example:

```
!x = 7
```

```
!time = 12
```

Once assigned a value, a control variable may appear in an expression. For example:

```
!time = !time + 1
```

```
series returns = log(price) - log(price(-!q))
smpl 1950q1+!i 1960q4+!i
```

Control variables are automatically deleted after a program finishes. As a result, control variables are not saved when you save the workfile. You can save the values of control variables by creating new EViews objects which contain the values of the control variable. For example, the following command:

```
scalar numberx=!q
```

saves the numeric value assigned to the control variables !q into a scalar object numberx.

A **string variable** is a variable whose value is a string of text. A string expression or string is text enclosed in double quotes:

```
"cumulative returns"
"3.14159"
"ar(1) ar(2) ma(1) ma(2)"
```

.....Alcatel-Lucent 

[www.alcatel-lucent.com/careers](http://www.alcatel-lucent.com/careers)

What if you could build your future and create the future?

One generation's transformation is the next's status quo. In the near future, people may soon think it's strange that devices ever had to be "plugged in." To obtain that status, there needs to be "The Shift".



String variables, which only exist during the time that your program is executing, have names that begin with a "%" symbol. The following lines assign values to string variables:

```
%mtvar = "cumulative returns"
%armas = "ar(1) ar(2) ma(1) ma(2)"
%pi = " 3.14159"
```

You may use strings variables to build up command text, variable names, or other string values. EViews provides a number of operators and functions for manipulating strings. Once assigned a value, a string variable may appear in any expression in place of the underlying string. Here is a quick example where we use string operations to concatenate the contents of three string variables.

```
%str1 = "USD/GBP "
%str2 = "cumulative returns"
%st3 = %st1 + %st2
```

In this example %ST3 is set to the value "USD/GBP cumulative returns". String variables can be assigned to the table object for the output:

```
table1(1,1) = %st3
which is equivalent to entering the command
table(1,1) = "USD/GBP cumulative returns"
```

You can use a string variable to refer to a command, or a name, or portion of names indirectly. Suppose, for example, that we assign the string variable

```
%x = "usdgbp"
```

If you enclose a string variable in curly braces (" and ") EViews will replace the expression with the name or name fragment given by the string value. In this context we refer to the expression "%x" as a replacement variable since the string variable %x is replaced in the command line by the name or names of objects to which the string refers. For example, the program line

```
plot %x
```

would be interpreted by EViews as

```
plot usdgbp
```

Changing the contents of %x to "usdjpy" changes the interpretation of the original line to

```
plot usdjpy
```

since the replacement variable uses the name obtained from the new %x.

**Program arguments** are special string variables that are passed to your program when you run the program. Arguments allow you to change the value of string variables every time you run the program. You may use them in any context where a string variable is appropriate. Program arguments will be named %0, %1, %2, and so on. When you run a program that takes arguments, you will also supply the values for the arguments. If you use the Run button or **File/Run**, you will see a dialog box where you can type in the values of the arguments. If you use the run command, you should list the arguments consecutively after the name of the program. For example, suppose we have a program named RETS containing a command

```
series returns=log(%0)-log(%0(-1))
```

To run RETS from the command line with

```
%0 = "USDGBP", enter
```

```
run rets usdgbp
```

This program creates a time series returns using the usdgbp exchange rate defined or loaded previously in your workfile.

Alternatively, you can run this program by clicking on the *Run* button on the program window, or selecting *File/Run*. In the Run Program dialog box that appears, type the name of the program in the Program name or path field and enter the values of the arguments in the Program arguments field. Any arguments in your program that are not initialized in the run command or Run Program dialog are treated as blanks.

### IF Statements

There are many situations where you want to execute commands only if some condition is satisfied. EViews uses IF and ENDIF, or IF, ELSE, and ENDIF statements to indicate the condition to be met and the commands to be executed. An IF statement starts with the if keyword, followed by an expression for the condition, and then the word then. You may use AND/OR statements in the condition, using parentheses to group parts of the statement as necessary. If the expression is TRUE, all of the commands until the matching endif are executed. If the expression is FALSE, all of these commands are skipped. For example:

```
if !q = 3 then series returns = dlog(USDGBP) endif
```

```
if !time > 100 and !time < 200 then !age = 1/!time else !age = 0 endif
```

## The FOR Loop

The for loop allows you to repeat a set of commands for different values of a control or string variable. The FOR loop begins with a for statement and ends with a next statement. Any number of commands may appear between these two statements. The syntax of the FOR statement differs depending upon whether it uses control variables or string variables.

**FOR Loops with Control Variables** To repeat statements for different values of a control variable, the for statement involves setting a control variable equal to an initial value, followed by the word to, and then an ending value. After the ending value you may include the word step followed by a number indicating by how much to change the control variable each time the loop is executed. If you do not include step, the step is assumed to be 1. For example,

```
for !j=1 to 10
vector(10) weights(!j)=returns(!j)/stddev(!j)
next
```

The for loop is executed first for the initial value, unless that value is already beyond the terminal value. After it is executed for the initial value, the control variable is incremented by step and EViews compares the variable to the limit. If the limit is passed, execution stops.

**Maastricht University** *Leading in Learning!*

**Join the best at the Maastricht University School of Business and Economics!**

**Top master's programmes**

- 33<sup>rd</sup> place Financial Times worldwide ranking: MSc International Business
- 1<sup>st</sup> place: MSc International Business
- 1<sup>st</sup> place: MSc Financial Economics
- 2<sup>nd</sup> place: MSc Management of Learning
- 2<sup>nd</sup> place: MSc Economics
- 2<sup>nd</sup> place: MSc Econometrics and Operations Research
- 2<sup>nd</sup> place: MSc Global Supply Chain Management and Change

Sources: Keuzegids Master ranking 2013; Elsevier 'Beste Studies' ranking 2012; Financial Times Global Masters in Management ranking 2012

**Maastricht University is the best specialist university in the Netherlands (Elsevier)**

**Visit us and find out why we are the best!**  
**Master's Open Day: 22 February 2014**

[www.mastersopenday.nl](http://www.mastersopenday.nl)

One important use of FOR loops with control variables is to change the sample. If you add a control variable to a date in a `smpl` command, you will get a new date as many observations forward as the current value of the control variable. Here is a FOR loop that gradually increases the size of the sample and computes an average returns:

```
for !i=1 to 60
  smpl 1973m1 1974m1+!i
  scalar avret!i = @mean(returns) next
```

One other important case where you will use loops with control variables is in accessing elements of a series or matrix objects. For example,

```
!rows=@rows(vec1)
vector cumsum1=vec1
for !i=2 to !rows cumsum1(!i)=cumsum1(!i-1)+vec1(!i) next
```

computes the cumulative sum of the elements in the vector `vec1` and saves it in the vector `cumsum1`. To access an individual element of a series, you will need to use the `@elem` function and `@otod` to get the desired element

```
for !i=2 to !rows
  cumsum1(!i) = @elem(ser1, @otod(!i)) next
```

The `@otod` function returns the date associated with the observation index (counting from the beginning of the workfile), and the `@elem` function extracts the series element associated with a given date.

You can nest for loops to contain loops within loops. The entire inner for loop is executed for each successive value of the outer for loop. For example:

```
matrix(25,10) xx
for !i=1 to 25
  for !j=1 to 10
    xx(!i,!j)=(!i-1)*10+!j
  next
next
```

**FOR Loops with String Variables** When you wish to repeat statements for different values of a string variable, you can use the FOR loop to let a string variable

range over a list of string values. Give the name of the string variable followed by the list of values. For example,

```
for %y usdgbp usdjpy
series %yrets = dlog(%y) next
```

creates the returns series of two exchange rates

```
series usdgbpret = dlog(usdgbp)
series usdjpyret = dlog(usdjpy)
```

You can put multiple string variables in the same `for` statement – EViews will process the strings in sets.

For example:

```
for %y %z usdgbp usdjpy nzdusd audusd
equation e%y.ls %y c %z
next
```

In this case, the elements of the list are taken in groups of three. The loop is executed two times for the different sample pairs:

```
equation eusdgbp.ls usdgbp c usdjpy
equation eusdgbp.ls nzdusd c audusd
```

**The WHILE Loop** In some cases, we wish to repeat a series of commands several times, but only while one or more conditions are satisfied. Like the `FOR` loop, the `WHILE` loop allows you to repeat commands, but the `WHILE` loop provides greater flexibility in specifying the required conditions. The `WHILE` loop begins with a `while` statement and ends with a `wend` statement. Any number of commands may appear between the two statements. `WHILE` loops can be nested. The `WHILE` statement consists of the `while` keyword followed by an expression involving a control variable. The expression should have a logical (true or false) value or a numerical value. In the latter case, zero is considered false and any non-zero value is considered true. If the expression is true, the subsequent statements, up to the matching `wend`, will be executed, and then the procedure is repeated. If the condition is false, EViews will skip the following commands and continue on with the rest of the program following the `wend` statement. For example:

```
!val = 1
!a = 1
```

```

while !val<10000 and !a<10
smpl 1950q1 1970q1+!a
series inclval = income!/val
!val = !val*10 !a = !a+1 wend

```

Unlike a FOR statement, the WHILE statement does not update the control variable used in the test condition. You need to explicitly include a statement inside the loop that changes the control variable, or your loop will never terminate. Use the F1 key to break out of a program which is in an infinite loop.

**Subroutines** A subroutine is a collection of commands that allows you to perform a given task repeatedly, with minor variations, without actually duplicating the commands. You can also use subroutines from one program to perform the same task in other programs. A subroutine starts with the keyword subroutine followed by the name of the routine and any arguments, and ends with the keyword endsub. Any number of commands can appear in between. The simplest type of subroutine has the following form:



**> Apply now**

**REDEFINE YOUR FUTURE  
AXA GLOBAL GRADUATE  
PROGRAM 2015**

**redefining / standards**



agence ulg - © Photonstop



```

subroutine rets
series returns = dlog(price)
endsub

```

where the keyword `subroutine` is followed only by the name of the routine. This subroutine has no arguments so that it will behave identically every time it is used. It creates the log-return time series from the existing price levels `price`.

You can use the `return` command to force EViews to exit from the subroutine at any time. A common use of `return` is to exit from the subroutine if an unanticipated error is detected.

To define a subroutine with arguments, you start with `subroutine`, followed by the subroutine name, a left parenthesis, the arguments separated by commas, and finally a right parenthesis. Each argument is specified by listing a type of EViews object, followed by the name of the argument. Control variables may be passed by the scalar type and string variables by the string type. For example:

```

subroutine rets1(series r, series p, scalar lg)
series r = dlog(p, lg)
endsub

```

This subroutine generalizes the example subroutine `RETS`. Calling `RETS1` will fill the series given by the argument `R` with the log-returns of frequency `LG` from the series `P`. So if you set `R` equal to `RETURNS`, `P` equal to `PRICES`, and `LG` equal to `1`, you will get the equivalent of the subroutine `RETS` above.

*Subroutine call* Your subroutine definitions should be placed, in any order, at the beginning of your program. The subroutines are executed by the program using a call statement. For example:

```

subroutine rets
series returns = dlog(price)
endsub
' program execution
load mywork
fetch z
call rets

```

Execution of this program begins with the `load` statement. The subroutine definition is executed only at the last line when it is "called". Subroutines may call each

other, or even call themselves. Alternatively, you may wish to place frequently used subroutines in a separate program file and use an include statement to insert them at the beginning of your program. If, for example, you put the subroutine lines in the file RETURNS.PRG, then you may put the line:

```
include returns
```

at the top of any other program that needs to call RETS or RETS1. You can use the subroutines in these programs as though they were built-in parts of the EViews programming language.

If a subroutine has got arguments, it is executed by using the call keyword call which follows by the name of the subroutine and a list of any argument values you wish to use, enclosed in parentheses and separated by commas. All arguments must be provided in the same order as in the declaration statement. For example:

```
include rets1  
load mywork  
fetch z price  
series returns  
call rets1(returns, price, 3)
```


Subroutines work with variables and objects that are either global or local. Global variables refer either to objects which exist in the workfile when the subroutine is called, or to the objects that are created in the workfile by a subroutine. Global variables remain in the workfile when the subroutine finishes. A local variable is one that has meaning only within the subroutine. Local variables are deleted from the workfile once a subroutine finishes.

Global objects may be used and updated directly from within the subroutine. If, however, a global object has the same name as an argument in a subroutine, the variable name will refer to the argument and not to the global variable.

*Local Subroutines* All objects created by a global subroutine will be global and will remain in the workfile upon exit from the subroutine. If you include the word local in the definition of the subroutine, you create a local subroutine. All objects created by a local subroutine will be local and will be removed from the workfile upon exit from the subroutine. Local subroutines are most useful when you wish to write a subroutine which creates many temporary objects that you do not want to keep. You may not use or update global objects directly from within the subroutine. The global objects corresponding to arguments may be used and updated by referring to the arguments. All other objects in the subroutine are local

and will be deleted when the subroutine finishes. If you want to save results from a local subroutine, you have to explicitly include them in the arguments.

Local subroutines can call global subroutines and vice versa. The global subroutine will only have access to the global variables, and the local subroutine will only have access to the local variables, unless information is passed between the routines via arguments.



**Empowering People. Improving Business.**

BI Norwegian Business School is one of Europe's largest business schools welcoming more than 20,000 students. Our programmes provide a stimulating and multi-cultural learning environment with an international outlook ultimately providing students with professional skills to meet the increasing needs of businesses.

BI offers four different two-year, full-time Master of Science (MSc) programmes that are taught entirely in English and have been designed to provide professional skills to meet the increasing need of businesses. The MSc programmes provide a stimulating and multi-cultural learning environment to give you the best platform to launch into your career.

- MSc in Business
- MSc in Financial Economics
- MSc in Strategic Marketing Management
- MSc in Leadership and Organisational Psychology

[www.bi.edu/master](http://www.bi.edu/master)

**BI NORWEGIAN BUSINESS SCHOOL**

EFMD **EQUIS** ACCREDITED

