EViews 5 Command and Programming Reference

Quantitative Micro Software
Table of Contents

CHAPTER 1. INTRODUCTION ................................................................................. 1
  Using Commands .......................................................................................... 1
  Batch Program Use ...................................................................................... 2
  How to Use this Manual .............................................................................. 2

CHAPTER 2. OBJECT AND COMMAND BASICS ............................................... 5
  Object Declaration ....................................................................................... 5
  Object Commands ....................................................................................... 6
  Object Assignment ...................................................................................... 9
  More on Object Declaration ...................................................................... 9
  Auxiliary Commands ................................................................................ 10
  Managing Workfiles and Databases ............................................................ 11
  Managing Objects ...................................................................................... 13
  Basic Command Summary ....................................................................... 16

CHAPTER 3. MATRIX LANGUAGE ..................................................................... 23
  Declaring Matrices ................................................................................... 23
  Assigning Matrix Values .......................................................................... 24
  Copying Data Between Objects .................................................................. 27
  Matrix Expressions .................................................................................. 34
  Matrix Commands and Functions .............................................................. 37
  Matrix Views and Procs ............................................................................ 39
  Matrix Operations versus Loop Operations ............................................. 41
  Summary of Automatic Resizing of Matrix Objects ................................ 42
  Matrix Function and Command Summary .............................................. 44

CHAPTER 4. WORKING WITH TABLES ............................................................. 47
  Creating a Table ....................................................................................... 47
  Assigning Table Values ........................................................................... 48
  Customizing Tables .................................................................................. 50
  Labeling Tables ....................................................................................... 56
  Printing Tables ......................................................................................... 57
  Exporting Tables to Files .......................................................................... 57
  Customizing Spreadsheet Views ............................................................... 57
  Table Summary ......................................................................................... 58
## Table of Contents

### Chapter 5. Working with Graphs
- Creating a Graph ........................................ 59
- Changing Graph Types .................................. 63
- Customizing a Graph .................................... 64
- Labeling Graphs ........................................ 80
- Printing Graphs .......................................... 80
- Exporting Graphs to Files .............................. 81
- Graph Summary .......................................... 81

### Chapter 6. EViews Programming
- Program Basics ......................................... 83
- Simple Programs ....................................... 86
- Program Variables ..................................... 88
- Program Modes ......................................... 96
- Program Arguments .................................... 97
- Control of Execution ................................... 99
- Multiple Program Files ............................... 107
- Subroutines ............................................. 108
- Programming Summary ................................ 115

### Chapter 7. Sample Programs
- Descriptive Statistics by Year ....................... 117
- Rolling ADF Test ...................................... 119
- Calculating Cumulative Sums ....................... 121
- Time Series Operations on a Sample ............. 122
- Creating Dummy Variables with a Loop ........... 123
- Extracting Test Statistics in a Loop ............... 124
- Between Group Estimation for Pooled Data ....... 126
- Hausman Test for Fixed Versus Random Effects 128
- Regression Output Table ............................ 130

### Appendix A. Object, View and Procedure Reference
- Alpha ....................................................... 136
- Coef ....................................................... 137
- Equation ................................................ 139
- Graph ..................................................... 143
- Group ..................................................... 145
<table>
<thead>
<tr>
<th>Link</th>
<th>147</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logl</td>
<td>148</td>
</tr>
<tr>
<td>Matrix</td>
<td>150</td>
</tr>
<tr>
<td>Model</td>
<td>152</td>
</tr>
<tr>
<td>Pool</td>
<td>153</td>
</tr>
<tr>
<td>Rowvector</td>
<td>156</td>
</tr>
<tr>
<td>Sample</td>
<td>157</td>
</tr>
<tr>
<td>Scalar</td>
<td>158</td>
</tr>
<tr>
<td>Series</td>
<td>159</td>
</tr>
<tr>
<td>Sspace</td>
<td>161</td>
</tr>
<tr>
<td>Sym</td>
<td>164</td>
</tr>
<tr>
<td>System</td>
<td>166</td>
</tr>
<tr>
<td>Table</td>
<td>169</td>
</tr>
<tr>
<td>Text</td>
<td>170</td>
</tr>
<tr>
<td>Valmap</td>
<td>171</td>
</tr>
<tr>
<td>Var</td>
<td>171</td>
</tr>
<tr>
<td>Vector</td>
<td>174</td>
</tr>
</tbody>
</table>

**APPENDIX B. COMMAND REFERENCE** ................................................. 177

**APPENDIX C. SPECIAL EXPRESSION REFERENCE** .............................. 519

**APPENDIX D. MATRIX AND STRING REFERENCE** .............................. 527

**APPENDIX E. PROGRAMMING LANGUAGE REFERENCE** ....................... 559

**APPENDIX F. OPERATOR AND FUNCTION REFERENCE** ....................... 573

- Operators ................................................................. 573
- Date and Observation Functions ........................................... 574
- Basic Mathematical Functions ............................................ 575
- Time Series Functions .................................................... 576
- Descriptive Statistics .................................................... 577
- By-Group Statistics ....................................................... 579
- Additional and Special Functions ....................................... 580
- Trigonometric Functions .................................................. 583
- Statistical Distribution Functions ..................................... 584

**INDEX** ......................................................................................... 589
Chapter 1. Introduction

EViews provides you with both a Windows and a command line interface for working with your data. Almost every operation that can be accomplished using menus may also be entered into the command window, or placed in programs for batch processing. You are free to choose the mixture of techniques which best fits your particular style of work.

The *Command and Programming Reference (CPR)* documents the use of commands and programs to perform various tasks in EViews—the companion *User’s Guide* describes in greater detail the general features of EViews, with an emphasis on the interactive Windows interface.

In addition to providing a basic command reference, the *Command and Programming Reference* documents the use of EViews’ powerful batch processing language and advanced programming features. With EViews, you can create and store commands in programs that automate repetitive tasks, or generate a record of your research project.

Using Commands

Commands may be used *interactively* or executed in *batch* mode.

**Interactive Use**

The *command window* is located just below the main menu bar at the top of the EViews window. A blinking insertion cursor in the command window indicates that keyboard focus is in the command window and that keystrokes will be entered in the window at the insertion point. If no insertion cursor is present, simply click in the command window to change the focus.

To work interactively, you will type a command into the command window, then press ENTER to execute the command. If you enter an incomplete command, EViews will open a dialog box prompting you for additional information.

A command that you enter in the window will be executed as soon as you press ENTER. The insertion point need not be at the end of the command line when you press ENTER. EViews will execute the line that contains the insertion point.
When you enter a command, EViews will add it to the list of previously executed commands contained in the window. You can scroll up to an earlier command, edit it, and hit ENTER. The modified command will be executed. You may also use standard Windows copy-and-paste between the command window and any other window.

The contents of the command area may also be saved directly into a text file for later use. First make certain that the command window is active by clicking anywhere in the window, and then select **File/Save As...** from the main menu.

You can resize the command window so that a larger number of previously executed commands are visible. Use the mouse to move the cursor to the bottom of the window, hold down the mouse button, and drag the bottom of the window downwards.

We will point out that as you open and close object windows in EViews, the keyboard focus may change from the command window to the active window. If you then wish to enter a command, you will first need to click in the command window to set the focus. You can influence EViews’ method of choosing keyboard focus by changing the global defaults—simply select **Options/Window and Font Options...** from the main menu, and change the **Keyboard Focus** setting as desired.

**Batch Program Use**

You can assemble a number of commands into a *program*, and then execute the commands in batch mode. Each command in the program will be executed in the order that it appears in the program. Using batch programs allows you to make use of advanced capabilities such as looping and condition branching, and subroutine and macro processing. Programs also are an excellent way to document a research project since you will have a record of each step of the project.

One way to create a program file in EViews is to select **File/New/Program**. EViews will open an untitled program window into which you may enter your commands. You can save the program by clicking on the **Save** or **SaveAs** button, navigating to the desired directory, and entering a file name. EViews will append the extension .PRG to the name you provide.

Alternatively, you can use your favorite text (ASCII) editor to create a program file containing your commands. The commands in this program may then be executed from within EViews.

**How to Use this Manual**

• Chapter 2, “Object and Command Basics”, on page 5 explains the basics of using commands to work with EViews objects, and provides a cross-referenced listing of basic EViews commands associated with various tasks.

• Appendix A, “Object, View and Procedure Reference”, beginning on page 135 provides a cross-referenced listing of commands, views, and procedures associated with each object.

The other sections provide documentation of more advanced EViews features:

• Chapter 3, “Matrix Language”, on page 23 describes the EViews matrix language and provides a summary of the available matrix operators, functions, and commands.

• Chapter 4, “Working with Tables”, on page 47 documents the table object and describes the basics of working with strings in EViews.

• Chapter 5, “Working with Graphs”, on page 59 documents the table object and describes the basics of working with strings in EViews.

• Chapter 6, “EViews Programming”, on page 83 describes the basics of using programs for batch processing and documents the programming language.

• Chapter 7, “Sample Programs”, on page 117 contains annotated sample EViews programs to perform a variety of common tasks.

The remaining sections contain dictionary-style reference material for all of the EViews commands, functions, and operators:

• Appendix B, “Command Reference”, on page 177 provides a full alphabetized listing of basic commands, views and procedures. This material contains the primary reference material for working with EViews.

• Appendix C, “Special Expression Reference”, on page 519 special expressions that may be used in series assignment and generation, or as terms in estimation specifications.

• Appendix D, “Matrix and String Reference”, on page 527 is an alphabetical listing of the commands and functions associated with the EViews matrix language.

• Appendix E, “Programming Language Reference”, on page 559 contains an alphabetical listing of the keywords and functions associated with the EViews programming language.

• Appendix F, “Operator and Function Reference”, beginning on page 573 describes the operators and functions that may be used with series and (in some cases) matrix objects.
While this reference manual is not designed to be read from cover-to-cover, we recommend that before beginning extensive work using EViews commands, you spend some time with Chapter 2, “Object and Command Basics”, which describes the basics of using commands to work with objects. A solid understanding of this material is important for getting the most out of EViews.

If you wish to use programs in EViews, you should, at the very least, examine the first part of Chapter 6, “EViews Programming”, which describes the basics of creating, loading, and running a batch program.
Chapter 2. Object and Command Basics

This chapter provides an overview of the command method of working with EViews and EViews objects. If you are new to EViews, you may find it useful to consult the User's Guide (especially Chapter 1, “Introduction”, on page 11 and Chapter 4, “Object Basics”, beginning on page 59) for a more detailed introduction to EViews and a discussion of objects, their views, and procedures.

The command line interface of EViews is comprised of a set of single line commands, each of which may be classified as one of the following:

- object declarations.
- object commands.
- object assignment statements.
- auxiliary commands.

An EViews program is composed of a sequence of these commands, and may also contain the following:

- control variable assignment statements.
- program control statements.

The use of control variables and program control statements is discussed in detail in the programming guide in Chapter 6, “EViews Programming”, on page 83. The following sections provide an overview of the first four types of commands.

Object Declaration

The first step is to create or declare an object. A simple declaration has the form:

```
object_type object_name
```

where `object_name` is the name you would like to give to the new object and `object_type` is one of the following object types:

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha</td>
<td>136</td>
</tr>
<tr>
<td>Model</td>
<td>152</td>
</tr>
<tr>
<td>Sym</td>
<td>164</td>
</tr>
<tr>
<td>Coef</td>
<td>137</td>
</tr>
<tr>
<td>Pool</td>
<td>153</td>
</tr>
<tr>
<td>System</td>
<td>166</td>
</tr>
<tr>
<td>Equation</td>
<td>139</td>
</tr>
<tr>
<td>Rowvector</td>
<td>156</td>
</tr>
<tr>
<td>Table</td>
<td>169</td>
</tr>
<tr>
<td>Graph</td>
<td>143</td>
</tr>
<tr>
<td>Sample</td>
<td>157</td>
</tr>
<tr>
<td>Text</td>
<td>169</td>
</tr>
</tbody>
</table>
For example, the declaration:

series lgdp

creates a new series called LGDP, while the command

equation eq1

creates a new equation object called EQ1.

Matrix objects are typically declared with their dimension in parentheses after the object type. For example

matrix(5,5) x

creates a $5 \times 5$ matrix named X, while

coef(10) results

creates a 10 element coefficient vector named RESULTS.

Note that in order to create an object you must have a workfile currently open in EViews. You can open or create a workfile interactively from the File Menu, or you can use the load or workfile commands to perform the same operations inside a program. See Chapter 3, “Workfile Basics”, on page 39 of the User’s Guide for details.

Object Commands

An object command is a command which accesses an object’s views and procedures (procs). Object commands have two main parts, a display action followed by a view specification. The view specification describes the view or procedure of the object to be acted upon. The display action determines what is to be done with the output from the view or procedure.

The complete syntax for an object command has the form:

\[
\text{action(action_opt) object_name.view_or_proc(view_proc_opt) arg_list}
\]

where

- action ............... is one of the four verbs (do, freeze, print, show)
- action_opt ........... an option that modifies the default behavior of the action
object_name ......the name of the object to be acted upon
view_or_proc ...the object view or procedure to be acted upon
view_proc_opt ....an option that modifies the default behavior of the view or procedure
arg_list ...............a list of view or procedure arguments, generally separated by spaces

The four possible actions are:

- **do** executes procedures without opening a window. If the object's window is not currently displayed, no output is generated. If the object's window is already open, do is equivalent to show.
- **freeze** creates a table or graph from the object view window.
- **print** prints the object view window.
- **show** displays the object view in a window.

In most cases, some of the components of the general object command are not necessary since some views and procs do not require an argument list or options.

Furthermore, you need not explicitly specify an action. If no action is provided, the show action is assumed for views and the do action is assumed for procedures. For example, when using the command to display the series view for a line graph:

```
gdp.line
```

EViews implicitly adds a show command.

```
show gdp.line
```

Alternatively, for the equation procedure `ls`:

```
eq1.ls cons c gdp
```

there is an implicit do action.

```
do eq1.ls cons c gdp
```

In some cases, you may wish to modify the default behavior by explicitly describing the action. For example,

```
print eq1.ls cons c gdp
```

both performs the implicit do action and then sends the output from the proc to the printer.

Examples:
show gdp.line
print(l) group1.stats
freeze(output1) eq1.ls cons c gdp
do eq1.forecast eq1f

The first example opens a window displaying a line graph of the series GDP. The second example prints (in landscape mode) descriptive statistics for the series in GROUP1. The third example creates a table named OUTPUT1 from the estimation results of EQ1 for a least squares regression of CONS on GDP. The final example executes the forecast procedure of EQ1, putting the forecasted values into the series EQ1F and suppressing any procedure output.

Of these four examples, only the first opens a window and displays output on the screen.

Output Control

As discussed above, the display action determines the destination for view and procedure output. Here we note in passing a few extensions to these general rules.

You may request that a view be simultaneously printed and displayed on your screen by the letter “p” as an option to the object command. For example, the expression

gdp.correl(24,p)

is equivalent to the two commands,

show gdp.correl(24)
print gdp.correl(24)

since correl is a series view. The “p” option can be combined with other options, separated by commas. So as not to interfere with other option processing, we strongly recommend that the “p” option should always be specified after any required options.

Note that the print command accepts the “l” or “p” option to indicate landscape or portrait orientation. For example,

print(l) gdp.correl(24)

Printer output can be redirected to a text file or frozen output. See the output command in Chapter, and the discussion in Appendix B, “Global Options”, beginning on page 902 of the User’s Guide, for details.

The freeze command used without options creates an untitled graph or table from a view specification:
freeze gdp.line

You also may provide a name for the frozen object in parentheses after the word freeze. For example,

freeze(figure1) gdp.bar

names the frozen bar graph of GDP as “figure1”.

Object Assignment

Object assignment statements are commands which assign data to an EViews object. Object assignment statements have the syntax:

object_name = expression

where object_name identifies the object whose data is to be modified and expression is an expression which evaluates to an object of an appropriate type.

The nature of the assignment varies depending on what type of object is on the left hand side of the equal sign. To take a simple example, consider the assignment statement:

\[ x = 5 \times \log(y) + z \]

where X, Y and Z are series. This assignment statement will take the log of each element of Y, multiply each value by 5, add the corresponding element of Z, and, finally, assign the result into the appropriate element of X.

More on Object Declaration

Object declarations can often be combined with either object commands or object assignment statements to create and initialize an object in a single line. For example:

\[ \text{series lgdp} = \log(\text{gdp}) \]

creates a new series called LGDP and initializes its elements with the log of the series GDP. Similarly, the command

\[ \text{equation eq1.ls y c x1 x2} \]

creates a new equation object called EQ1 and initializes it with the results from regressing the series Y against a constant term, the series X1 and the series X2.

Additional examples,

\[ \text{scalar elas} = 2 \]
\[ \text{series tr58} = \text{@trend}(1958) \]
\[ \text{group nipa gdp cons inv g x} \]
equation cnsfnc2.ls log(cons)=c(1)+c(2)*yd
vector beta = @inverse(x*x)*(x*y)

An object can be declared multiple times so long as it is always declared to be of the same type. The first declaration will create the object, subsequent declarations will have no effect unless the subsequent declaration also specifies how the object is to be initialized. For example,

\begin{verbatim}
smpl @first 1979
series dummy = 1
smpl 1980 @last
series dummy=0
\end{verbatim}

creates a series named DUMMY that has the value 1 prior to 1980 and the value 0 thereafter.

Redeclaration of an object to a different type is not allowed and will generate an error.

Auxiliary Commands

Auxiliary commands are commands which are either unrelated to a particular object (i.e., not views or procs), or which act on an object in a way that is generally independent of the type or contents of the object. Auxiliary commands typically follow the syntax:

\begin{verbatim}
command(option_list) argument_list
\end{verbatim}

where command is the name of the view or procedure to be executed, option_list is a list of options separated by commas, and argument_list is a list of arguments generally separated by spaces.

An example of an auxiliary command is

\begin{verbatim}
store(d=c:\newdata\db1) gdp m x
\end{verbatim}

which will store the three objects GDP, M and X in the database named DB1 in the directory C:\NEWDATA.

There is also a set of auxiliary commands which are designed to be used interactively, and sometimes performs operations that create new untitled or unnamed objects (see “Interactive Use Commands” on page 20). For example, the command:

\begin{verbatim}
ls y c x1 x2
\end{verbatim}

will regress the series Y against a constant term, the series X1 and the series X2, and create a new untitled equation object to hold the results.
Since these commands are designed primarily for interactive use they will prove useful for carrying out simple tasks. Overuse of these interactive tools, or their use in programs, will make it difficult to manage your work since unnamed objects cannot be referenced by name from within a program, cannot be saved to disk, and cannot be deleted except through the graphical Windows interface. Wherever possible, you should use named rather than untitled objects for your work. For example, we may replace the above auxiliary command with the statement:

\[
\text{equation } eq1.ls \ y \ c \ x1 \ x2
\]

To create the named equation object EQ1. This example uses declaration of the object EQ1 and the equation method \( ls \) to perform the same task as the auxiliary command above.

### Managing Workfiles and Databases

There are two types of object containers: **workfiles** and **databases**. All EViews objects must be held in an object container, so before you begin working with objects you must create a workfile or database. Workfiles and databases are described in depth in Chapter 3, “Workfile Basics”, on page 39 and Chapter 10, “EViews Databases”, beginning on page 241 of the User’s Guide.

#### Managing Workfiles

To declare and create a new workfile, follow the `workfile` command with a name for the workfile, an option for the frequency of the workfile, and the start and end dates. The workfile frequency type options are:

- a  annual.
- s  semi-annual.
- q  quarterly.
- m  monthly.
- w  weekly.
- d  daily (5 day week).
- 7  daily (7 day week).
- u  undated.

For example,

\[
\text{workfile } macro1 \ q \ 1965:1 \ 1995:4
\]

creates a new quarterly workfile named MACRO1 from the first quarter of 1965 to the fourth quarter of 1995.
workfile cps88 u 1 1000

creates a new undated workfile named CPS88 with 1000 observations.

Note that if you have multiple workfiles open, the workfile command may be used to change the active workfile.

To save your workfile, type “save” followed by a workfile name. If any part of the path or workfile name has spaces, you should enclose the entire expression in quotation marks. The active workfile will be saved in the default path under the given name. You may optionally provide a path to save the workfile in a different directory:

```
save a:\mywork
```

If necessary, enclose the path name in quotations. To close the workfile, type “close” followed by the workfile name. For example,

```
close mywork
```

closes the workfile window of MYWORK.

To open a previously saved workfile, follow the load keyword with the name of the workfile. You can optionally include a path designation to open workfiles that are not saved in the default path. For example,

```
load "c:\mywork\proj1"
```

You may also use the open command to open a previously saved workfile. To use open, you have to type the full workfile name including the extension .WK1:

```
open proj2.wk1
```

Managing Databases

To create a new database, follow the dbcreate keyword with a name for the new database. Alternatively, you could use the db keyword followed by a name for the new database. The two commands differ only when the named database already exists. If you use dbcreate and the named database already exists on disk, EViews will error indicating that the database already exists. If you use db and the named database already exists on disk, EViews will simply open the existing database. Note that the newly opened database (either by dbcreate or db) will become the default database.

For example,

```
dbcreate mydata1
```

creates a new database named MYDATA1 in the default path, opens a new database window, and makes MYDATA1 the default database.
Managing Objects—13

`db c:\evdata\usdb`

opens the USDB database in the specified directory if it already exists. If it does not,
EViews creates a new database named USDB, opens its window, and makes it the default
database.

You can also use `dbopen` to open an existing database and to make it the default database. For example,

`dbopen findat`

opens the database named FINDAT in the default directory. If the database does not exist,
EViews will error indicating that the specified database cannot be found. You can also use
`open` to open an existing database. To use `open`, you must provide the full name of the
database, including the file extension `.EDB`:

`open findat.edb`

You may use `dbrename` to rename an existing database. Follow the `dbrename` keyword by
the current (old) name and a new name.

`dbrename temp1 newmacro`

To delete an existing database, use the `dbdelete` command. Follow `dbdelete` by the
name of the database to delete.

`dbdelete c:\data\usmacro`

`dbcopy` makes a copy of the existing database. Follow `dbcopy` by the name of the source
file and the name of the destination file.

`dbcopy c:\evdata\macro1 a:\macro1`

`dbpack`, `dbrepair`, and `dbrebuild` are database maintenance commands. See also
Chapter 10, “EViews Databases”, beginning on page 241 of the User’s Guide for a detailed
description.

Managing Objects

In the course of a program you will often need to manage the objects in a workfile by copy-
ing, renaming, deleting and storing them to disk. EViews provides a number of auxiliary
commands which perform these operations. The following discussion introduces you to
the use of these commands; a full description of each command is provided in Chapter ,
"", beginning on page 177.
Copying Objects

You can create a duplicate copy of an object using the `copy` command. The `copy` command is an auxiliary command with the format:

```
copy source_name dest_name
```

where `source_name` is the name of the object you wish to duplicate, and `dest_name` is the name you want attached to the new copy of the object.

The `copy` command may also be used to copy objects in databases and to move objects between workfiles and databases.

Copy with Wildcard Characters

EViews supports the use of wildcard characters ("?" for a single character match and "*" for a pattern match) in destination specifications when using the `copy` and `rename` commands. Using this feature, you can copy or rename a set of objects whose names share a common pattern in a single operation. This can be useful for managing series produced by model simulations, series corresponding to pool cross-sections, and any other situation where you have a set of objects which share a common naming convention.

A destination wildcard pattern can be used only when a wildcard pattern has been provided for the source, and the destination pattern must always conform to the source pattern in that the number and order of wildcard characters must be exactly the same between the two. For example, the following patterns

<table>
<thead>
<tr>
<th>Source Pattern</th>
<th>Destination Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>x*</td>
<td>y*</td>
</tr>
<tr>
<td>*c</td>
<td>b*</td>
</tr>
<tr>
<td>x*?</td>
<td>yz*fabc</td>
</tr>
</tbody>
</table>

conform to each other, while these patterns do not

<table>
<thead>
<tr>
<th>Source Pattern</th>
<th>Destination Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>a*</td>
<td>b</td>
</tr>
<tr>
<td>*x</td>
<td>?y</td>
</tr>
<tr>
<td>x<em>y</em></td>
<td><em>x</em>y*</td>
</tr>
</tbody>
</table>

When using wildcards, the destination name is formed by replacing each wildcard in the destination pattern by the characters from the source name that matched the corresponding wildcard in the source pattern. Some examples should make this principle clear:
Managing Objects

Note, as shown in the second example, that a simple asterisk for the destination pattern does not mean to use the unaltered source name as the destination name. To copy objects between containers preserving the existing name, either repeat the source pattern as the destination pattern

\[ \text{copy } x* \text{ db1::x*} \]

or omit the destination pattern entirely

\[ \text{copy } x* \text{ db1::} \]

If you use wildcard characters in the source name and give a destination name without a wildcard character, EViews will keep overwriting all objects which match the source pattern to the name given as destination.

For additional discussion of wildcards, see Appendix C, “Wildcards”, on page 907 of the User’s Guide.

Renaming Objects

You can give an object a different name using the rename command. The rename command has the format:

\[ \text{rename source_name dest_name} \]

where source_name is the original name of the object and dest_name is the new name you would like to give to the object.

rename can also be used to rename objects in databases.

You may use wildcards when renaming series. The name substitution rules are identical to those described above for copy.

Deleting Objects

Objects may be removed from the workfile using the delete command. The delete command has the format:
delete name_pattern

where name_pattern can either be a simple name such as “XYZ”, or a pattern containing the wildcard characters “?” and “*”, where “?” means to match any one character, and “*” means to match zero or more characters. When a pattern is provided, all objects in the workfile with names matching the pattern will be deleted. Appendix C, “Wildcards”, on page 907 of the User's Guide describes further the use of wildcards.

delete can also be used to remove objects from databases.

Saving Objects

All named objects will be saved automatically in the workfile when the workfile is saved to disk. You can store and retrieve the current workfile to and from disk using the save and load commands. Unnamed objects will not be saved as part of the workfile.

You can also save objects for later use by storing them in a database. The store command has the format:

store(option_list) object1 object2 ...

where object1, object2, ..., are the names of the objects you would like to store in the database. If no options are provided, the series will be stored in the current default database (see Chapter 10 of the User's Guide for a discussion of the default database). You can store objects into a particular database by using the option “d=db_name” or by prepending the object name with a database name followed by a double colon “::”, such as:

store db1::x db2::x

Fetch Objects

You can retrieve objects from a database using the fetch command. The fetch command has the same format as the store command:

fetch(option_list) object1 object2 ...

To specify a particular database use the “d=” option or the “::” extension as for store.

Basic Command Summary

The following list summarizes the EViews basic commands. The full descriptions of these commands are given in Appendix B, “Command Reference”, beginning on page 177.

A list of views and procedures available for each object may be found in the next chapter. Commands for working with matrix objects are listed in Chapter 3, “Matrix Language”, on page 23, and EViews programming expressions are described in Chapter 6, “EViews Programming”, beginning on page 83.
Command Actions

do........................execute action without opening window (p. 252).
freeze.....................create view object (p. 275).
print......................print view (p. 373).
show......................show object window (p. 432).

Global Commands

cd......................change default directory (p. 213).
exit........................exit the EViews program (p. 264).
output..................redirect printer output (p. 347).
param....................set parameter values (p. 368).
rndseed..................set the seed of the random number generator (p. 389).
smpl.....................set current workfile sample (p. 436).
setconvert..............set the default frequency conversion method (p. 410).

Object Creation Commands

alpha.....................alpha series (p. 185).
coef........................coefficient vector (p. 223).
equation..................equation object (p. 260).
frml........................numeric or alpha series object with a formula for auto-updating
(p. 278).
genr........................numeric or alpha series object (p. 281).
graph....................graph object—create using a graph command or by merging existing
graphs (p. 288).
group.....................group object (p. 290).
link.....................series or alpha link object (p. 309).
logl.....................likelihood object (p. 314).
matrix....................matrix object (p. 335).
model....................model object (p. 339).
pool......................pool object (p. 372).
rowvector..................rowvector object (p. 390).
sample....................sample object (p. 392).
scalar.....................scalar object (p. 396).
series....................numeric series (p. 405).
sspace.....................sspace object (p. 444).
sym.....................sym object (p. 458).
system.....................system object (p. 459).
table .................. table object (p. 444).
text .................. text object (p. 469).
valmap ................. valmap object (p. 487).
var ....................... var estimation object (p. 487).
vector .................. vector object (p. 487).

Object Container, Data, and File Commands
ccopy .................. copy series from DRI database (p. 213).
cfetch ................... fetch series from DRI database (p. 218).
clabel .................. display DRI series description (p. 221).
close .................... close object, program, or workfile (p. 222).
create ................... create a new workfile (p. 235).
db ....................... open or create a database (p. 239).
dbcopy ................... make copy of a database (p. 240).
dbcreate .................. create a new database (p. 240).
dbdelete .................. delete a database (p. 241).
dbopen .................. open a database (p. 242).
dbpack ................... pack a database (p. 243).
dbrebuild ............. rebuild a database (p. 243).
dbrename ............. rename a database (p. 244).
dbrepair ........ repair a database (p. 245).
driconvert ............ convert the entire DRI database to an EViews database (p. 254).
expand .................. expand workfile range (p. 264).
fetch ................. fetch objects from databases or databank files (p. 264).
hconvert ........... convert an entire Haver Analytics database to an EViews database (p. 290).
hfetch .................. fetch series from a Haver Analytics database (p. 291).
hlabel .................. obtain label from a Haver Analytics database (p. 294).
load ..................... load a workfile (p. 313).
open .................... open a program or text (ASCII) file (p. 342).
pageappend .......... append observations to workfile page (p. 350).
pagecontract ........ contract workfile page (p. 352).
pagecopy ............. copy contents of a workfile page (p. 353).
pagerecreate .......... create a workfile page (p. 355).
pagedelete .......... delete a workfile page (p. 356).
pageload ............ load one or more pages into a workfile from a workfile or a foreign data source (p. 357).
pagerename ..........rename a workfile page (p. 358).
pagesave ..........save page into a workfile or a foreign data source (p. 358).
pageselect ..........make specified page active (p. 360).
pagetop .............reshape the workfile page by stacking observations (p. 360).
pagestruct ........apply a workfile structure to the page (p. 363).
pageunstack ..........reshape the workfile page by unstacking observations into multi-
range .................reset the workfile range (p. 377).
save ......................save workfile to disk (p. 393).
sort ....................sort the workfile (p. 440).
unlink ...............open a file (p. 342).
wfcreate ........create a new workfile (p. 491).
wfopen ..........open workfile or foreign source data as a workfile (p. 492).
wfsave ..............save workfile to disk as a workfile or a foreign data source 
wfselect ..........change active workfile page (p. 495).
workfile ........create or change active workfile (p. 498).
write ....................write series to a disk file (p. 498).

Object Utility Commands

close ..................close window of an object, program, or workfile (p. 222).

Object Assignment Commands

data ....................enter data from keyboard (p. 236).
frml ....................assign formula for auto-updating to a numeric or alpha series 
goals ...................create numeric or alpha series object (p. 281).
gend ....................assign random integer values to object (p. 510).
rndseed ................set random number generator seed (p. 389).

Graph Commands

area ....................area graph (p. 191).
bar ......................bar graph (p. 199).
errbar ................ error bar graph (p. 261).
hilo .....................high-low(-open-close) graph (p. 292).
20—Chapter 2. Object and Command Basics

line .................. line-symbol graph (p. 306).
pie .................... pie chart (p. 370).
scat .................... scatterplot (p. 399).
spike ................... spike graph (p. 442).
xy ...................... XY graph with one or more X plotted against one or more Y (p. 510).
xyline ................. XY line graph (p. 512).
xypair ................. XY pairs graph (p. 510).

Programming Commands
open .................... open a program file (p. 342).
output .................. redirects print output to objects or files (p. 347).
poff ...................... turns off automatic printing in programs (p. 565).
pon ...................... turns on automatic printing in programs (p. 565).
program ................ create a new program (p. 374).
run ...................... read data from a foreign disk file (p. 377).
statusline ............. open a file (p. 342).
tic ...................... reset the timer (p. 469).
toc ...................... display elapsed time (since timer reset) in seconds (p. 470).

Table Commands
setcell ................ format and fill in a table cell (p. 408).
setcolwidth .......... set width of a table column (p. 410).
setline ................ place a horizontal line in table (p. 426).

Interactive Use Commands
arch .................... estimate autoregressive conditional heteroskedasticity (ARCH and GARCH) models (p. 187).
archtest ................ LM test for the presence of ARCH in the residuals (p. 187).
auto ..................... Breusch-Godfrey serial correlation Lagrange Multiplier (LM) test (p. 187).
binary ................... binary dependent variable models (includes probit, logit, gompit) models (p. 187).
cause ................... pairwise Granger causality tests (p. 187).
censored ............... estimate censored and truncated regression (includes tobit) models (p. 187).
chow ................... Chow breakpoint and forecast tests for structural change (p. 187).
coint .................... Johansen cointegration test (p. 187).
Basic Command Summary—21

- cor: correlation matrix (p. 187).
- count: count data modeling (includes poisson, negative binomial and quasi-maximum likelihood count models) (p. 187).
- cov: covariance matrix (p. 187).
- cross: cross correlogram (p. 187).
- fit: static forecast from an equation (p. 187).
- forecast: dynamic forecast from an equation (p. 187).
- hist: histogram and descriptive statistics (p. 510).
- hpf: Hodrick-Prescott filter (p. 306).
- logit: logit (binary) estimation (p. 314).
- ls: linear and nonlinear least squares regression (includes weighted least squares and ARMAX) (p. 315).
- ordered: ordinal dependent variable models (includes ordered probit, ordered logit, and ordered extreme value models) (p. 345).
- probit: probit (binary) estimation (p. 374).
- reset: Ramsey’s RESET test for functional form (p. 383).
- seas: seasonal adjustment for quarterly and monthly time series (p. 403).
- smooth: exponential smoothing (p. 434).
- solve: solve a model (p. 438).
- stats: descriptive statistics (p. 449).
- testadd: likelihood ratio test for adding variables to equation (p. 461).
- testdrop: likelihood ratio test for dropping variables from equation (p. 464).
- tsls: linear and nonlinear two-stage least squares (TSLS) regression models (includes weighted TSLS, and TSLS with ARMA errors) (p. 474).
Chapter 3. Matrix Language

EViews provides you with tools for working directly with data contained in matrices and vectors. You can use the EViews matrix language to perform calculations that are not available using the built-in views and procedures.

The following objects can be created and manipulated using the matrix command language:

- \textit{coef}: column vector of coefficients to be used by equation, system, pool, logl, and sspace objects
- \textit{matrix}: two-dimensional array
- \textit{rowvector}: row vector
- \textit{scalar}: scalar
- \textit{sym}: symmetric matrix (stored in lower triangular form)
- \textit{vector}: column vector

We term these objects \textit{matrix objects} (despite the fact that some of these objects are not matrices).

Declaring Matrices

You must declare matrix objects prior to use. Detailed descriptions of declaration statements for the various matrix objects are provided in Chapter, “”, beginning on page 177.

Briefly, a declaration consists of the object \textit{keyword}, followed either by size information in parentheses and the name to be given to the object, followed (optionally) by an assignment statement. If no assignment is provided, the object will be initialized to have all zero values.

The various matrix objects require different sizing information. A matrix requires the number of rows and the number of columns. A \textit{sym} requires that you specify a single number representing both the number of rows and the number of columns. A \textit{vector}, \textit{rowvector}, or \textit{coef} declaration can include information about the number of elements. A \textit{scalar} requires no size information. If size information is not provided, EViews will assume that there is only one element in the object.

For example:

\begin{verbatim}
  matrix(3,10) xdata
  sym(9) moments
\end{verbatim}
vector(11) betas
rowvector(5) xob
creates a $3 \times 10$ matrix XDATA, a symmetric $9 \times 9$ matrix MOMENTS, an $11 \times 1$ column vector BETAS, and a $1 \times 5$ rowvector XOB. All of these objects are initialized to zero.

To change the size of a matrix object, you can repeat the declaration statement. Furthermore, if you use an assignment statement with an existing matrix object, the target will be resized as necessary. For example:

\[
\begin{align*}
\text{sym}(10) & \quad \text{bigz} \\
\text{matrix} & \quad \text{zdata} \\
\text{matrix}(10,2) & \quad \text{zdata} \\
\text{zdata} & = \text{bigz}
\end{align*}
\]

will first declare ZDATA to be a matrix with a single element, and then redeclare ZDATA to be a $10 \times 2$ matrix. The assignment statement in the last line will resize ZDATA so that it contains the contents of the $10 \times 10$ symmetric matrix BIGZ.

Assigning Matrix Values

There are three ways to assign values to the elements of a matrix: you may assign values to specific matrix elements, you may fill the matrix using a list of values, or you may perform matrix assignment.

Element assignment

The most basic method of assigning matrix values is to assign a value for a specific row and column element of the matrix. Simply enter the matrix name, followed by the row and column indices, in parentheses, and then an assignment to a scalar value.

For example, suppose we declare the $2 \times 2$ matrix $A$:

\[
\text{matrix}(2,2) \quad a
\]

The first command creates and initializes the $2 \times 2$ matrix $A$ so that it contains all zeros. Then after entering the two commands:

\[
\begin{align*}
a(1,1) & = 1 \\
a(2,1) & = 4
\end{align*}
\]

we have

\[
A = \begin{bmatrix} 1 & 0 \\ 4 & 0 \end{bmatrix}.
\]
You can perform a large number of element assignments by placing them inside of programming loops:

```plaintext
vector(10) y
matrix (10,10) x
for !i = 1 to 10
    y(!i) = !i
for !j = 1 to 10
    x(!i,!j) = !i + !j
next
next
```

Note that the `fill` procedure provides an alternative to using loops for assignment.

### Fill assignment

The second assignment method is to use the `fill` procedure to assign a list of numbers to each element of the matrix in the specified order. By default, the procedure fills the matrix column by column, but you may override this behavior.

You should enter the name of the matrix object, followed by a period, the `fill` keyword, and then a `comma delimited` list of values. For example, the commands:

```plaintext
vector(3) v
v1.fill 0.1, 0.2, 0.3
matrix(2,4) x
matrix.fill 1, 2, 3, 4, 5, 6, 7, 8
```

create the matrix objects

\[ V = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \end{bmatrix}, \quad X = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \end{bmatrix} \]  \hspace{1cm} (3.2)

If we replace the last line with

```plaintext
matrix.fill(b=r) 1,2,3,4,5,6,7,8
```

then \( X \) is given by

\[ X = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}. \]  \hspace{1cm} (3.3)
In some situations, you may wish to repeat the assignment over a list of values. You may use the “l” option to fill the matrix by repeatedly looping through the listed numbers until the matrix elements are exhausted. Thus,

\[
\text{matrix}(3,3) \ y \\
y.\text{fill}(l) \ 1, \ 0, \ -1
\]

creates the matrix

\[
Y = \begin{bmatrix} 
1 & 1 & 1 \\
0 & 0 & 0 \\
-1 & -1 & -1 \\
\end{bmatrix}
\]  

(3.4)

See fill (p. 267) for a complete description of the fill procedure.

Matrix assignment

You can copy data from one matrix object into another using assignment statements. To perform an assignment, you should enter the name of the target matrix followed by the equal sign “=”, and then a matrix object expression. The expression on the right-hand side should either be a numerical constant, a matrix object, or an expression that returns a matrix object.

There are a variety of rules for how EViews performs the assignment that depend upon the types of objects involved in the assignment.

Scalar values on the right-hand side

If there is a scalar on the right-hand side of the assignment, every element of the matrix object is assigned the value of the scalar.

Examples:

\[
\text{matrix}(5,8) \ \text{first} \\
\text{scalar} \ \text{second} \\
\text{vec}(10) \ \text{third} \\
\text{first} = 5 \\
\text{second} = c(2) \\
\text{third} = \text{first}(3,5)
\]

Since declaration statements allow for initialization, you can combine the declaration and assignment statements. Examples:

\[
\text{matrix}(5,8) \ \text{first} = 5 \\
\text{scalar} \ \text{second} = c(2) \\
\text{vec}(10) \ \text{third} = \text{first}(3,5)
\]
Same object type on right-hand side
If the source object on the right is a matrix or vector, and the target or destination object on the left is of the same type, the target will be resized to have the same dimension as the source, and every source element will be copied. For example:

```eviews
matrix(10,2) zdata = 5
matrix ydata = zdata
matrix(10,10) xdata = ydata
```
declares that ZDATA is a $10 \times 2$ matrix filled with 5's. YDATA is automatically resized to be a $10 \times 2$ matrix and is filled with the contents of ZDATA.

Note that even though the declaration of XDATA calls for a $10 \times 10$ matrix, XDATA is a $10 \times 2$ matrix of 5's. This behavior occurs because the declaration statement above is equivalent to issuing the two commands,

```eviews
matrix(10,10) xdata
xdata = ydata
```
which will first declare the $10 \times 10$ matrix XDATA, and then automatically resize it to $10 \times 2$ when you fill it with the values for YDATA.

The next section discusses assignment statements in the more general case, where you are converting between object types. In some cases, the conversion is automatic; in other cases, EViews provides you with additional tools to perform the conversion.

Copying Data Between Objects
In addition to the basic assignment statements described in the previous section, EViews provides you with a large set of tools for copying data to and from matrix objects.

At times, you may wish to move data between different types of matrix objects. For example, you may wish to take the data from a vector and put it in a matrix. EViews has a number of built-in rules which make these conversions automatically.

At other times, you may wish to move data between a matrix object and an EViews series or group object. There are a separate set of tools which allow you to convert data across a variety of object types.

Copying data from matrix objects
Data may be moved between different types of matrix objects using assignments. If possible, EViews will resize the target object so that it contains the same information as the object on the right side of the equation.

The basic rules governing expressions of the form “$Y = X$” may be summarized as follows:
Object type of Y does not change.

The target object Y will, if possible, be resized to match the object X; otherwise, EViews will issue an error. Thus, assigning a vector to a matrix will resize the matrix, but assigning a matrix to a vector will generate an error if the matrix has more than one column.

The data in X will be copied to Y.

Specific exceptions to the rules given above are:

- If X is a scalar, Y will keep its original size and will be filled with the value of X.
- If X and Y are both vector or rowvector objects, Y will be changed to the same type as X.

“Summary of Automatic Resizing of Matrix Objects” on page 42 contains a complete summary of the conversion rules for matrix objects.

Here are some simple examples illustrating the rules for matrix assignment:

```plaintext
vector(3) x
x(1) = 1
x(2) = 2
x(3) = 3
vector y = x
matrix z = x
```

Y is now a 3 element vector because it has the same dimension and values as X. EViews automatically resizes the Z Matrix to conform to the dimensions of X so that Z is now a $3 \times 1$ matrix containing the contents of X: $Z(1,1) = 1$, $Z(2,1) = 2$, $Z(3,1) = 3$.

Here are some further examples where automatic resizing is allowed:

```plaintext
vector(7) y = 2
scalar value = 4
matrix(10,10) w = value
w = y
matrix(2,3) x = 1
rowvector(10) t = 100
x = t
```

W is declared as a $10 \times 10$ matrix of 4’s, but it is then reset to be a $7 \times 1$ matrix of 2’s. X is a $1 \times 10$ matrix of 100’s.

Lastly, consider the commands
vector(7) y = 2
rowvector(12) z = 3
coef(20) beta
y = z
z = beta

Y will be a rowvector of length 3, containing the original contents of Z, and Z will be a column vector of length 20 containing the contents of BETA.

There are some cases where EViews will be unable to perform the specified assignment because the resize operation is not defined. For example, suppose that X is a 2 × 2 matrix. Then the assignment statement

vector(7) y = x

will result in an error. EViews cannot change Y from a vector to a matrix and there is no way to assign the 4 elements of the matrix X to the vector Y. Other examples of invalid assignment statements involve assigning matrix objects to scalars or syms to vector objects.

Copying data from parts of matrix objects

In addition to the standard rules for conversion of data between objects, EViews provides functions for extracting and assigning parts of matrix objects. Matrix functions are described in greater detail later in this chapter. For now, note that some functions take a matrix object and perhaps other parameters as arguments and return a matrix object.

A comprehensive list of the EViews commands and functions that may be used for matrix object conversion appears in “Utility Functions and Commands” on page 44. However, a few examples will provide you with a sense of the type of operations that may be performed.

Suppose first that you are interested in copying data from a matrix into a vector. The following commands will copy data from M1 and SYM1 into the vectors V1, V2, V3, and V4.

matrix(10, 10) m1
sym(10) sym1
vector v1 = @vec(m1)
vector v2 = @columnextract(m1,3)
vector v3 = @rowextract(m1,4)
vector v4 = @columnextract(sym1,5)

The @vec function creates a 100 element vector, V1, from the columns of M1 stacked one on top of another. V2 will be a 10 element vector containing the contents of the third col-


umn of M1 while V3 will be a 10 element vector containing the fourth row of M1. The \@vec, \@rowextract, and \@columnextract functions also work with sym objects. V4 is a 10 element vector containing the fifth column of SYM1.

You can also copy data from one matrix into a smaller matrix using \@subextract. For example:

\begin{verbatim}
matrix(20,20) m1=1
matrix m2 = \@subextract(m1,5,5,10,7)
matrix m3 = \@subextract(m1,5,10)
matrix m4 = m1
\end{verbatim}

M2 is a 6 \times 3 matrix containing a submatrix of M1 defined by taking the part of the matrix M1 beginning at row 5 and column 5 and ending at row 10 and column 7. M3 is the 16 \times 11 matrix taken from M1 at row 5 and column 10 to the last element of the matrix (row 20 and column 20). In contrast, M4 is defined to be an exact copy of the full 20 \times 20 matrix.

Data from a matrix may be copied into another matrix object using the commands colplace, rowplace, and matplace. Consider the commands:

\begin{verbatim}
matrix(100,5) m1 = 0
matrix(100,2) m2 = 1
vector(100) v1 = 3
rowvector(100) v2 = 4
matplace(m1,m2,1,3)
colplace(m1,v1,3)
rowplace(m1,v2,80)
\end{verbatim}

The matplace command places M2 in M1 beginning at row 1 and column 3. V1 is placed in column 3 of M1, while V2 is placed in row 80 of M1.

**Copying data between matrix objects and other objects**

The previous sections described techniques for copying data between matrix objects such as vectors, matrices and scalars. In this section, we describe techniques for copying data between matrix objects and other EViews objects such as series and groups.

Keep in mind that there are two primary differences between the ordinary series or group objects and the matrix objects. First, operations involving series and groups use information about the current workfile sample, while matrix objects do not. Second, there are important differences in the handling of missing values (NAs) between the two types of objects.
Direct Assignment

The easiest method to copy data from series or group objects to a matrix object is to use direct assignment. Place the destination matrix object on the left side of an equal sign, and place the series or group to be converted on the right.

If you use a series object on the right, EViews will only include the observations from the current sample to make the vector. If you place a group object on the right, EViews will create a rectangular matrix, again only using observations from the current sample.

While very convenient, there are two principal limitations of this approach. First, EViews will only include observations in the current sample when copying the data. Second, observations containing missing data (NAs) for a series, or for any series in the group, are not placed in the matrix. Thus, if the current sample contains 20 observations but the series or group contains missing data, the dimension of the vector or matrix will be less than 20. Below, we provide you with methods which allow you to override the current sample and to retain missing values.

Examples:

```
smpl 1963:3 1993:6
fetch hsf gmpyq
group mygrp hsf gmpyq
vector xvec = gmpyq
matrix xmat = mygrp
```

These statements create the vector XVEC and the two column matrix XMAT containing the non-missing series and group data from 1963:3 to 1993:6. Note that if GMPYQ has a missing value in 1970:01, and HSF contains a missing value in 1980:01, both observations for both series will be excluded from XMAT.

When performing matrix assignment, you may refer to an element of a series, just as you would refer to an element of a vector, by placing an index value in parentheses after the name. An index value $i$ refers to the $i$-th element of the series from the beginning of the workfile range. For example, if the range of the current annual workfile is 1961 to 1980, the expression GNP(6) refers to the 1966 value of GNP. These series element expressions may be used in assigning specific series values to matrix elements, or to assign matrix values to a specific series element. For example:

```
matrix(5,10) x
series yser = nrnd
x(1,1) = yser(4)
yser(5) = x(2,3)
yser(6) = 4000.2
```
assigns the fourth value of the series YSER to X(1,1), and assigns to the fifth and sixth values of YSER, the X(2,3) value and the scalar value “4000.2”, respectively.

While matrix assignments allow you to refer to elements of series as though they were elements of vectors, you cannot generally use series in place of vectors. Most vector and matrix operations will error if you use a series in place of a vector. For example, you cannot perform a rowplace command using a series name.

Furthermore, note that when you are not performing matrix assignment, a series name followed by a number in parentheses will indicate that the lag/lead operator be applied to the entire series. Thus, when used in generating series or in an equation, system, or model specification, GNP(6) refers to the sixth lead of the GNP series. To refer to specific elements of the GNP series in these settings, you should use the @elem function.

Copy using @convert

The @convert function takes a series or group object and, optionally, a sample object, and returns a vector or rectangular matrix. If no sample is provided, @convert will use the workfile sample. The sample determines which series elements are included in the matrix. Example:

```plaintext
smpl 61 90
group groupx inv gdp ml
vector v = @convert(gdp)
matrix x = @convert(groupx)
```

X is a $30 \times 3$ matrix with the first column containing data from INV, the second column from GDP, and the third column from M1.

As with direct assignment, the @convert function excludes observations for which the series or any of the series in the group contain missing data. If, in the example above, INV contains missing observations in 1970 and 1980, V would be a 29 element vector while X would be a $28 \times 3$ matrix. This will cause errors in subsequent operations that require V and X to have a common row dimension.

There are two primary advantages of using @convert over direct assignment. First, since @convert is a function, it may be used in the middle of a matrix expression. Second, an optional second argument allows you to specify a sample to be used in conversion. For example,

```plaintext
sample s1.set 1950 1990
matrix x = @convert(grp,s1)
sym y = @inverse(@inner(@convert(grp,s1)))
```

performs the conversion using the sample defined in S1.
Copy data between Series and Matrices

EViews also provides three useful commands that perform explicit conversions between series and matrices with control over both the sample, and the handling of NAs.

`stom` (Series TO Matrix) takes a series or group object and copies its data to a vector or matrix using either the current workfile sample, or the optionally specified sample. As with direct assignment, the `stom` command excludes observations for which the series or any of the series in the group contain missing data.

Example:

```
sample smpl_cnvrt.set 1950 1995
smpl 1961 1990
group group1 gnp gdp money
vector(46) vec1
matrix(3,30) mat1
stom(gdp,vec1,smpl_cnvrt)
stom(group1,mat1)
```

While the operation of `stom` is similar to `@convert`, `stom` is a command and cannot be included in a matrix expression. Furthermore, unlike `@convert`, the destination matrix or vector must already exist and have the proper dimension.

`stomna` (Series TO Matrix with NAs) works identically to `stom`, but does not exclude observations for which there are missing values. The elements of the series for the relevant sample will map directly into the target vector or matrix. Thus,

```
smpl 1951 2000
vector(50) gvector
stom(gdp,gvector)
```

will always create a 50 element vector GVECTOR that contains the values of GDP from 1951 to 2000, including observations with NAs.

`mtos` (Matrix TO Series) takes a matrix or vector and copies its data into an existing series or group, using the current workfile sample or a sample that you provide.

Examples:

```
mtos(mat1,group1)
mtos(vec1,resid)
mtos(mat2,group1,smpl1)
```

As with `stom` the destination series or group must already exist and the destination dimension given by the sample must match that of the source vector or matrix.
Matrix Expressions

A matrix expression is an expression which combines matrix objects using mathematical operators or relations, functions, and parentheses. While we discuss matrix functions in great detail below, some examples will demonstrate the relevant issues.

Examples:

\[
\begin{align*}
&\@inner(\@convert(grp,s1)) \\
&\text{mat1*vec1} \\
&\@inverse(\text{mat1+mat2})*\text{vec1} \\
&\text{mat1 > mat2}
\end{align*}
\]

EViews uses the following rules to determine the order in which the expression will be evaluated:

- You may nest any number of pairs of parentheses to clarify the order of operations in a matrix expression.
- If you do not use parentheses, the operations are applied in the following order:
  1. Unary negation operator and functions
  2. Multiplication and division operators
  3. Addition and subtraction operators
  4. Comparison operators: "$\geq $", "$>$", "$\leq $", "$<$", "$\neq $"

Examples:

\[
\begin{align*}
&\@inverse(\text{mat1+mat2})*\@inverse(\text{mat3+mat4}) \\
&\text{vec1*}@\text{inverse(mat1+mat2)}*@\text{transpose(vec1)}
\end{align*}
\]

In the first example, the matrices MAT1 and MAT2 will be added and then inverted. Similarly the matrices MAT3 and MAT4 are added and then inverted. Finally, the two inverses will be added together. In the second example, EViews first inverts MAT1 + MAT2 and uses the result to calculate a quadratic form with VEC1.

Matrix Operators

EViews provides standard mathematical operators for matrix objects.

Negation (-)

The unary minus changes the sign of every element of a matrix object, yielding a matrix or vector of the same dimension. Example:
matrix jneg = -jpos

Addition (+)
You can add two matrix objects of the same type and size. The result is a matrix object of
the same type and size. Example:
matrix(3,4) a
matrix(3,4) b
matrix sum = a + b

You can add a square matrix and a sym of the same dimension. The upper triangle of the
sym is taken to be equal to the lower triangle. Adding a scalar to a matrix object adds the
scalar value to each element of the matrix or vector object.

Subtraction (–)
The rules for subtraction are the same as the rules for addition. Example:
matrix(3,4) a
matrix(3,4) b
matrix dif = a - b

Subtracting a scalar object from a matrix object subtracts the scalar value from every ele-
ment of the matrix object.

Multiplication (*)
You can multiply two matrix objects if the number of columns of the first matrix is equal to
the number of rows of the second matrix.

Example:
matrix(5,9) a
matrix(9,22) b
matrix prod = a * b

In this example, PROD will have 5 rows and 22 columns.

One or both of the matrix objects can be a sym. Note that the product of two sym objects
is a matrix, not a sym. The @inner function will produce a sym by multiplying a matrix
by its own transpose.

You can premultiply a matrix or a sym by a vector if the number of columns of the matrix
is the same as the number of elements of the vector. The result is a vector whose dimen-
sion is equal to the number of rows of the matrix.

Example:
matrix(5,9) mat
vector(9) vec
vector res = mat * vec

In this example, RES will have 5 elements.

You can premultiply a rowvector by a matrix or a sym if the number of elements of the rowvector is the same as the number of rows of the matrix. The result is a rowvector whose dimension is equal to the number of columns of the matrix.

Example:
rowvector rres
matrix(5,9) mat
rowvector(5) row
rres = row * mat

In this example, RRES will have 9 elements.

You can multiply a matrix object by a scalar. Each element of the original matrix is multiplied by the scalar. The result is a matrix object of the same type and dimensions as the original matrix. The scalar can come before or after the matrix object. Examples:
matrix prod = 3.14159*orig
matrix xxx = d_mat*7

Division (/)
You can divide a matrix object by a scalar. Example:
matrix z = orig/3

Each element of the object ORIG will be divided by 3.

Relational Operators (==, >, >=, <, <=, <>)
Two matrix objects of the same type and size may be compared using the comparison operators (). The result is a scalar logical value. Every pair of corresponding elements is tested, and if any pair fails the test, the value 0 (FALSE) is returned; otherwise, the value 1 (TRUE) is returned.

For example, the commands
if result <> value then
   run crect
endif
It is possible for a vector to be not greater than, not less than, and not equal to a second vector. For example:

\[
\begin{align*}
\text{vector(2) v1} \\
\text{vector(2) v2} \\
v1(1) &= 1 \\
v1(2) &= 2 \\
v2(1) &= 2 \\
v2(2) &= 1
\end{align*}
\]

Since the first element of V1 is smaller than the first element of V2, V1 is not greater than V2. Since the second element of V1 is larger than the second element of V2, V1 is not less than V2. The two vectors are not equal.

**Matrix Commands and Functions**

EViews provides a number of commands and functions that allow you to work with the contents of your matrix objects. These commands and functions may be divided into roughly four distinct types:

1. Utility Commands and Functions
2. Matrix Algebra Functions
3. Descriptive Statistics Functions
4. Element Functions

The utility commands and functions provide support for creating, manipulating, and assigning values to your matrix objects. We have already seen the `@convert` function and the `stom` command, both of which convert data from series and groups into vectors and matrices.

The matrix algebra functions allow you to perform common matrix algebra manipulations and computations. Among other things, you can use these routines to compute eigenvalues, eigenvectors and determinants of matrices, to invert matrices, to solve linear systems of equations, and to perform singular value decompositions.

The descriptive statistics functions compute summary statistics for the data in the matrix object. You can compute statistics such as the mean, median, minimum, maximum, and variance, over all of the elements in your matrix.

The matrix element functions allow you to create a new matrix containing the values of a function evaluated at each element of another matrix object. Most of the functions that are available in series expressions may be applied to matrix objects. You can compute the log-
arithm of every element of a matrix, or the cumulative normal distribution at every element of a vector.

A listing of the commands and functions is included in the matrix summary on page 44. Functions for computing descriptive statistics for data in matrices are discussed in “Descriptive Statistics” on page 577. Additional details on matrix element computations are provided in “Matrix Operators” on page 34.

**Functions versus Commands**

A function generally takes arguments, and always returns a result. Functions are easily identified by the initial “@” character in the function name.

There are two basic ways that you can use a function. First, you may assign the result to an EViews object. This object may then be used in other EViews expressions, providing you access to the result in subsequent calculations. For example,

```
matrix y = @transpose(x)
```

stores the transpose of matrix X in the matrix Y. Since Y is a standard EViews matrix, it may be used in all of the usual expressions.

Second, you may use a function as part of a matrix expression. Since the function result is used *in-line*, it will not be assigned to a named object, and will not be available for further use. For example, the command

```
scalar z = vec1*@inverse(v1+v2)*@transpose(vec1)
```

uses the results of the @inverse and @transpose functions in forming the scalar expression assigned to Z. These function results will not be available for subsequent computations.

By contrast, a command takes object names and expressions as arguments, and operates on the named objects. Commands do not return a value.

Commands, which do not have a leading “@” character, must be issued alone on a line, rather than as part of a matrix expression. For example, to convert a series X to a vector V1, you would enter

```
stom(x,v1)
```

Because the command does not return any values, it may not be used in a matrix expression.
NA Handling

As noted above, most of the methods of moving data from series and groups into matrix objects will automatically drop observations containing missing values. It is still possible, however, to encounter matrices which contain missing values.

For example, the automatic NA removal may be overridden using the stomna command (p. 451). Additionally, some of the element operators may generate missing values as a result of standard matrix operations. For example, taking element-by-element logarithms of a matrix using @log will generate NAs for all cells containing nonpositive values.

EViews follows two simple rules for handling matrices that contain NAs. For all operators, commands, and functions, except the descriptive statistics function, EViews works with the full matrix object, processing NAs as required. For descriptive statistic functions, EViews automatically drops NAs when performing the calculation. These rules imply the following:

- Matrix operators will generate NAs where appropriate. Adding together two matrices that contain NAs will yield a matrix containing NAs in the corresponding cells. Multiplying two matrices will result in a matrix containing NAs in the appropriate rows and columns.
- All matrix algebra functions and commands will generate NAs, since these operations are undefined. For example, the Cholesky factorization of a matrix that contains NAs will contain NAs.
- All utility functions and commands will work as before, with NAs treated like any other value. Copying the contents of a vector into a matrix using colplace will place the contents, including NAs, into the target matrix.
- All of the matrix element functions will propagate NAs when appropriate. Taking the absolute value of a matrix will yield a matrix containing absolute values for non-missing cells and NAs for cells that contain NAs.
- The descriptive statistics functions are based upon the non-missing subset of the elements in the matrix. You can always find out how many values were used in the computations by using the @OBS function.

Matrix Views and Procs

The object listing in Chapter , “”, on page 135 lists the various views and procs for all of the matrix objects.
Matrix Graph and Statistics Views

All of the matrix objects, with the exception of the scalar object, have windows and views. For example, you may display line and bar graphs for each column of the $10 \times 5$ matrix $Z$:

```
z.line
z.bar(p)
```

each column will be plotted against the row number of the matrix.

Additionally, you can compute descriptive statistics for each column of a matrix, as well as the correlation and covariance matrix between the columns of the matrix:

```
z.stats
z.cor
z.cov
```

EViews performs listwise deletion by column, so that each group of column statistics is computed using the largest possible set of observations.

The full syntax for the commands to display and print these views is listed in the object reference.

Matrix input and output

EViews provides you with the ability to read and write files directly from matrix objects using the read and write procedures.

You must supply the name of the source file. If you do not include the optional path specification, EViews will look for the file in the default directory. The input specification follows the source file name. Path specifications may point to local or network drives. If the path specification contains a space, you must enclose the entire expression in double quotes "".

In reading from a file, EViews first fills the matrix with NAs, places the first data element in the “(1,1)” element of the matrix, then continues to read the data by row or by column, depending upon the options set.

The following command reads data into MAT1 from an Excel file CPS88 in the network drive specified in the path directory. The data are read by column, and the upper left data cell is A2.

```
mat1.read(a2,s=sheet3) "\net1\dr 1\cps88.xls"
```

To read the same file by row, you should use the “t” option:
To write data from a matrix, use the write keyword, enter the desired options, then the name of the output file. For example:

mat1.write mydt.txt

writes the data in MAT1 into the ASCII file MYDT.TXT located in the default directory.

There are many more options for controlling reading and writing of data; Chapter 5, “Basic Data Handling”, on page 73 of the User’s Guide provides extensive discussion. See also read (p. 377) and write (p. 498).

Matrix Operations versus Loop Operations

You can perform matrix operations using element operations and loops instead of the built-in functions and commands. For example, the inner product of two vectors may be computed by evaluating the vectors element-by-element:

```plaintext
scalar inprod1 = 0
for !i = 1 to @rows(vec1)
    inprod1 = inprod1 + vec1(!i)*vec2(!i)
next
```

however, this approach will generally be much slower than using the built-in function:

```plaintext
scalar inprod2 = @inner(vec1,vec2)
```

You should use the built-in matrix operators rather than loop operators whenever you can. The matrix operators are always much faster than the equivalent loop operations.

There will be cases when you cannot avoid using loop operations. For example, suppose you wish to subtract the column mean from each element of a matrix. Such a calculation might be useful in constructing a fixed effects regression estimator. First, consider a slow method involving only loops and element operations:

```plaintext
matrix(2000,10) x = @convert(mygrp1)
scalar xsum
for !i = 1 to @columns(x)
    xsum = 0
    for !j = 1 to @rows(x)
        xsum = xsum + x(!j,!i)
    next
    xsum = xsum/@rows(x)
for !j = 1 to @rows(x)
```
x(!j,!i) = x(!j,!i) - xsum
next
next

The loops are used to compute a mean for each column of data in X, and then to subtract the value of the mean from each element of the column. A better and much faster method for subtracting column means uses the built-in operators:

```eviews
matrix x = @convert(mygrp1)
vector(@rows(x)) xmean
for !i = 1 to @columns(x)
    xmean = @mean(@columnextract(x,!i))
    colplace(x,@columnextract(x,!i)-xmean,!i)
next
```

This command extracts each column of X, computes the mean, and fills the vector XMEAN with the column mean. You then subtract the mean from the column and place the result back into the appropriate column of X. While you still need to loop over the control variable !i, you avoid the need to loop over the elements of the columns.

Summary of Automatic Resizing of Matrix Objects

When you perform a matrix object assignment, EViews will resize, where possible, the destination object to accommodate the contents of the source matrix. This resizing will occur if the destination object type can be modified and sized appropriately and if the values of the destination may be assigned without ambiguity. You can, for example, assign a matrix to a vector and vice versa, you can assign a scalar to a matrix, but you cannot assign a matrix to a scalar since the EViews does not allow scalar resizing.

The following table summarizes the rules for resizing of matrix objects as a result of declarations of the form

```
object_type y = x
```

where `object_type` is an EViews object type, or as the result of an assignment statement for Y after an initial declaration, as in

```
object_type y
y = x
```

Each row of the table corresponds to the specified type of the destination object, Y. Each column represents the type and size of the source object, X. Each cell of the table shows the type and size of object that results from the declaration or assignment.
For example, consider the command

```
matrix(500,4) y = x
```

where X is a coef of size 50. The object type is given by examining the table entry corresponding to row “matrix Y” ($n = 500$, $k = 4$), and column “coef X” ($p = 50$). The entry reads “matrix($p, 1$)”, so that the result Y is a 50 × 1 matrix.
Similarly, the command

\[ \text{vector}(30) \ y = x \]

where \( X \) is a 10 element rowvector yields the 10 element rowvector \( Y \). In essence, EViews first creates the 30 element rowvector \( Y \), then resizes it to match the size of \( X \), then finally assigns the values of \( X \) to the corresponding elements of \( Y \).

## Matrix Function and Command Summary

**Utility Functions and Commands**

- `colplace` ............... Places column vector into matrix (p. 527).
- `@columnextract` ... Extracts column from matrix (p. 528).
- `@columns` ............. Number of columns in matrix object (p. 528).
- `@convert` .............. Converts series or group to a vector or matrix after removing NAs (p. 529).
- `@explode` ............. Creates square matrix from a sym (p. 535).
- `@filledmatrix` ....... Creates matrix filled with scalar value (p. 535).
- `@filledrowvector` .... Creates rowvector filled with scalar value (p. 535).
- `@filledsym` .......... Creates sym filled with scalar value (p. 536).
- `@filledvector` ....... Creates vector filled with scalar value (p. 536).
- `@getmaindiagonal` Extracts main diagonal from matrix (p. 536).
- `@identity` ............. Creates identity matrix (p. 537).
- `@implode` ............. Creates sym from lower triangle of square matrix (p. 537).
- `@makediagonal` ...... Creates a square matrix with ones down a specified diagonal and zeros elsewhere (p. 542).
- `matplace` .............. Places matrix object in another matrix object (p. 543).
- `mtos` .................... Converts a matrix object to series or group (p. 544).
- `@permute` ............. Permutes the rows of the matrix (p. 547).
- `@resample` ............ Randomly draws from the rows of the matrix (p. 548).
- `@rowextract` .......... Extracts rowvector from matrix object (p. 550).
- `rowplace` .............. Places a rowvector in matrix object (p. 550).
- `@rows` ................... Returns the number of rows in matrix object (p. 550).
- `stom` ................... Converts series or group to vector or matrix after removing observations with NAs (p. 551).
- `stomna` ................... Converts series or group to vector or matrix without removing observations with NAs (p. 552).
- `@subextract` .......... Extracts submatrix from matrix object (p. 554).
- `@transpose` .......... Transposes matrix object (p. 556).
@unitvector ........Extracts column from an identity matrix (p. 556).
@vec ................Stacks columns of a matrix object (p. 558).
@vech .............Stacks the lower triangular portion of matrix by column (p. 558).

Matrix Algebra Functions
@cholesky .........Computes Cholesky factorization (p. 527).
@cond ................Calculates the condition number of a square matrix or sym
                    (p. 528).
@det ................Calculate the determinant of a square matrix or sym (p. 532).
@eigenvalues ......Returns a vector containing the eigenvalues of a sym (p. 533).
@eigenvectors ......Returns a square matrix whose columns contain the eigenvectors
                    of a matrix (p. 534).
@inner ..............Computes the inner product of two vectors or series, or the inner
                    product of a matrix object (p. 537).
@inverse ..........Returns the inverse of a square matrix object or sym (p. 539).
@issingular ........Returns 1 if the square matrix or sym is singular, and 0 otherwise
                    (p. 540).
@kronecker ........Computes the Kronecker product of two matrix objects (p. 540).
@norm ...............Computes the norm of a matrix object or series (p. 546).
@outer ..............Computes the outer product of two vectors or series, or the outer
                    product of a matrix object (p. 547).
@rank ...............Returns the rank of a matrix object (p. 547).
@solvesystem ......Solves system of linear equations, $Mx = v$, for $x$ (p. 551).
@svd ................Performs singular value decomposition (p. 555).
@trace ..............Computes the trace of a square matrix or sym (p. 555).

Matrix Descriptive Statistics Functions
@cor ................Computes correlation between two vectors, or between the columns
                    of a matrix (p. 530)
@cov ..................Computes covariance between two vectors, or between the columns
                    of a matrix (p. 530).

The remaining descriptive statistics functions that may be used with matrices and vectors
are described in “Descriptive Statistics” on page 577.

Matrix Element Functions
EVViews supports matrix element versions of the following categories of functions.
<table>
<thead>
<tr>
<th>Category</th>
<th>Matrix Element Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Mathematical Functions (p. 575)</td>
<td>All, except for &quot;@inv&quot;</td>
</tr>
<tr>
<td>Additional and Special Functions (p. 580)</td>
<td>All</td>
</tr>
<tr>
<td>Trigonometric Functions (p. 583)</td>
<td>All</td>
</tr>
<tr>
<td>Statistical Distribution Functions (p. 584)</td>
<td>All</td>
</tr>
</tbody>
</table>
Chapter 4. Working with Tables

There are three types of tables in EViews: tabular views, which are tables used in the display of views of other objects, named table objects, and unnamed table objects. The main portion of this discussion focuses on the use of commands to customize the appearance of named table objects. The latter portion of the chapter describes the set of tools that may be used to customize the display characteristics of spreadsheet views of objects (see “Customizing Spreadsheet Views” beginning on page 57).

You may use EViews commands to generate custom tables of formatted output from your programs. A table object is an object made up of rows and columns of cells, each of which can contain either a number or a string, as well as information used to control formatting for display or printing.


Creating a Table

There are two basic ways to create a table object: by freezing an object view, or by issuing a table declaration.

Creating Tables from an Object View

You may create a table object from another object, by combining an object view command with the freeze (p. 275) command. Simply follow the freeze keyword with an optional name for the table object, and the tabular view to be frozen. For example, since the command

\[ \text{grp6.stats} \]

displays the statistics view of the group GRP6, the command

\[ \text{freeze(mytab)} \text{ grp6.stats} \]

creates and displays a table object MYTAB containing the contents of the previous view.

You should avoid creating unnamed tables when using commands in programs since you will be unable to refer to, or work with the resulting object using commands. If the MYTAB option were omitted in the previous example, EViews would create and display an untitled table object. This table object may be customized interactively, but may not be referred to in programs. You may, of course, assign a name to the table interactively.
Once you have created a named table object, you may use the various table object procs to further customize the appearance of your table. See “Customizing Tables” beginning on page 50.

Declaring Tables
To declare a table, indicate the number of rows and columns and provide a valid name. For example,

```
  table(10,20) bestres
```

creates a table with 10 rows and 20 columns named BESTRES. You can change the size of a table by declaring it again. Redeclaring to a larger size does not destroy the contents of the table; any cells in the new table that existed in the original table will contain their previous values.

Tables are automatically resized when you attempt to fill a table cell outside the table’s current dimensions. This behavior is different from matrices which give an error when an out of range element is accessed.

Assigning Table Values
You may modify the contents of cells in a table using assignment statements. Each cell of the table can be assigned either a string or a numeric value.

Assigning Strings
To place a string value into a table cell, follow the table name by a cell location (row and column pair in parentheses), then an equal sign and a string expression.

For example,

```
  table bestres
  bestres(1,6) = "convergence criterion"
  %strvar = "lm test"
  bestres(2,6) = %strvar
  bestres(2,6) = bestres(2,6) + " with 5 df"
```

creates the table BESTRES and places various string values into cells of the table.

Assigning Numbers
Numbers can be entered directly into cells, or they can be converted to strings before being placed in the table.

Unless there is a good reason to do otherwise, we recommend that numbers be entered directly into table cells. If entered directly, the number will be displayed according to the
numerical format set for that cell; if the format is changed, the number will be redisplayed according to the new format. If the number is first converted to a string, the number will be frozen in that form and cannot be reformatted.

For example,

```
table tab1
    tab1(3,4) = 15.345
    tab1(4,2) = 1e-5
    !ev = 10
    tab1(5,1) = !ev
    scalar f = 12345.67
    tab1(6,2) = f
```

creates the table TAB1 and assigns numbers to various cells.

Assignment with Formatting

The `setcell` command is like direct cell assignment in that it allows you to set the contents of a cell, but `setcell` also allows you to provide a set of formatting options for the cell. If you desire greater control over formatting, or if you wish to alter the format of a cell without altering its contents, you should use the tools outlined in “Customizing Tables” beginning on page 50.

The `setcell` command takes the following arguments:

- the name of the table
- the row and the column of the cell
- the number or string to be placed in the cell
- (optionally) a justification code or a numerical format code, or both

The justification codes are:

- “c” for centered (default)
- “r” for right-justified
- “l” for left-justified

The numerical format code determines the format with which a number in a cell is displayed; cells containing strings will be unaffected. The format code can either be a positive integer, in which case it specifies the number of digits to be displayed after the decimal point, or a negative integer, in which case it specifies the total number of characters to be used to display the number. These two cases correspond to the `fixed decimal` and `fixed character` fields in the number format dialog.
Note that when using a negative format code, one character is always reserved at the start of a number to indicate its sign, and if the number contains a decimal point, that will also be counted as a character. The remaining characters will be used to display digits. If the number is too large or too small to display in the available space, EViews will attempt to use scientific notation. If there is insufficient space for scientific notation (six characters or less), the cell will contain asterisks to indicate an error.

Some examples of using `setcell`:

```plaintext
setcell(tabres,9,11,%label)
```

puts the contents of `%LABEL` into row 9, column 11 of the table `TABRES`.

```plaintext
setcell(big_tab1,1,1,%info,"c")
```

inserts the contents of `%INFO` in `BIG_TAB1(1,1)`, and displays the cell with centered justification.

```plaintext
setcell(tab1,5,5,!data)
```

puts the number `!DATA` into cell (5,5) of table `TAB1`, with default numerical formatting.

```plaintext
setcell(tab1,5,6,!data,4)
```

puts the number `!DATA` into `TAB1`, with 4 digits to the right of the decimal point.

```plaintext
setcell(tab1,3,11,!data,"r",3)
```

puts the number `!DATA` into `TAB1`, right-justified, with 3 digits to the right of the decimal point.

```plaintext
setcell(tab1,4,2,!data,-7)
```

puts the number in `!DATA` into `TAB1`, with 7 characters used for display.

**Customizing Tables**

EViews provides considerable control over the appearance of table objects, providing a variety of table procedures allowing you specify row heights and column widths, content formatting, justification, font face, size, and color, cell background color and borders. Cell merging and annotation are also supported.

**Column Width and Row Height**

We begin by noting that if the contents of a cell is wider or taller than the display width or height of the cell, part of the cell contents may not be visible. You may use the `setwidth` (p. 431) and `setheight` (p. 423) table procedures to change the dimensions of a column or row of table cells.

To change the column widths for a set of columns in a table, use the `setwidth` keyword followed by a column range specification in parentheses, and a desired width.
The column range should be either a single column number or letter (e.g., “5”, “E”), a colon delimited range of columns (from low to high, e.g., “3:5”, “C:E”), or the keyword “@ALL”. The width unit is computed from representative characters in the default font for the current table (the EViews table default font at the time the table was created), and corresponds roughly to a single character. Width values may be non-integer values with resolution up to 1/10 of a unit. The default width value for columns in an unmodified table is 10.

For example, both

```plaintext
tabl.setwidth(2) 12
tabl.setwidth(B) 12
```

set the width of column 2 to 12 width units, while the command

```plaintext
tabl.setwidth(2:10) 20
```

sets the widths for columns 2 through 10 to 20 width units. To set all of the column widths, use the “@ALL” keyword.

```plaintext
tabl.setwidth(@all) 20
```

Similarly, you may specify row heights using the `setheight` keyword, followed by a row specification in parentheses, and a desired row height.

Rows are specified either as a single row number (e.g., “5”), as a colon delimited range of rows (from low to high, e.g., “3:5”), or using the keyword “@ALL”. Row heights are given in height unit values, where height units are in character heights. The character height is given by the font-specific sum of the units above and below the baseline and the leading in the default font for the current table. Height values may be non-integer values with resolution up to 1/10 of a height unit. The default row height value is 1.

For example,

```plaintext
tabl.setheight(2) 1
```

sets the height of row 2 to match the table default font character height, while

```plaintext
tabl.setheight(2) 3.5
```

increases the row height to 3-1/2 character heights.

Similarly, the command

```plaintext
tabl.setheight(2:7) 1.5
```

sets the heights for rows 2 through 7 to 1-1/2 character heights.
tabl.setheight(@all) 2

sets all row heights to twice the default height.

Earlier versions of EViews supported the setting of column widths using the setcolwidth (p. 410) command. This command, which is provided for backward compatibility, only offers a subset of the capabilities of setwidth (p. 431).

Cell Formatting

A host of cell characteristics may be set using table procedures. Each procedure is designed to work on individual cells, ranges of cells, or the entire table.

Content Formatting

Cell content formatting allows you to alter the appearance of the data in a table cell without changing the contents of the cell. Using the table proc setformat (p. 419), you may, for example, instruct EViews to change the format of a number to scientific or fixed decimal, or to display a date number in a different date format. These changes in display format do not alter the cell values.

To format the contents of table cells, simply follow the table name with a period and the setformat proc keyword, followed by a cell range specification in parentheses, and then a valid numeric or date format string. The cell range may be specified in a number of ways, including individual cells, cell rectangles, row or column ranges or the entire table. See setformat (p. 419) for a description of cell range specification and numeric and date format string syntax.

For example, to set the format for the fifth column of a matrix to fixed 5-digit precision, you may provide the format specification:

tabl.setformat(e) f.5

To set a format for the cell in the third row of the fifth column to scientific notation with 5 digits of precision, specify the individual cell, as in,

tabl.setformat(3,e) e.5
tabl.setformat(e3) e.5

To specify the format for a rectangle of cells, specify the upper left and lower right cells in the rectangle. The following commands set cells in the same region to show 3-significant digits, with negative numbers in parentheses,

tabl.setformat(2,B,10,D) (g.3)
tabl.setformat(r2c2:r10c4) (g.3)
tabl.setformat(b2:d1) (g.3)

The rectangle of cells is delimited by row 2, column 2, and row 10, column 4.
Alternately you may provide a date format for the table cells. The command

\[
\text{tabl.setformat(@all) "dd/MM/YY HH:MI:SS.SSS"}
\]

will display numeric values in the entire table using formatted date strings containing days followed by months, years, hours, minutes and seconds, to a resolution of thousandths of a second.

Note that changing the display format of a cell that contains a string will have no effect unless the cell is later changed to contain a numerical value.

Justification and Indentation

The cell justification and indentation control the position of the table cell contents within the table cell itself.

You may use the \texttt{setjust} (p. 425) proc to position the cell contents in the cell. Simply use the \texttt{setjust} keyword, followed by a cell range specification in parentheses, and one or more keywords describing a vertical or horizontal position for the cell contents. You may use the keywords \texttt{auto}, \texttt{left}, \texttt{right}, and \texttt{center} to control horizontal positioning, and \texttt{top}, \texttt{middle}, and \texttt{bottom} to control vertical positioning. You may use the \texttt{auto} keyword to specify left justification for string cells and right justification for numeric cells.

For example,

\[
\text{tabl.setjust(@all) top left}
\]

sets the justification for all cells in the table to top left, while

\[
\text{tabl.setjust(2,B,10,D) center}
\]

horizontally centers the cell contents in the rectangle from B2 to D10, while leaving the vertical justification unchanged.

In addition, you may use \texttt{setindent} (p. 424) to specify a left or right indentation from the edge of the cell for cells that are left or right justified, respectively. You should use the \texttt{setindent} keyword followed by a cell range in parentheses, and an indentation unit, specified in 1/5 of a width unit. Indentation is only relevant for non-center justified cells.

For example,

\[
\text{tabl.setjust(2,B,10,D) left}
\]
\[
\text{tabl.indent(2,B,10,D) 2}
\]

left-justifies, then indents the specified cells by 2/5 of a width unit from the left-hand side of the cell.

Alternatively,

\[
\text{tab2.setjust(@all) center}
\]
tab2.indent(@all) 3

will set the indentation for all cells in the table to 3/5 of a width unit, but this will have no
effect on the center justified cells. If the cells are later modified to be left or right justified,
the indentation will be used. If you subsequently issue the command

    tab2.indent(@all) right

the cells will be indented 3/5 of a width unit from the right-hand edges.

Fonts

You may specify font face and characteristics, and the font color for table cells using the
\texttt{setfont} (p. 418) and \texttt{settextcolor} (p. 430) table procs.

The \texttt{setfont} proc should be used to set the font face, size, boldface, italic, strikethrough
and underline characteristics for table cells. You should provide a cell range specification,
and one or more font arguments corresponding to font characteristics that you wish to
modify. For example,

    tab1.setfont(3,B,10,D) "Times New Roman" +u 8pt

changes the text in the specified cells to Times New Roman, 8 point, underline. Similarly,

    tab1.setfont(4,B) -b +i -s

adds the italic to and removes boldface and strikethrough from the B4 cell.

To set the color of your text, use \texttt{settextcolor} with a cell range specification and a color
specification. Color specifications may be provided using the \texttt{@RGB} settings, or using one
of the EViews predefined colors keywords:

    tab1.settextcolor(f2:g10) @rgb(255, 128, 0)
    tab1.settextcolor(f2:g10) orange

sets the text color for the specified cells to orange. See \texttt{setfillcolor} (p. 416) for a com-
tele description of color specifications.

Background and Borders

You may set the background color for cells using the \texttt{setfillcolor} (p. 416) table proce-
dure. Specify the cell range and provide a color specification using \texttt{@RGB} settings or one
of the predefined color keywords. The commands

    tab1.setfillcolor(R2C3:R3C6) lgray
    tab1.setfillcolor(2,C,3,F) @rgb(192, 192, 192)

both set the background color of the specified cells to light gray.
The `setlines` (p. 427) table proc may be used to draw borders or lines around specified table cells. If a single cell is specified, you may draw borders around the cell or a double line through the center of the cell. If multiple columns or rows is selected, you may, in addition, add borders between cells.

Follow the name of the table object with a period, the `setlines` keyword, a cell range specification, and one or more line arguments describing the lines and borders you wish to draw. For example,

```
    tab1.setlines(b2:d6) +a -h -v
```

which first adds all borders ("a") to the cells in the rectangle defined by B2 and D6, then removes the inner horizontal ("h"), and inner vertical ("v") borders. The command

```
    tab1.setlines(2,b) +o
```

adds borders to the outside ("o"), all four borders, of the B2 cell.

You may also use the `setline` (p. 426) command to place double horizontal separator lines in the table. Enter the `setline` keyword, followed by the name of the table, and a row number, both in parentheses. For example,

```
    setline(bestres,8)
```

places a separator line in the eighth row of the table BESTRES. To remove the line, you must use the `setlines` proc:

```
    bestres.setlines(8) -d
```

removes the double separator lines from all of the cells in the eighth row of the table.

**Cell Annotation and Merging**

Each cell in a table object is capable of containing a comment. Cell comments contain text that is hidden until the mouse cursor is placed over the cell containing the comment. Comments are useful for adding notes to a table without changing the appearance of the table.

To add a comment with the `comment` (p. 226) table proc, follow the name of the table object with a period, a single cell identifier (in parentheses), and the comment text enclosed in double quotes. If no comment text is provided, a previously defined comment will be removed.

To add a comment “hello world” to the cell in the second row, fourth column, you may use the command:

```
    tab1.comment(d2) "hello world"
```

To remove the comment simply repeat the command, omitting the text:
In addition, EViews permits you to merge cells horizontally in a table object. To merge together multiple cells in a row or to unmerge previously merged cells you should use the setmerge (p. 428) table proc. Enter the name of the table object, a period, followed by a cell range describing the cells in a single row that are to be merged.

If the first specified column is less than the last specified column (left specified before right), the cells in the row will be merged left to right, otherwise, the cells will be merged from right to left. The contents of the merged cell will be taken from the first cell in the merged region. If merging from left to right, the leftmost cell contents will be used; if merging from right to left, the rightmost cell contents will be displayed.

For example,

```
tab1.setmerge(a2:d2)
```
merges the cells in row 2, columns 1 to 4, from left to right, while

```
tab2.setmerge(d2:a2)
```
merges the cells in row 2, columns 2 to 5, from right to left. The cell display will use the leftmost cell in the first example, and the rightmost in the second.

If you specify a merge involving previously merged cells, EViews will unmerge all cells within the specified range. We may then unmerge cells by issuing the setmerge command using any of the previously merged cells. The command

```
tab2.setmerge(r2c4)
```
unmerges the previously merged cells.

**Labeling Tables**

Tables have a label view to display and edit information such as the graph name, last modified date, and remarks. To modify or view the label information, use the label (p. 302) command:

```
table11.label(r) Results from GMM estimation
```
This command shows the label view, and the “r” option appends the text “Results from GMM estimation” to the remarks field of TABLE11.

To return to the basic table view, use the table keyword:

```
table11.table
```
All changes made in label view will be saved with the table.
Printing Tables

To print a table, use the print (p. 373) command, followed by the table object name. For example,

```
print table1
```

The print destination is taken from the EViews global print settings.

Exporting Tables to Files

You may use the table save (p. 393) procedure to save the table to disk as a CSV (comma separated file), tab-delimited ASCII text, RTF (Rich text format), or HTML file.

You must specify a file name and an optional file type, and may also provide options to specify the cells to be saved, text to be written for NA values, and precision with which numbers should be written. RTF and HTML files also allow you to save the table in a different size than the current display. If a file type is not provided, EViews will write a CSV file.

For example,

```
tab1.save(t=csv, n="NAN") mytable
```
saves TAB1 in the default directory as a CSV file MYTABLE.CSV, with NA values translated to the text “NAN”.

Alternatively,

```
tab1.save(r=B2:C10, t=html, s=.5) c:\data\MyTab2
```
saves the specified cells in TAB1 as an HTML file to MYTAB2.HTM in the directory C:\DATA. The table is saved at half of the display size.

Customizing Spreadsheet Views

A subset of the table procs for customizing table display are also available for customizing spreadsheet views of objects. You may use the setformat (p. 419), setindent (p. 424), setjust (p. 425), and setwidth (p. 431) procs to specify the spreadsheet views of numeric and alpha series objects, series objects in groups, or matrix objects.

Suppose, for example, that you wish to set the format of the spreadsheet view for series SER1. Then

```
ser1.setformat f.5
ser1.setjust right center
ser1.setindent 3
```
ser1.setwidth 10
ser1.sheet
sets the spreadsheet display format for SER1 and then displays the view.

Similarly, you may set the characteristics for a matrix object
mat1.setformat f.6
mat1.setwidth 8
mat1.sheet
For group spreadsheet formatting, you must specify a column range specification. For example,
group1.setformat(2) (f.7)
group1.setwidth(2) 10
group1.setindent(b) 6
group1.sheet
set the formats for the second series in the group, then displays the spreadsheet view.
group1.setwidth(@(all)) 10
sets the width for all columns in the group spreadsheet to 10.

Note that the group specified formats are used only to display series in the group and are not exported to the underlying series. Thus, if MYSER is the second series in GROUP1, the spreadsheet view of MYSER will use the original series settings, not those specified using the group procs.

Table Summary
See “Table” (p. 169) for a full listing of formatting procs that may be used with table objects. See also the individual objects for the procs that are available for formatting spreadsheet views.
Chapter 5. Working with Graphs

EViews provides an extensive set of commands to generate and customize graphs from the command line or using programs. A summary of the graph commands detailed below may be found under “Graph” (p. 143).

In addition, Chapter 14, “Graphs, Tables, and Text Objects”, on page 387 of the User’s Guide describes graph customization in detail, focusing on the interactive method of working with graphs.

Creating a Graph

There are three types of graphs in EViews: graphs that are views of other objects, and named or unnamed graph objects. The commands provided for customizing the appearance of your graphs are available for use with named graph objects. You may use the dialogs interactively to modify the appearance of all types of graphs.

Displaying graphs as object views

To display a graph as a view of an object, you may form a command using the object name followed by a graph type keyword and any relevant options for that type of graph. You may use any of the following graph type command keywords: line, bar, area, spike, errbar, hilo, pie, scat, xy, xyline, and xypair.

For example, you may use the following two commands to display graph views of series and groups:

\begin{verbatim}
ser2.area(n)
grp6.xyline(b)
\end{verbatim}

The first command plots the series SER2 as an area graph with normalized scaling. The second command provides an XY line graph view of the group GRP6, with the series plotted in pairs.

Alternatively, there are many EViews commands which display specialized graph views of objects. For example, you may display a boxplot view of a group or series, using the boxplot (p. 204) or boxplotby (p. 205) procs, respectively (see “Boxplots” on page 382 of the User’s Guide for a detailed explanation of boxplots).

For a group object, the command

\begin{verbatim}
group01.boxplot(nofarout)
\end{verbatim}

displays boxplots of the series in GROUP01, and does not include far outliers. Likewise,
income.boxplotby sex race

displays boxplots for the INCOME series classified by SEX and RACE.

Similarly, you may display a histogram view of a series using the hist (p. 294) series view command:

lwage.hist

computes descriptive statistics and displays a histogram of the LWAGE series.

It is important to note that graph views of objects differ from graph objects in important ways.

- First, graph views of objects may not be customized using commands after they are first created. The graph commands for customizing an existing graph are designed for use only with graph objects.

- Second, while you may use interactive dialogs to customize an existing object graph view, we caution you that there is no guarantee that the customization will be permanent. In many cases, the customized settings will not be saved with the object and will be discarded when the view changes, or if the object is closed and then opened. For example, if you display a histogram graph view of a series, change the bar color and close the series, the color will revert to the default when the object is opened.

In contrast, graph objects may be customized extensively after they are created. Any customization of a graph object is permanent, and will be saved with the object.

In the remainder of our discussion, we will focus on the creation and customization of graph objects.

Creating graph objects from object views

If you wish to create a graph object from another object, you should combine the object view command with the freeze (p. 275) command. Simply follow the freeze keyword with an optional name for the graph object, and the object view to be frozen. For example,

freeze grp6.xyline(b)

creates and displays an unnamed graph object of the GRP6 view showing an XY line graph with the series are plotted in pairs. Note that if the current view of GRP6 is an XY line graph, freezing the object view will create a graph object using existing custom appearance settings such as line color, axis assignment, etc., overridden using any options provided in the command (e.g., “b”).
You should avoid creating unnamed graphs when using commands in programs since you will be unable to refer to, or work with the resulting object in a program. Instead, you should tell EViews to create a named object, as in

\[ \text{freeze(graph1)} \text{ grp6.line} \]

which creates a graph object GRAPH1 containing a line graph of the data in GRP6. Note that using the \text{freeze} command with a name for the graph will create the graph object and store it in the workfile without showing it. Furthermore, since we have frozen a graph type (line) that is different from our current XY line view, existing custom appearance settings will not be copied to the new graph.

Once you have created a named graph object, you may use the various graph object procs to further customize the appearance of your graph. See “Customizing a Graph” beginning on page 64.

Creating named graph objects

There are three direct methods for creating a named graph object. First, you may use the \text{freeze} command as described in “Creating graph objects from object views” on page 60. Alternatively, you may declare a graph object using the \text{graph} command. The \text{graph} command may be used to create graph object with a specific graph type or to merge existing graph objects.

Specifying a graph by type

To specify a graph by type you should use the \text{graph} keyword, followed by a name for the graph, the type of graph you wish to create, and a list of series (see “Graph Type Commands” on page 143 for a list of types). If a type is not specified, a line graph will be created.

For example,

\[ \text{graph gr1 ser1 ser2} \]
\[ \text{graph gr2.line ser1 ser2} \]

creates graph objects GR1 and GR2 containing the line graph view of SER1 and SER2.

Similarly,

\[ \text{graph gr3.xyline group3} \]

creates a graph object GR3 containing the XY line graph view of the series in GROUP3.

Each graph type provides additional options, which may be included when declaring the graph. Among the most important options are those for controlling scaling or graph type.

The scaling options include:
Automatic scaling ("a"), in which series are graphed using the default single scale. The default is left for most graphs, or left and bottom for XY graphs.

- Dual scaling without crossing ("d") scales the first series on the left and all other series on the right. The left and right scales will not overlap.
- Dual scaling with possible crossing ("x") is the same as the “d” option, but will allow the left and right scales to overlap.
- Normalized scaling (“n”), scales using zero mean and unit standard deviation.

For example, the commands

```plaintext
graph g1.xyline(d) unemp gdp inv
show g1
```

create and display an XY line graph of the specified series with dual scales and no crossing.

The graph type options are:

- Stacked graph (“s”) plots the cumulative addition of the series, so the value of a series is the difference between the lines, bars, or areas.
- Mixed graph (“l”) creates a single graph in which the first series is the selected graph type (bar, area, or spike) and all remaining series are line graphs.
- Multiple graph (“m”) plots each series in a separate graph within an object view.
- Pairing (“b”), for XY graphs: By default, the first series will be plotted on the horizontal axis and all remaining series will be plotted on the vertical axis. Using the pairing option, you may choose to plot the data in pairs, where the first two series are plotted against each other, the second two series are plotted against each other, and so forth.

For example, the commands

```plaintext
group grp1 sales1 sales2
graph grsales.bar(s) grp1
show grsales
```

create a group GRP1 containing the series SALES1 and SALES2, then create and display a stacked bar graph GRSALES of the series in the group.

You should consult the command reference entry for each graph type for additional information, including a list of the available options (e.g., see `bar` (p. 199) for complete details on bar graphs).
Merging graph objects
The graph command may also be used to merge existing named graph objects into a named multiple graph object. For example,

```
    graph gr2.merge gr1 grsales
```

creates a multiple graph object GR2, combining two graph objects previously created.

Creating unnamed graph objects
There are two ways of creating an unnamed graph object. First, you may use the `freeze` (p. 275) command as described in “Creating graph objects from object views” on page 60.

Alternatively, you may use any of the graph type keywords as a command. Follow the keyword with any available options for that type, and a list of the objects to graph. EViews will create an unnamed graph of the specified type that is not stored in the workfile. For instance,

```
    line(x) ser1 ser2 ser3
```

creates a line graph with series SER1 scaled on the left axis and series SER2 and SER3 scaled on the right axis.

If you later decide to name this graph, you may do so interactively by clicking on the Name button in the graph button bar. Alternatively, EViews will prompt you to name or delete any unnamed objects before closing the workfile.

Note that there is no way to name an unnamed graph object in a program. We recommend that you avoid creating unnamed graphs in programs since you will be unable to use the resulting object.

Changing Graph Types
You may change the graph type of a named graph object by following the object name with the desired graph type keyword and any options for that type. For example,

```
    grsales.bar(l)
```

converts the bar graph GRSALES, created above, into a mixed bar-line graph, where SALES1 is plotted as a bar graph and SALES2 is plotted as a line graph within a single graph.

Note that special graphs, such as boxplots and histograms, do not allow you to change the graph type.

Graph options are preserved when changing graph types. This includes attributes such as line color and axis assignment, as well as objects added to the graph, such as text labels,
lines and shading. Commands to modify the appearance of named graph objects are described in “Customizing a Graph” on page 64.

Note, however, that the line and fill graph settings are set independently. Line attributes apply to line and spike graphs, while fill attributes apply to bar, area, and pie graphs. For example, if you have modified the color of a line in a spike graph, this color will not be used for the fill area if the graph is changed to an area graph.

Customizing a Graph

EViews provides a wide range of tools for customizing the appearance of a named graph object. Nearly every display characteristic of the graph may be modified, including the appearance of lines and filled areas, legend characteristics and placement, frame size and attributes, and axis settings. In addition, you may add text labels, lines, and shading to the graph.

You may modify the appearance of a graph using dialogs or via the set of commands described below. Note that the commands are only available for graph objects since they take the form of graph procedures.

Line characteristics

For each data line in a graph, you are able to modify color, width, pattern and symbol. using the setelem (p. 412) command. Follow the command keyword with an integer representing the data element in the graph you would like to modify, and one or more keywords for the characteristic you wish to change. The list of symbol and pattern keywords supported is provided in setelem (p. 412). The related command setfillcolor (p. 416) contains a list of available color keywords and RGB settings.

To modify line color and width you should use the lcolor and lwidth keywords:

```eviews
graph gr1.line ser1 ser2 ser3
gr1.setelem(3) lcolor(orange) lwidth(2)
gr1.setelem(3) lcolor(255, 128, 0) lwidth(2)
```

The first command creates a line graph GR1 with colors and widths taken from the global defaults, while the latter two commands equivalently change the graph element for the third series to an orange line 2 points wide.

Each data line in a graph may be drawn with a line, symbols, or both line and symbols. The drawing default is given by the global options, but you may elect to add lines or symbols using the lpattern or symbol keywords.

To add circular symbols to the lines for element 3, you may enter
gr1.setelem(3) symbol(circle)

Note that this operation modifies the existing options for the symbols, but that the line type, color and width settings from the original graph will remain. To return to line only or symbol only in a graph in which both lines and symbols are displayed, you may turn off either symbols or patterns, respectively, by using the “none” type:

gr1.setelem(3) lpat(none)

or

gr1.setelem(3) symbol(none)

The first example removes the line from the drawing for the third series, so only the circular symbol is used. The second example removes the symbol, so only the line is used.

If you attempt to remove the lines or symbols in a graph that contains only lines or symbols, respectively, the graph will change to show the opposite type. For example,

gr1.setelem(3) lpat(dash2) symbol(circle)
gr1.setelem(3) symbol(none)
gr1.setelem(3) lpat(none)

will change GR1 to show symbols only for element 3, since the latter command turns off lines in a line-only graph.

The examples above describe customization of the basic elements common to most graph types. “Modifying Boxplots” on page 78 provides additional discussion of setelem options for customizing boxplot data elements.

Use of color with lines and filled areas

By default, EViews automatically formats graphs to accommodate output in either color or black and white. When a graph is sent to a printer or saved to a file in black and white, EViews translates the colored lines and fills seen on the screen into an appropriate black and white representation. The black and white lines are drawn with line patterns, and fills are drawn with gray shading. Thus, the appearance of lines and fills on the screen may differ from what is printed in black and white (this color translation does not apply to boxplots).

You may override this auto choice display method by changing the global defaults for graphs. You may choose, for example, to display all lines and fills as patterns and gray shades with patterns, whether or not the graph uses color. All subsequently created graphs will use the new settings.

Alternatively, if you would like to override the color, line pattern, and fill settings for a given graph object, you may use the options (p. 343) graph proc.
Color

To change the color settings for an existing graph object, you should use `options` with the `color` keyword. If you wish to turn off color altogether for all lines and filled areas, you should precede the keyword with a negative sign, as in:

```
gr1.options -color
```

To turn on color, you may use the same command with the “-” omitted.

Lines and patterns

To always display solid lines in your graph, irrespective of the color settings, you should use `options` with the `linesolid` keyword. For example,

```
gr1.options linesolid
```

sets graph GR1 to use solid lines when rendering on the screen in color and when printing, even if the graph is printed in black and white. Note that this setting may make identification of individual lines difficult in a printed black and white graph, unless you change the widths or symbols associated with individual lines (see “Line characteristics” on page 64).

Conversely, you may use the `linepat` option to use patterned lines regardless of the color setting.

```
gr1.options linepat
```

One advantage of using the `linepat` option is that it allows you to see the pattern types that will be used in black and white printing without turning off color in your graph. For example, using the setelem command again, change the line pattern of the second series in GR1 to a dashed line:

```
gr1.setelem(2) lpat(dash1)
```

This command will not change the appearance of the colored lines on the screen if color is turned on and auto choice of line and fill type is set. Thus, the line will remain solid, and the pattern will not be visible until the graph is printed in black and white. To view the corresponding patterns, either turn off color so all lines are drawn as black patterned lines, or use the `linepat` setting to force patterns.

To reset the graph or to override modified global settings so that the graph uses auto choice, you may use the `lineauto` option:

```
gr1.options lineauto
```

This setting instructs the graph to use solid lines when drawing in color, and use line patterns and gray shades when drawing in black and white.
Note that regardless of the color or line pattern settings, you may always view the selected line patterns in the **Lines & Symbols** section of the graph options dialog. The dialog can be brought up interactively by double clicking anywhere in the graph.

### Filled area characteristics

You can modify the color, gray shade, and hatch pattern of each filled area in a bar, area, or pie graph.

To modify these settings, use `setelem` (p. 412), followed by an integer representing the data element in the graph you would like to modify, and a keyword for the characteristic you wish to change. For example, consider the lines

```eviews
graph mygraph.area(s) series1 series2 series3
mygraph.setelem(1) fcolor(blue) hatch(fdialgonal) gray(6)
mygraph.setelem(1) fcolor(0, 0, 255) hatch(fdialgonal) gray(6)
```

The first command create MYGRAPH, a stacked area graph of SERIES1, SERIES2, and SERIES3. The latter two commands are equivalent, modifying the first series by setting its fill color to blue with a forward diagonal hatch. If MYGRAPH graph is viewed without color, the area will appear with hatched gray shade of index 6.

See `setfillcolor` (p. 416) for a list of available color keywords, and `setelem` (p. 412) for gray shade indexes and available hatch keywords. Note that changes to gray shades will not be visible in the graph unless color is turned off.

### Using preset lines and fills

For your convenience, EViews provides you with a collection of preset line and fill characteristics. Each line preset defines a color, width, pattern, and symbol for a line, and each fill preset defines a color, gray shade, and hatch pattern for a fill. There are thirty line and thirty fill presets.

The global graph options are initially set to use the EViews preset settings. These options are used when you create a graph to provide line or fill a different appearance. The first line preset is applied to the first data line, the second preset is applied to the second data line, and so on. If your graph contains more than thirty lines or fills, the presets are simply reused in order.

You may customize the graph defaults in the global **Graph Options** dialog. Your settings will replace the existing EViews defaults, and will be applied to all graphs created in the future.

EViews allows you to use either the original EViews presets, or those you have specified in the global graph options dialog in setting the characteristics of an existing graph. The keyword `preset` is used to indicate that you should use the set of options from the corre-
sponding EViews preset; the keyword \texttt{default} is used to indicate that you should use the set of options from the corresponding global graph element defaults.

For example,

\begin{verbatim}
mygraph.setelem(2) preset(3)
\end{verbatim}

allows the second fill area in MYGRAPH to use the original EViews presets for a third fill area. In current versions of EViews, these settings include a green fill, a medium gray shade of 8, and no hatch.

Alternatively,

\begin{verbatim}
mygraph.setelem(2) default(3)
\end{verbatim}

also changes the second area of MYGRAPH, but uses the third set of user-defined presets. If you have not set modified your global graph defaults, the two commands will yield identical results.

When using the \texttt{preset} or \texttt{default} keywords with boxplots, the line color of the specified preset will be applied to the box, whiskers, and staples. See “Modifying Boxplots” on page 78 for additional information.

Scaling and axes

There are four separate commands that may be used to modify the axes and scaling characteristics of your graphs:

- First, the \texttt{setelem} (p. 412) command with the \texttt{axis} option may be used to assign data elements to different axes.
- Second, the \texttt{scale} (p. 397) command modifies the axis on which your data is scaled. This is the vertical scale in most graphs, or the horizontal and vertical scales in XY graphs. You may employ the \texttt{scale} command to modify the scaling of the data itself, for example, as when you use a logarithmic scale, or to alter the scaling of the axis, as when you enable dual scaling.
- Third, the \texttt{datelabel} (p. 237) command modifies the labeling of the bottom date/time axis in time plots. Use this command to change the way date labels are formatted or to specify label frequency.

For boxplots, you should instead use the \texttt{bplabel} (p. 210) to modify the bottom axis labeling. See “Modifying the boxplot axis” beginning on page 80 for details.
- Lastly, the \texttt{axis} (p. 197) command can be used to customize the final appearance of any axes in the graph object. Use this command to modify tick marks, change the font size of labels, turn on grid or zero lines, or duplicate axes.
Assigning data to an axis

In most cases when a graph is created, all data elements are initially assigned to the left axis. XY graphs differ slightly in that data elements are initially assigned to either the left and bottom axes (these defaults may be changed using the global Graph Options dialog).

Once a graph is created, individual elements may generally be assigned to either the left or right axis. In XY graphs, you may reassign individual elements to either the left, right, top, or bottom axis, while in boxplots or stacked time/observation graph all data elements must be assigned to the same vertical axis.

To assign a data element to a different axis, use the setelem command with the axis option. For example, the commands

```
graph graph02.line ser1 ser2
graph02.setelem(2) axis(right)
```

first create GRAPH02, a line graph of SER1 and SER2, and then turn GRAPH02 into a dual scaled graph by assigning the second data element, SER2, to the right axis.

In this example, GRAPH02 uses the default setting for dual scale graphs by disallowing crossing, so that the left and right scales do not overlap. To allow the scales to overlap, use the scale command with the overlap option, as in

```
graph02.scale overlap
```

The left and right scales now span the entire axes, allowing the data lines to cross.

For XY graphs without pairing, the first series is generally plotted along the bottom axis, and the remaining series are plotted on the left axis. XY graphs allow more manipulation than time/observation plots, because the top and bottom axes may also be assigned to an element. For example,

```
graph graph03.xyline s1 s2 s3 s4
graph03.setelem(1) axis(top)
graph03.setelem(2) axis(right)
```

first creates an XY line graph GRAPH03 of the series S1, S2, S3, and S4. The first series is then assigned to the top axis, and the second series is moved to the right axis. Note that the graph now uses three axes: top, left, and right.

Note that the element index in the setelem command is not necessary for boxplots and stacked time/observation graphs, since all data elements must be assigned to the same axis.

While EViews allows dual scaling for the vertical axes, the horizontal axes must use a single scale on either the top or bottom axis. When a new element is moved to or from one of
the horizontal axes, EViews will, if necessary, reassign elements as required so that there is a single horizontal scale.

For example, using the graph created above, the command

\begin{verbatim}
graph03.setelem(3) axis(bottom)
\end{verbatim}

moves the third series to the bottom axis, forcing the first series to be reassigned from the top to the left axis. If you then issue the command,

\begin{verbatim}
graph03.setelem(3) axis(right)
\end{verbatim}

EViews will assigns the third series to the right axis as directed, with the first (next available element, starting with the first) series taking its place on the horizontal bottom axis. If the first element is then moved to a vertical axis, the second element will take its place on the horizontal axis.

For XY graphs with pairing, the same principles apply. However, since the elements are graphed in pairs, there is a set of elements that should be assigned to the same horizontal axis. You can switch which set is assigned to the horizontal using the `axis` keyword. For example,

\begin{verbatim}
graphe graph04.xypair s1 s2 s3 s4
graph03.setelem(2) axis(left)
\end{verbatim}

creates an XY graph that plots the series S1 against S2, and S3 against S4. Usually, the default settings assign the first and third series are to the left axis, and the second and fourth series to the bottom axis. The second command line moves the second series from the bottom to the left axis. Since S2 and S4 are tied to the same axis, the fourth series will be assigned to the left axis. The first and third series will take their place on the bottom axis.

**Modifying the data axis**

The `scale` (p. 397) command may be used to change the way data is scaled on an axis. To rescale the data, specify the axis you wish to change and use one of the following options: `linear`, `linearzero` (linear with zero included in axis), `log` (logarithmic), `norm` (standardized). For example,

\begin{verbatim}
graphe graph05.line ser1 ser2
graph05.scale(left) log
\end{verbatim}

creates a line graph GRAPH05 of the series SER1 and SER2, and changes the left axis scaling method to logarithmic.

The interaction of the data scales (these are the left and right axes for non-XY graphs) can be controlled using `scale` with the `overlap` option keyword. The `overlap` option con-
trols the overlap of vertical scales, where each scale has at least one series assigned to it. For instance,

```
  graph graph06.line s1 s2
  graph06.setelem(2) axis(right)
  graph06.scale overlap
```

first creates GRAPH06, a line graph of series S1 and S2, and assigns the second series to the right axis. The last command allows the vertical scales to overlap.

The `scale` command may also be used to change or invert the endpoints of the data scale, using the `range` or `invert` options:

```
  graph05.scale(left) invert range(minmax)
```

inverts the left scale of GRAPH05 and sets its endpoints to the minimum and maximum values of the data.

**Modifying the date/time axis**

EViews automatically determines an optimal set of labels for the bottom axis of time plots. If you wish to modify the frequency or date format of the labels, you should use the `datelabel` command. When working with boxplots, you should use `bplabel` instead as described in “Modifying the boxplot axis” beginning on page 80.

To control the number of observations between labels, use `datelabel` with the `interval` option to specify a desired step size. The stand-alone step size keywords include: `auto` (use EViews’ default method for determining step size), `ends` (label first and last observations), and `all` (label every observation). For example,

```
  mygraph.datelabel interval(ends)
```

labels only the endpoints of MYGRAPH. You may also use a step size keyword in conjunction with a step number to further control the labeling. These step size keywords include: `obs` (one observation), `year` (one year), `m` (one month), `q` (one quarter), where each keyword determines the units of the number specified in the step option. For example, to label every ten years you may specify

```
  mygraph.datelabel interval(year, 10)
```

In addition to specifying the space between labels, you may indicate a specific observation to receive a label. The step increment will then center around this observation. For example,

```
  mygraph.datelabel interval(obs, 10, 25)
```

labels every tenth observation, centered around the twenty-fifth observation.
You may also use `datelabel` to modify the format of the dates or change their placement on the axis. Using the `format` or `span` options,

```
mygraph02.datelabel format(yy) -span
```

formats the labels so that they display as two digit years, and disables interval spanning. If interval spanning is enabled, labels will be centered between the applicable tick marks. If spanning is disabled, labels are placed directly on the tick marks. For instance, in a plot of monthly data with annual labeling, the labels may be centered over the twelve monthly ticks (spanning enabled) or placed on the annual tick marks (spanning disabled).

**Customizing axis appearance**

You may customize the appearance of tick marks, modify label font size, add grid lines, or duplicate axes labeling in your graph using `axis` (p. 197).

Follow the `axis` keyword with a descriptor of the axis you wish to modify and one or more options. For instance, using the `ticksin`, `minor`, and `font` keywords:

```
mygraph.axis(left) ticksin -minor font(10)
```

The left axis of MYGRAPH is now drawn with the tick marks inside the graph, no minor ticks, and a label font size of 10 point.

To add lines to a graph, use the `grid` or `zeroline` options:

```
mygraph01.axis(left) -label grid zeroline
```

MYGRAPH01 hides the labels on its left axis, draws horizontal grid lines at the major ticks, and draws a line through zero on the left scale.

In single scale graphs, it is sometimes desirable to display the axis labels on both the left and right hand sides of the graph. The `mirror` option may be used to turn on or off the display of duplicate axes. For example,

```
graph graph06.line s1 s2
graph06.setelem(1) axis(left)
graph06.setelem(2) axis(left)
graph06.axis mirror
```

turns on mirroring of the left axis to the right axis of the graph. Note that you need not specify an axis to modify since mirroring sets both the left and right axes to be the same.

If dual scaling is enabled, mirroring will be overridden. In our example, assigning a data element to the right axis,
graph06.setelem(1) axis(right)
will override axis mirroring. Note that if the element 1 is subsequently reassigned to the
test graph to the left scale, mirroring will again be enabled. To turn off mirroring entirely, simply precede
the mirror keyword with a minus sign. The command

graph06.axis -mirror

turns off axis mirroring.

Customizing the graph frame

The graph frame is used to set the basic graph proportions and display characteristics that
are not part of the main portion of the graph.

Graph size

The graph frame size and proportions may be modified using the options (p. 343) com-
mmand. Simply specify a width and height using the size keyword. For example,

testgraph.options size(5,4)
resizes the frame of TESTGRAPH to 5 x 4 virtual inches.

Other frame characteristics

The options command can be used to remove the box around the graph frame. Use the
inbox keyword:

testgraph.options -inbox

Note that removing the box also removes the yellow background from the graph.

By default, EViews occupies the entire width of the graph for plotting data. If you wish to
indent the data in the graph frame (for line, bar, area, or spike graphs), use the indent
keyword:

testgraph.options indent

Labeling data values

Bar and pie graphs allow you to label the value of your data within the graph. Use the
options (p. 343) command with one of the following keywords: barlabelabove, bar-
labelinside, piechart. For example,

mybargraph.options barlabelabove

places a label above each bar in the graph indicating its data value. Note that the label will
be visible only when there is sufficient space in the graph.
Outlining and spacing filled areas

By default, EViews draws a black outline around each bar or area in a bar or area graph, respectively. To disable the outline, use options with the outlinebars or outlineareas keywords:

```
mybargraph.options -outlinebars
```

Disabling the outline is useful for graphs whose bars are spaced closely together, enabling you to see the fill color instead of an abundance of black outlines.

EViews attempts to place a space between each bar in a bar graph. This space disappears as the number of bars increases. You may remove the space between bars by using the barspace keyword:

```
mybargraph.options -barspace
```

Modifying the Legend

A legend’s location, text, and appearance may be customized. Note that special graph types such as box plots and histograms use text objects for labeling instead of a legend. These text objects may only be modified interactively by double-clicking on the object to bring up the text edit dialog.

To change the text string of a data element for use in the legend, use the `name` command:

```
graph graph06.line ser1 ser2
graph06.name(1) Unemployment
graph06.name(2) DMR
```

The first line creates a line graph GRAPH06 of the series SER1 and SER2. Initially, the legend shows "SER1" and "SER2." The second and third command lines change the text in the legend to "Unemployment" and "DMR."

Note that the `name` command is equivalent to using the `setelem` command with the `legend` option. For instance,

```
graph06.setelem(1) legend(Unemployment)
graph06.setelem(2) legend(DMR)
```

produces the same results.

To remove a label from the legend, you may use `name` without providing a text string:

```
graph06.name(2)
```

removes the second label "DMR" from the legend.
For an XY graph, the name command modifies any data elements that appear as axis labels, in addition to legend text. For example,

```
graph xygraph.xy ser1 ser2 ser3 ser4
xygraph.name(1) Age
xygraph.name(2) Height
```

creates an XY graph named XYGRAPH of the four series SER1, SER2, SER3, and SER4. “SER1” appears as a horizontal axis label, while “SER2”, “SER3”, and “SER4” appear in the legend. The second command line changes the horizontal label of the first series to “Age”. The third line changes the second series label in the legend to “Height”.

To modify characteristics of the legend itself, use `legend` (p. 304). Some of the primary options may be set using the `inbox`, `position` and `columns` keywords. Consider, for example, the commands

```
graph graph07.line s1 s2 s3 s4
graph07.legend -inbox position(botleft) columns(4)
```

The first line creates a line graph of the four series S1, S2, S3, and S4. The second line removes the box around the legend, positions the legend in the bottom left corner of the graph window, and specifies that four columns should be used for the text strings of the legend.

When a graph is created, EViews automatically determines a suitable number of columns for the legend. A graph with four series, such as the one created above, would display two columns of two labels each. The `columns` command above, with an argument of four, creates a long and slender legend, with each of the four series in its own column.

You may also use the `legend` command to change the font size or to disable the legend completely:

```
graph07.legend font(10)
graph07.legend -display
```

Note that if the legend is hidden, any changes to the text or position of the legend remain, and will reappear if the legend is displayed again.

**Adding text to the graph**

Text strings can be placed anywhere within the graph window. Using the `addtext` (p. 183) command,

```
graph07.addtext(t) Fig 1: Monthly GDP
```

adds the text “Fig 1: Monthly GDP” to the top of the GRAPH07 window. You can also use specific coordinates to specify the position of the upper left corner of the text. For example,
adds the text string “Figure 1” to GRAPH08. The text is placed 0.2 virtual inches in, and 0.1 virtual inches down from the top left corner of the graph frame. The “x” option instructs EViews to place the text object inside a box.

An exiting text object can be edited interactively by double-clicking on the object to bring up a text edit dialog. The object may be repositioned by specifying new coordinates in the dialog, or by simply dragging the object to its desired location.

Adding lines and shading

You may wish to highlight or separate specific areas of your graph by adding a line or shaded area to the interior of the graph using the `draw` (p. 252) command. Specify the type of line or shade option (line, dashline, or shade), which axis it should be attached to, (left, right, bottom, top) and its position. For example,

```
graph09.draw(line, left) 5.2
```
draws a horizontal line at the value 5.2 on the left axis. Alternately,

```
graph09.draw(shade, left) 4.8 5.6
```
draws a shaded horizontal area bounded by the values 4.8 and 5.6 on the left axis. You can also specify `color` and `line width`:

```
graph09.draw(dashline, bottom, color(blue), width(2)) 1985:1
```
draws a vertical blue dashed line of width two points at the date “1985:1” on the bottom axis. Color may be specified using one or more of the following options: `rgb(n1, n2, n3)`, `color(n1, n2, n3)`, where the arguments correspond to RGB settings, or `color(keyword)`, where `keyword` is one of the predefined color keywords.

Using graphs as templates

After customizing a graph as described above, you may wish to use your custom settings in another graph. Using a graph template allows you to copy the graph type, line and fill settings, axis scaling, legend attributes, and frame settings of one graph into another. This enables a graph to adopt all characteristics of another graph—everything but the data itself. To copy only custom line or fill settings, use the `preset` or `default` options of the `setelem` command (as described in “Using preset lines and fills” on page 67).

Modifying an existing graph

To modify a named graph object, use the `template` command:
Customizing a Graph—77

```
graph10.template customgraph
```

This command copies all the appearance attributes of CUSTOMGRAPH into GRAPH10, including the graph type and axis scaling. Note that this may produce undesirable results if copying customized axis settings from a graph whose data does not closely match the target graph’s data.

To copy text labels, lines and shading in the template graph in addition to all other option settings, use the “t” option:

```
graph10.template(t) customgraph
```

This command copies any text or shading objects added to the template graph using the addtext or draw commands. Note that using the “t” option overwrites any existing text and shading objects in the target graph.

If you are using a box plot as a template for another graph type, or vice versa, note that the graph types and box plot specific attributes will not be changed. In addition, when the “t” option is used, vertical lines or shaded areas will not be copied between the graphs, since the horizontal scales of the two graphs are different.

Using a template during graph creation

All graph type commands also provide a template option for use when creating a new graph. For instance,

```
graph mygraph.line(o = customgraph) ser1 ser2
```

creates the graph MYGRAPH of the series SER1 and SER2, using CUSTOMGRAPH as a template. Note that the graph type keyword (line, in this case) is irrelevant, as the graph type will be that of the template graph. See “Customizing a Graph” on page 64, for details on which attributes are copied from a template graph. To include text, lines and shading in the copy, use the “t” option in place of the “o” option.

When used as a graph procedure, this method is equivalent to the one described above for an existing graph:

```
graph10.template(t) customgraph
graph10.bar(t = customgraph)
```

These two methods produce the same results. Note, again that the bar keyword in the second line is irrelevant.

Arranging multiple graphs

When you create a multiple graph, EViews automatically arranges the graphs within the graph window. See “Creating a Graph” on page 59 for information on how to create a mul-
Multiple graph. You may use either the “m” option during graph creation or the `merge` command.

To change the placement of the graphs, use the `align` (p. 184) command. Specify the number of columns in which to place the graphs and the horizontal and vertical space between graphs, measured in virtual inches. For example,

```
graph graph11.merge graph01 graph02 graph03
graph11.align(2, 1, 1.5)
```

creates a multiple graph `GRAPH11` of the graphs `GRAPH01`, `GRAPH02`, and `GRAPH03`. By default, the graphs are stacked in one column. The second command re-aligns the graphs in two columns, with 1 virtual inch between the graphs horizontally and 1.5 inches between the graphs vertically.

**Modifying Boxplots**

The appearance of boxplots can be customized using many of the commands described above. A few special cases and additional commands are described below.

**Customizing lines and symbols**

As with other graph types, the `setelem` command can be used with box plots to modify line and symbol attributes, assign the boxes to an axis, and use preset and default settings. To use the `setelem` (p. 412) command with boxplots, use a box element keyword after the command. For example,

```
boxgraph01.setelem(mean) symbol(filledcircle)
```

changes the means in the box plot `BOXGRAPH01` to filled circles. Note that all boxes within a single graph have the same attributes, and changes to appearance are applied to all boxes. For instance,

```
boxgraph01.setelem(box) lcolor(orange) lpat(dash1) lwidth(2)
```

plots all boxes in `BOXGRAPH01` with an orange dashed line of width 2 points. Also note that when shaded confidence intervals are used, a lightened version of the box color will be used for the shading. In this way, the above command also changes the confidence interval shading to a light orange.

**Assigning boxes to an axis**

The `setelem` command may also be used to assign the boxes to another axis:

```
boxgraph01.setelem axis(right)
```

Note that since all boxes are assigned to the same axis, the index argument specifying a graph element is not necessary.
Using preset line colors

During general graph creation, lines and fills take on the characteristics of the user-defined presets. When a boxplot is created, the first user-defined line color is applied to the box, whiskers, and staples. Similarly, when you use the preset or default options of the setelem command with a boxplot, the line color of the preset is applied to the box, whiskers, and staples. See “Using preset lines and fills” on page 67 for a description of presets.

The preset and default methods work just as they do for other graph types, although only the line color is applied to the graph. For example,

```
boxgraph01.setelem default(3)
```

applies the line color of the third user-defined line preset to the box, whiskers, and staples of BOXGRAPH01. Note again that setelem does not require an argument specifying an index, since the selected preset will apply to all boxes.

There are a number of setelem arguments that do not apply to box plots. The fill-color, fillgray, and fillhatch option keywords are not available, as there are no custom areas to be filled. The legend option is also not applicable, as boxplots use axis text labels in place of a legend.

Hiding boxplot elements

In addition to the setelem command, boxplots provide a setbpelem (p. 407) command for use in enabling or disabling specific box elements. Any element of the box plot can be hidden, except the box itself. Use the command with a list of box elements to show or hide. For example,

```
boxgraph01.setbpelem -mean far
```

hides the means and confirms that the far outliers are shown in BOXGRAPH01.

Modifying box width and confidence intervals

The width of the individual boxes in a boxplot can be drawn in three ways: fixed width over all boxes, proportional to the sample size, or proportional to the square root of the sample size. To specify one of these methods, use the setbpelem command with the width option, and one of the supported keywords (fixed, rootn, n). For example,

```
boxgraph01.setbpelem width(rootn)
```

draws the boxes in BOXGRAPH01 with widths proportional to the square root of their sample size.

There are three methods for displaying the confidence intervals in boxplots. They may be notched, shaded, or not drawn at all, which you may specify using one of the supported keywords (notch, shade, none). For example,
boxgraph01.setbpelem ci(notch)
draws the confidence intervals in BOXGRAPH01 as notches.

Modifying the boxplot axis
The bplabel command for boxplots is similar to the datelabel command in time plots (see “Modifying the date/time axis” beginning on page 71). Use it to specify the frequency or alignment of labels on the horizontal axis. Provide the step method you wish to use, and optionally a step or alignment number. For example,

boxgraph02.bplabel interval(ends)

labels the first and last boxplot in BOXGRAPH02. To label every fiftieth box, aligned with the tenth box:

boxgraph02.bplabel interval(cust, 50, 10)

A boxplot’s horizontal scale can be modified further by placing the text labels at an angle, changing the text itself, or selectively hiding any of the labels. Using the bplabel command,

boxgraph02.bplabel angle(60) label(10, "First") -label(60)

sets the labels in BOXGRAPH2 to a 60 degree angle, changes the tenth box label to “First”, and hides the sixtieth label.

Labeling Graphs
As with all EViews objects, graphs have a label view to display and edit information such as the graph name, last modified date, and remarks. To modify or view the label information, use the label command:

graph12.label(r) Data from CPS 1988 March File

This command shows the label view, and the “r” option appends the text “Data from CPS 1988 March File” to the remarks field of GRAPH12.

To return to the graph view, use the graph keyword:

graph12.graph

All changes made in label view will be saved when the graph is saved.

Printing Graphs
A graph may be printed using the print command. For example,
print graph11 graph12

prints GRAPH11 and GRAPH12 on a single page.

In addition, many graph commands and graph views of objects include a print option. For example, you can create and simultaneously print a line graph GRA1 of SER1 using the “p” option:

```
graph gral.line(p) ser1
```

You should check the individual commands for this availability of this option.

Exporting Graphs to Files

You may use the `save` (p. 393) proc of a graph object to save the graph as a Windows metafile (.wmf), Enhanced Windows metafile (.emf), or Postscript file (.eps).

You must specify a file name and file type, and may also provide the file height, width, units of measurement, and color use. Postscript files also allow you to save the graph with or without a bounding box and to specify portrait or landscape orientation. For instance,

```
graph11.save(t=postscript, u=cm, w=12, -box) MyGraph1
```

saves GRAPH11 in the default directory as a Postscript file MyGraph1.eps, with a width of 12 cm and no bounding box. The height is determined by holding the aspect ratio of the graph constant. Similarly,

```
graph11.save(t=emf, u=pts, w=300, h=300, -c) c:\data\MyGraph2
```

saves GRAPH11 as an Enhanced Windows metafile MYGRAPH2.EMF. The graph is saved in black and white, and scaled to 300 × 300 points.

Graph Summary

See “Graph” (p. 143) for a full listing of procs that may be used to customize graph objects, and for a list of the graph type commands.
Chapter 6. EViews Programming

EViews’ programming features allow you to create and store commands in programs that automate repetitive tasks, or generate a record of your research project.

For example, you can write a program with commands that analyze the data from one industry, and then have the program perform the analysis for a number of other industries. You can also create a program containing the commands that take you from the creation of a workfile and reading of raw data, through the calculation of your final results, and construction of presentation graphs and tables.

If you have experience with computer programming, you will find most of the features of the EViews language to be quite familiar. The main novel feature of the EViews programming language is a macro substitution language which allows you to create object names by combining variables that contain portions of names.

Program Basics

Creating a Program

A program is not an EViews object within a workfile. It is simply a text file containing EViews commands. To create a new program, click File/New/Program. You will see a standard text editing window where you can type in the lines of the program. You may also open the program window by typing program in the command window, followed by an optional program name. For example

```
program firstprg
```

opens a program window named FIRSTPRG. Program names should follow standard EViews rules for file names.

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.

If a program line is longer than the current program window, EViews will autowrap the text of the line. Autowrapping alters the appearance of the program line by displaying it on multiple lines, but does not change the contents of the line. While resizing the window will change the autowrap position, the text remains unchanged and is still contained in a single line.

If you wish to have greater control over the appearance of your lines, you can manually break long lines using the ENTER key, and then use the underscore continuation character
“_” as the last character on the line to join the multiple lines. For example, the three separate lines

```
equation eq1.ls _
y x c _
ar(1) ar(2)
```

are equivalent to the single line

```
equation eq1.ls y x c ar(1) ar(2)
```

formed by joining the lines at the continuation character.

**Saving a Program**

After you have created and edited your program, you will probably want to save it. Press the **Save** or **SaveAs** button on the program window toolbar. When saved, the program will have the extension .PRG.

**Opening a Program**

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. By default, EViews will look for the program in the default directory. If appropriate, include the full path to the file. The entire name should be enclosed in quotations if necessary. For example:

```
open mysp500.prg
open "c:\mywork is here\eviews\myhouse.prg"
```

opens MYSP500.PRG in the default directory, and MYHOUSE.PRG in the directory “C:\MYWORK IS HERE\EVIEWS”.

**Executing a Program**

When you enter, line by line, a series of commands in the command window, we say that you are working in *interactive mode*. Alternatively, you can type all of the commands in a program and execute or run them collectively as a batch of commands. When you execute or run the commands from a program, we say that you are in *program* (non-interactive) *mode*. 
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.

If the checkbox labeled Version 4 compatible variable substitution is selected, EViews will use the variable substitution behavior found in EViews 4 and earlier. To support the use of alpha series, EViews 5 has changed the way that % substitution variables are evaluated in expressions. To return to EViews 4 compatible rules for substitution, you may either use this checkbox or include a "MODE=VER4" statement in your program. See “Version 5 Compatibility Notes” on page 93 and “Program Modes” on page 96 for additional discussion.

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. If there is a serious error so that it is impractical for EViews to continue, the program will halt even if the maximum number of errors is not reached. See “Handling Execution Errors” on page 106.

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
```
The use of the .PRG extension is not required since EViews will automatically append one. The default run options described above are taken from the global settings, but may be overridden using command options or program options.

For example, you may use the “v” or “verbose” options to run the program in verbose mode, and the “q” or “quiet” options to run the program in quiet mode. If you include a
number as an option, EViews will use that number to indicate the maximum number of errors encountered before execution is halted. Any arguments to the program may be listed after the filename:

```
run(v, 500) mysp500
```

or

```
run(q) progarg arg1 arg2 arg3
```

Alternatively, you may modify your program to contain program options for quiet and verbose mode, and for setting version compatibility.

You can also have EViews run a program automatically upon startup by choosing **File/Run** from the menu bar of the Windows Program Manager or **Start/Run** in Windows and then typing “eviews”, followed by the name of the program and the values of any arguments. If the program has as its last line the command `exit`, EViews will close following the execution of the program.

See “Multiple Program Files” on page 107 for additional details on program execution.

**Stopping a Program**

Pressing the ESC or F1 keys halts execution of a program. It may take a few seconds for EViews to respond to the halt command.

Programs will also stop when they encounter a `stop` command, when they read the maximum number of errors, or when they finish processing a file that has been executed via a `run` statement.

If you include the `exit` keyword in your program, the EViews application will close.

**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.

While you could execute the commands by typing them in the command window, you could just as easily open a program window, type in the commands and click on the **Run** button. Entering commands in this way 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 (the data series are provided in the database PROGDEMO in your EViews directory so that you can try out the program). 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, fetch a series from an EViews database named PROGDEMO, run a regression, compute residuals and a forecast, make a plot of the forecast, and save the results.

' housing analysis
workfile myhouse m 1968m3 1997m6
fetch progdemo::hsf
smpl 1968m5 1992m12
equation reg1.ls hsf c hsf(-1)
reg1.makeresid hsfres
smpl 1993m1 1997m6
reg1.forecast hsffit
freeze(hsfplot) hsffit.line
save

The first line of the program is a comment, as denoted by the apostrophe “‘”. In executing a program, EViews will ignore all text following the apostrophe until the end of the line.

HSF is total housing units started. We end up with a saved workfile named MYHOUSE containing the HSF series, an equation object REG1, a residual and forecast series, HSFRES and HSFIT, and a graph HSFLOT of the forecasts.

You can run this program by clicking on Run and filling in the dialog box.

Now, suppose you wish to perform the same analysis, but for the S&P 500 stock price index (FSPCOM). Edit the program, changing MYHOUSE to MYSP500, and change all of the references of HSF to FSPCOM:

' s&p analysis
workfile mysp500 m 1968m3 1997m6
fetch progdemo::fspcom
smpl 1968m5 1992m12
equation reg1.ls fspcom c fspcom(-1)
reg1.makeresid fspcomres
smpl 1993m1 1997m6
reg1.forecast fspcomfit
freeze(fscomplot) fspcomfit.line
save

Click on Run to execute the new analysis. Click on the Save button to save your program file as MYPROG.PRG in the EViews directory.
Since most of these two programs are identical, it seems like a lot of typing to make two separate programs. In “Program Arguments” on page 97 we show you a method for handling these two forecasting problems with a single program. First however, we must define the notion of variables that exist solely when running programs.

**Program Variables**

While you can use programs just to edit, run, and re-run collections of EViews commands, the real power of the programming language comes from the use of program variables and program control statements.

**Control 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:

```
!x
!1
!counter
```

You need not 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
!12345 = 0
!counter = 12
!pi = 3.14159
```

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

```
!counter = !counter + 1
genr dnorm = 1/sqr(2*pi)*exp(-1/2*epsilon^2)
scalar stdx = x/sqr(!varx)
smpl 1950:1+!i 1960:4+!i
```

Control variables do not exist outside of your program and are automatically erased 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 commands

```eviews
scalar stdx = sq(!varx)
c(100) = !length
sample years 1960+!z  1990
```

use the numeric values assigned to the control variables !VARX, !LENGTH, and !Z.

**String Variables**

A *string expression* or *string* is text enclosed in double quotes:

```
"gross domestic product"
"3.14159"
"ar(1) ar(2) ma(1) ma(2)"
```

A *string variable* is a variable whose value is a string of text. String variables, which may only exist during the time that your program is executing, have names that begin with a “%” symbol. String variables are assigned by putting the string variable name on the left of an “=” sign and a string expression on the right. For example, the following lines assign values to string variables

```
%value = "value in millions of u.s. dollars"
%armas = "ar(1) ar(2) ma(1) ma(2)"
%mysample = " 83m1 96m12"
%dep = " hs"
%pi = " 3.14159"
```

You may use strings variables to help you build up command text, variable names, or other string values. EViews provides a number of operators and functions for manipulating strings; a complete list is provided in “Strings” on page 867 of the User’s Guide.

Once assigned a value, a string variable may appear in any expression in place of the underlying string. When substituted for, the string variable will be replaced by the contents of the string variable, enclosed in double quotes.

Here is a quick example where we use string operations to concatenate the contents of three string variables.

```
!repeat = 500
%st1 = " draws from the normal"
%st2 = "Cauchy "
%st3 = @str(!repeat) + @left(%st1,16) + %st2 + "distribution"
```
In this example %ST3 is set to the value “500 draws from the Cauchy distribution”. Note the spaces before “draws” and after “Cauchy” in the string variable assignments. After string variable substitution, the latter assignment is equivalent to entering

%st3 = "500" + " draws from the " + "Cauchy " + "distribution"

Similarly, the table assignment statement

table1(1,1) = %st3

is equivalent to entering the command

table(1,1) = "500 draws from the Cauchy distribution"

One important usage of string variables is in assigning string values to alpha series. For example, we may have the assignment statement

%z = "Ralph"
alpha full_name = %z + last_name

which is equivalent to the expression

alpha full_name = "Ralph" + last_name

We again emphasize that string variable substitution involves replacing the string variable by its string value contents enclosed in double quotes.

As with any string value, you may convert a string variable containing a number into a number by using the @val function. For example,

%str = ".05"
!level = @val(%str)

creates a control variable !LEVEL=0.05. If the first character of the string is not a numeric character (and is not a plus or a minus sign), @val returns the value “NA”. Any characters to the right of the first non-digit character are ignored. For example,

%date = "04/23/97"
scalar day = @val(@right(%date,5))
scalar month = @val(%date)

creates scalar objects DAY=23 and MONTH=4.

Full details on working with strings are provided in “Strings” on page 867 in the User’s Guide.
Replacement Variables

When working with EViews commands, you may wish to use a string variable, not simply to refer to a string value, but as an indirect way of referring to something else, perhaps a command, or a name, or portion of names for one or more underlying items.

Suppose, for example, that the string variable \%X contains the string “GDP”,

\%x = "gdp"

We may be interested, not in the actual string value “gdp”, but rather in indirectly referring to an underlying object named “GDP”. For example, the series declaration

series \%x = 300.2

will generate an error since the substituted expression is

series "gdp" = 300.2

and the series declaration series expects the name of a series, not a string value. In this circumstance, we would like to replace the string variable with the underlying name held in the string.

If you enclose a string variable in curly braces (“{“ and “}”) EViews will replace the expression with the name, names, 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

equation eq1.ls {%x} c {%x}(-1)

would be interpreted by EViews as

equation eq1.ls gdp c gdp(-1)

Changing the contents of \%X to “M1” changes the interpretation of the original line to

equation eq1.ls m1 c m1(-1)

since the replacement variable uses the name obtained from the new \%X.

Similarly, when trying to find the number of valid (non-missing) observations in a series named INCOME, you may use the @obs function along with the name of the series:

@obs(income)

If you wish to use a string variable \%VAR to refer to the INCOME series, you must use the replacement variable in the @OBS function, as in

\%var = "income"
since you wish to refer indirectly to the object named in %VAR. Note that the expression

\[ @\text{obs}(%\text{var}) \]

will return an error since \texttt{@OBS} requires a series or matrix object name as an argument.

Any string variable may be used as the basis of a replacement variable. Simply form your string using one or more string operations:

\[
%\text{object} = \text{"group"}
%\text{space} = \text{"\"}
%\text{reg1} = \text{"gender"}
%\text{reg2} = \text{"income"}
%\text{reg3} = \text{"age"}
%\text{regs} = %\text{reg1} + %\text{space} + %\text{reg2} + %\text{space} + %\text{reg3}
\]

then enclose the string variable in braces. In the expression,

\[
\{ %\text{object} \} g1 \{ %\text{regs} \}
\]

EViews will substitute the names found in \%OBJECT and \%REGS so that the resulting command is

\[
group g1 gender income age
\]

It is worth noting that replacement variables may be used as building blocks to form object names. For example, the commands

\[
%b = \text{"2"}
%c = \text{"temp"}
series z{%b}
matrix(2, 2) x{%b}
vector(3) x_{%c}_y
\]

declares a series named Z2, a 2 × 2 matrix named X2, and a vector named X_TEMP_Y.

Up until now we have focused on replacement variables formed from string variables. Note however, that control variables may also be used as replacement variables. For example, the commands

\[
!i = 1
series y{!x} = \text{nrand}
!j = 0
series y{!j}{!i} = \text{nrand}
\]
are equivalent to

```
series y1 = nrnd
series y01 = nrnd
```

and will create two series Y1 and Y01 that contain a set of (pseudo-)random draws from a standard normal distribution. Note that in cases where there is no possibility of ambiguity, EViews will treat a control variable as a replacement variable, even if the braces are not provided. For example,

```
!x = 3
series y!x = 3
```

will generate the series Y3 containing the value 3.

Replacement variables provide you with great flexibility in naming objects in your programs. We caution, however, that unless you take some care, they may cause considerable confusion. We suggest, for example, that you avoid using the same base names to refer to different objects. Consider the following program:

```
' possibly confusing commands (avoid)
!a = 1
series x{!a}
!a = 2
matrix x{!a}
```

In this small code snippet it is easy to see that X1 is the series and X2 is the matrix. But in a more complicated program, where the control variable assignment !A = 1 may be separated from the declaration by many program lines, it may be difficult to tell at a glance what kind of object X1 represents. A better approach might be to use different names for different kinds of variables:

```
!a = 1
series ser{!a}
!a = 2
matrix mat{!a}
```

so that the replacement variable names are more apparent from casual examination of the program.

**Version 5 Compatibility Notes**

While the underlying concepts behind string and replacement variables have not been changed since the first version EViews, beginning in EViews 5 there are two important changes in the implementation of these concepts.
String vs. Replacement Variables

First, the use of contextual information to distinguish between the use of string and replacement variables has been eliminated.

Previously, the underlying notion that the expression “%X” refers exclusively to the string variable %X while the expression “{%X}” refers to the corresponding replacement variable was modified slightly to fit the context in which the expression was used. In earlier versions of EViews, the string variable expression “%X” was treated as a string variable in cases where a string was expected, but was treated as a replacement variable in other settings.

For example, suppose that we have the string variables

```plaintext
%y = "cons"
%x = "income"
```

When used in settings where a string is expected, all versions of EViews treat %X and %Y as string variables. Thus, in table assignment,

```plaintext
table1(2, 3) = %x + " " + %y
```

is equivalent to the expression

```plaintext
table1(2, 3) = "cons" + " " + "income"
```

However, when string variables were used in other settings, earlier versions of EViews used the context to determine that the string variable should be treated as a replacement variable; for example, the three commands

```plaintext
equation eq1.ls %y c %x
equation eq1.ls {%y} c {%x}
equation eq1.ls cons c income
```

were all equivalent. Strictly speaking, the first command should have generated an error since string variable substitution would replace %Y with the double-quote delimited string “cons” and %X with the string “income”, as in

```plaintext
equation eq1.ls "cons" c "income"
```

Instead, earlier versions of EViews determined that the only valid interpretation of %Y and %X in the first command was as replacement variables so EViews simply substitutes names for %Y and %X.

Similarly, the commands

```plaintext
genr %y = %x
genr {%y} = {%x}
```
all yielded the same result, since %Y and %X were treated as replacement variables not as string variables.

This contextual interpretation of string variables was convenient since, as seen from the examples above, it meant that users rarely needed to use braces around string variables. The EViews 5 introduction of alphanumeric series meant, however, that the existing interpretation of string variables was no longer valid. The following example shows clearly the problem:

```
alpha parent = "mother"
%x = "parent"
alpha temp = %x
```

Note that in the final assignment statement, the command context alone is not sufficient to determine whether %X should refer to the string variable value “parent”, or to the replacement variable (alpha series) PARENT containing the string “mother”.

In the EViews 5 interpretation of string and replacement variables, users must now always use the expression “{%X}” to refer to the substitution variable corresponding to %X so that the final line above resolves to the command

```
alpha temp = "parent"
```

To interpret the line as a replacement variable, you must enter

```
alpha temp = {%x}
```

which resolves to the command

```
alpha temp = parent
```

Under an EViews 4 and earlier interpretation of the final line, the braces would not be necessary since “%X” would be treated as a replacement variable.

**String Variables in String Expressions**

The second major change in EViews 5 is that all text in a string expression is treated as a literal string. The important implication of this rule is that string variable text is no longer substituted for inside of a string expression.

Consider the assignment statements

```
%b = "mom!"
%a = "hi %b"
table(1, 2) = %a
```
In EViews 4 and earlier, the “%B” text in the string expression was treated as a string variable, not as literal text. Accordingly, the string variable %A contains the text “hi mom!”. One consequence of this approach was that there was no way to get the literal text of the form “%B” into a string using a program.

Beginning in EViews 5, the “%B” in the second string variable assignment is treated as literal text. The string variable %A will contain the text “hi %b”. Obtaining a %A that contains the EViews 4 result is straightforward. Simply move the first string variable %B outside of the string expression, and use the string concatenation operator:

\[
%a = "hi " + %b
\]

assigns the text “hi mom!” to the string variable %A.

Version 4 Compatibility Mode

While the changes to the handling of string variables are important for extending the programming language to handle the new features of the EViews 5, we recognize that users may have a large library of existing programs which make use of the previous behavior. Accordingly, EViews 5 provides a Version 4 compatibility mode in which you may run EViews programs using the previous context sensitive handling of string and substitution variables, and the previous rules for resolving string variables in string expressions.

There are two ways to ensure that your program is run in Version 4 compatibility mode. First, you may specify Version 4 compatibility mode at the time the program is run. Compatibility may be set interactively from the Run Program dialog (“Executing a Program” on page 84) by selecting the Version 4 compatible variable substitution checkbox, or in a program using the “ver4” option (see run (p. 391)).

Alternatively, you may include “MODE=VER4” statement in your program. See “Program Modes” on page 96 for details.

Program Modes

EViews provides you with the opportunity to set program execution modes at the time that the program is first run. In addition, you may use the “MODE=” statement to change the execution mode of a program from within the program itself. One important benefit to using “MODE=” statements is that the program can begin executing in one mode, and switch to a second mode as the program executes.

Mode options are available for turning on quiet or verbose mode, or for switching between Version 4 and Version 5 compatibility.

To change the mode for quiet or verbose mode, simply add a line to your program reading “MODE=” followed by either the “QUIET” or the “VERBOSE” keyword, as in
mode = quiet

For version 4 and 5 compatibility, you should use the keywords “VER4” and “VER5”. To use version 4 compatibility mode, you may specify

mode = ver4

as a line in the body of the program.

Multiple settings may be set in a single “MODE=” line:

mode = quiet ver4

and multiple mode statements may be specified in a program to change the mode as the program runs

mode = quiet
[some program lines]
mode = verbose
[additional program lines]

Note that setting the execution mode explicitly in a program overrides any settings specified at the time the program is executed.

Program Arguments

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. Any number of arguments may be included in a program; they 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 REGPROG:

```plaintext
equation eq1
smpl 1980q3 1994q1
eq1.ls {%0} c {%1} {%1}(-1) time
```

To run REGPROG from the command line with %0 = “lgdp” and %1 = “m1”, we enter
run regprog lgdp ml

This program performs a regression of the variable LGDP, on C, M1, M1(-1), and TIME, by executing the command

eq1.ls lgdp c m1 m1(-1) time

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. For this example, type “regprog” for the name of the program, and “lgdp” and “m1” for the arguments.

Any arguments in your program that are not initialized in the run command or Run Program dialog are treated as blanks. For example, suppose you have a one-line program named REGRESS:

\[
equation\text{ eq1.ls }y\ c\ time\ \{\%0\}\ \{\%1\}\ \{\%2\}\ \{\%3\}\ \{\%4\}\ \{\%5\}\ \{\%6\}\ \{\%7\}\ \{\%8\}
\]

The command

\[
run\ regress\ x\ x(-1)\ x(-2)
\]
executes

\[
equation\ text{ eq1.ls }y\ c\ time\ x\ x(-1)\ x(-2)
\]

while the command

\[
run\ regress
\]
executes

\[
ls\ y\ c\ time
\]

In both cases, EViews ignores arguments that are not included in your run command.

As a last example, we repeat our simple forecasting program from above, but use arguments to simplify our work. Suppose you have the program, MYPROG:

\[
wfcreate\ \{\%0\}\ m\ 1968m3\ 1997m6
fetch\ progdemo:\{\%1\}
smpl\ 1968m5\ 1992m12
equation\ reg1.ls\ \{\%1\}\ c\ \{\%1\}(-1)
reg1.makeresid\ \{\%1\}res
smpl\ 1993m1\ 1997m6
reg1.forecast\ \{\%1\}fit
\]
The results of running the two example programs at the start of this chapter can be duplicated by running MYPROG with arguments:

```eviews
run myprog myhouse hsf
```

and

```eviews
run myprog mysp500 fspcom
```

### Control of Execution

EViews provides you with several ways to control the execution of commands in your programs. Controlling execution in your program means that you can execute commands selectively or repeat commands under changing conditions. The methods for controlling execution will be familiar from other computer languages.

#### 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.

All comparisons follow the rules outlined in “Numeric Relational Operators” beginning on page 116 and “String Relational Operators” beginning on page 869 of the User’s Guide. Note that beginning in EViews 5, all string comparisons in programs are case-sensitive. You may perform caseless comparison by using the @UPPER or @LOWER string functions.

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. The expression to be tested may also take a numerical value. In this case, 0 and NA are equivalent to FALSE and any other non-zero value is TRUE. For example:

```eviews
if !stand=1 or (!rescale=1 and !redo=1) then
    series gnpstd = gnp/sqr(gvar)
    series constd = cons/sqr(cvar)
endif
if !a>5 and !a<10 then
    smpl 1950q1 1970q1+!a
```
endif
if !scale then
    series newage = age!/scale
endif

Note that in this example, all indentation is done for program clarity and has no effect on the execution of the program lines.

An IF statement may have an ELSE clause containing commands to be executed if the condition is FALSE. If the condition is true, all of the commands up to the keyword else will be executed. If the condition is FALSE, all of the commands between else and endif will be executed. For example,

    if !scale>0 then
        series newage = age!/scale
    else
        series newage = age
    endif

IF statements may also be applied to string variables:

    if %0="CA" or %0="IN" then
        series stateid = 1
    else
        if %0="MA" then
            series stateid=2
        else
            if %0="IN" then
                series stateid=3
            endif
        endif
    endif
endif

Note that the nesting of our comparisons does not create any difficulties.

You should take care when using the IF statement with series or matrices to note that the comparison is defined on the entire object and will evaluate to false unless every element of the element-wise comparison is true. Thus, if X and Y are series, the IF statement

    if x<>y then
        [some program lines]
    endif
evaluates to false if any element of X is not equal to the corresponding value of Y in the default sample. If X and Y are identically sized vectors or matrices, the comparison is over all of the elements X and Y. This behavior is described in greater detail in “Relational Operators (=, >, > =, <, < =, < >)” on page 36.

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 or Scalars

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 don’t include step, the step is assumed to be 1. For example,

\[
\text{for } !j=1 \text{ to } 10 \\
\quad \text{series decile}(!j) = \text{income} < \text{level}(!j) \\
\text{next}
\]

In this example, \( \text{STEP}=1 \) and the variable \( J \) is twice used as a replacement variable, first for the ten series declarations \( \text{DECILE1} \) through \( \text{DECILE10} \) and for the ten variables \( \text{LEVEL1} \) through \( \text{LEVEL10} \).

\[
\text{for } !j=10 \text{ to } 1 \text{ step } -1 \\
\quad \text{series rescale}(!j) = \text{original} / !j \\
\text{next}
\]

In this example, the step is \(-1\), and \( J \) is used as a replacement variable to name the ten constructed series \( \text{RESCALE10} \) through \( \text{RESCALE1} \) and as a scalar in dividing the series \( \text{ORIGINAL} \).

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 \( \text{step} \) and EViews compares the variable to the limit. If the limit is passed, execution stops.

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 observa-
tions forward as the current value of the control variable. Here is a FOR loop that gradually increases the size of the sample and estimates a rolling regression:

\[
\text{for } \text{!horizon}=10 \text{ to } 72 \\
\text{smpl } 1970:1 \text{ } 1970:1+\text{!horizon} \\
\text{equation eq{!horizon}.ls sales c orders} \\
\text{next}
\]

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

\[
\text{!rows} = \text{@rows(vec1)} \\
\text{vector cumsum1 = vec1} \\
\text{for } \text{i}=2 \text{ to } \text{!rows} \\
\text{cumsum1(i) = cumsum1(i-1) + vec1(i)} \\
\text{next}
\]

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 \text{@elem} function and \text{@otod} to get the desired element

\[
\text{for } \text{i}=2 \text{ to } \text{!rows} \\
\text{cumsum1(i) = @elem(ser1, @otod(i))} \\
\text{next}
\]

The \text{@otod} function returns the date associated with the observation index (counting from the beginning of the workfile), and the \text{@elem} function extract 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,

\[
\text{matrix(25,10) xx} \\
\text{for } \text{i}=1 \text{ to } 25 \\
\text{for } \text{j}=1 \text{ to } 10 \\
\text{xx(i,j)=(i-1)*10+j} \\
\text{next} \\
\text{next}
\]

You should avoid changing the control variable within a FOR loop. For example, consider

\[
' \text{potentially confusing loop (avoid doing this)} \\
\text{for } \text{i}=1 \text{ to } 25
\]
vector a!i
!i=!i+10
next

Here, both the FOR assignment and the assignment statement within the loop change the value of the control variable I. Loops of this type are difficult to follow and may produce unintended results. If you find a need to change a control variable inside the loop, consider using a WHILE loop as explained below.

You may execute FOR loops with scalars instead of control variables. However, you must first declare the scalar, and you may not use the scalar as a replacement variable. For example,

```
scalar i
scalar sum = 0
vector (10) x
for i=1 to 10
   x(i) = i
   sum = sum + i
next
```

In this example, the scalars I and SUM remain in the workfile after the program has finished running, unless they are explicitly deleted.

**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 gdp gnp ndp nnp
equation {%y}trend.ls %y c {%y}(-1) time
next
```

executes the commands

```
equation gdptrend.ls gdp c gdp(-1) time
equation gnptrend.ls gnp c gnp(-1) time
equation ndptrend.ls ndp c ndp(-1) time
equation nnptrend.ls nnp c nnp(-1) time
```

You can put multiple string variables in the same FOR statement—EViews will process the strings in sets. For example,
In this case, the elements of the list are taken in groups of three. The loop is executed three times for the different sample pairs and equation names:

```
smpl 1955q1 1960q4 1970q2 1980q3 mid 1975q4
smpl 1995q1 late
smpl %1 %2

equation {%3}eq.ls sales c orders
```

Note the difference between this construction and nested FOR loops. Here, all string variables are advanced at the same time, whereas with nested loops, the inner variable is advanced over all choices, while the outer variable is held constant. For example,

```
!eqno = 1
for %1 1955q1 1960q4
  for %2 1970q2 1980q3 1975q4
    smpl %1 %2
    'form equation name as eq1 through eq6
    equation eq{!eqno}.ls sales c orders
    !eqno=!eqno+1
  next
next
```


**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/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 inc{!val} = income/!val
    !val = !val*10
    !a = !a+1
wend
```

There are four parts to this WHILE loop. The first part is the initialization of the control variables used in the test condition. The second part is the WHILE statement which includes the test. The third part is the statements updating the control variables. Finally the end of the loop is marked by the word 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.

In the example above of a FOR loop that changed the control variable, a WHILE loop provides a much clearer program:

```
!i = 1
while !i<=25
    vector a{!i}
    !i = !i + 11
wend
```
Handling Execution Errors

By default, EViews will stop executing after encountering any errors, but you can instruct the program to continue running even if errors are encountered (see “Executing a Program” on page 84). In the latter case, you may wish to perform different tasks when errors are encountered. For example, you may wish to skip a set of lines which accumulate estimation results when the estimation procedure generated errors.

To test for and handle execution errors, you should use the @errorcount function to return the number of errors encountered while executing your program:

```eviews
!errs = @errorcount
```

The information about the number of errors may be used by standard program statements to control the behavior of the program.

For example, to test whether the estimation of an equation generated an error, you should compare the number of errors before and after the command:

```eviews
!old_count = @errorcount
equation eq1.ls y x c
!new_count = @errorcount
if !new_count > !old_count then
   [various commands]
endif
```

Here, we perform a set of commands only if the estimation of equation EQ1 incremented the error count.

Other Tools

Occasionally, you will wish to stop a program or break out of a loop based on some conditions. To stop a program executing in EViews, use the stop command. For example, suppose you write a program that requires the series SER1 to have nonnegative values. The following commands check whether the series is nonnegative and halt the program if SER1 contains any negative value:

```eviews
series test = (ser1<0)
if @sum(test) <> 0 then
   stop
endif
```

Note that if SER1 contains missing values, the corresponding value of TEST will also be missing. Since the @sum function ignores missing values, the program does not halt for SER1 that has missing values, as long as there is no negative value.
Sometimes, you do not wish to stop the entire program when a condition is satisfied; you just wish to exit the current loop. The `exitloop` command will exit the current `for` or `while` statement and continue running the program.

For example, suppose you computed a sequence of LR test statistics LR11, LR10, LR9, ..., LR1, say to test the lag length of a VAR. The following program sequentially carries out the LR test starting from LR11 and tells you the statistic that is first rejected at the 5% level:

```eviews
!df = 9
for !lag = 11 to 1 step -1
    !pval = 1 - @cchisq(lr{!lag},!df)
    if !pval<=.05 then
        exitloop
    endif
next
scalar lag=!lag
```

Note that the scalar LAG has the value 0 if none of the test statistics are rejected.

### Multiple Program Files

When working with long programs, you may wish to organize your code using multiple files. For example, suppose you have a program file named POWERS.PRG which contains a set of program lines that you wish to use.

While you may be tempted to string files together using the `run` command, we caution you that EViews will stop after executing the commands in the referenced file. Thus, a program containing the lines

```eviews
run powers.prg
series x = nrnd
```

will only execute the commands in the file POWERS, and will stop before generating the series X. This behavior is probably not what you intended.

You should instead use the `include` keyword to include the contents of a program file in another program file. For example, you can place the line

```eviews
include powers
```

at the top of any other program that needs to use the commands in POWERS. `include` also accepts a full path to the program file and you may have more than one `include` statement in a program. For example the lines,

```eviews
include c:\programs\powers.prg
```
will first execute all of the commands in C:\PROGRAMS\POWERS.PRG, then will execute the commands in DURBIN_H.PRG, and then will execute the remaining lines in the program file.

Subroutines provide a more general, alternative method of reusing commands and using arguments.

### 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.

### Defining Subroutines

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:

```eviews
subroutine z_square
  series x = z^2
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 forms the square of the existing series Z and stores it in the new series X.

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. The following program exits the subroutine if Durbin’s $h$ statistic for testing serial correlation with a lagged dependent variable cannot be computed (for details, see Greene, 1997, p.596, or Davidson and MacKinnon, 1993, p. 360):

```eviews
subroutine durbin_h
  equation eqn.ls cs c cs(-1) inc
  scalar test=1-eqn.@regobs*eqn.@cov(2,2)
  ' an error is indicated by test being nonpositive
  ' exit on error
  if test<=0 then
    return
  endif
```

Subroutines

Subroutine with arguments

The subroutines so far have been written to work with a specific set of variables. More generally, subroutines can take arguments. If you are familiar with another programming language, you probably already know how arguments allow you to change the behavior of the group of commands each time the subroutine is used. Even if you haven’t encountered subroutines, you are probably familiar with similar concepts from mathematics. You can define a function, say

\[ f(x) = x^2 \]  \hspace{1cm} (6.1)  

where \( f \) depends upon the argument \( x \). The argument \( x \) is merely a place holder—it’s there to define the function and it does not really stand for anything. Then if you want to evaluate the function at a particular numerical value, say 0.7839, you can write \( f(0.7839) \). If you want to evaluate the function at a different value, say 0.50123, you merely write \( f(0.50123) \). By defining the function, you save yourself from writing out the whole expression every time you wish to evaluate it for a different value.

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 power(series v, series y, scalar p)
    v = y^p
endsub
```

This subroutine generalizes the example subroutine Z_SQUARE. Calling POWER will fill the series given by the argument \( V \) with the power \( P \) of the series specified by the argument \( Y \). So if you set \( V \) equal to \( X \), \( Y \) equal to \( Z \), and \( P \) equal to 2, you will get the equivalent of the subroutine Z_SQUARE above. See the discussion below on how to call subroutines.

Subroutine Placement

Your subroutine definitions should be placed, in any order, at the beginning of your program. The subroutines will not be executed until they are executed by the program using a call statement. For example,

```
subroutine z_square
```
series x=z^2
endsub

' start of program execution
load mywork
fetch z

call z_square

Execution of this program begins with the load statement; the subroutine definition is skipped and is executed only at the last line when it is “called.”

The subroutine definitions must not overlap—after the subroutine keyword, there should be an endsub before the next subroutine declaration. 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 POWERS.PRG then you may put the line

    include powers

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

Calling Subroutines

Once a subroutine is defined in your program, you may execute the commands in the subroutine by using the call keyword. call should be followed by the name of the subroutine, and a list of any argument values you wish to use, enclosed in parentheses and separated by commas. If the subroutine takes arguments, they must all be provided in the same order as in the declaration statement. Here is an example program file that calls subroutines:

    include powers
    load mywork
    fetch z gdp
    series x
    series gdp2
    series gdp3
    call z_square
    call power(gdp2,gdp,2)
    call power(gdp3,gdp,3)
The first calls fills the series X with the value of Z squared. The second call creates the series GDP2 which is GDP squared. The last call creates the series GDP3 as the cube of GDP.

When the subroutine argument is a scalar, the subroutine may be called with a scalar object, a control variable, a simple number (such as “10” or “15.3”), a matrix element (such as “mat1(1,2)”) or a scalar expression (such as “y + 25”). Subroutines that take matrix and vector arguments can be called with a matrix name, and if not modified by the subroutine, may also take a matrix expression. All other arguments must be passed to the subroutine with a simple object (or string) name referring to a single object of the correct type.

Global and Local Variables

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. The program that calls the subroutine will not know anything about a local variable since the local variable will disappear once the subroutine finishes and returns to the original program.

Global Subroutines

By default, subroutines in EViews are global. Any global subroutine may refer to any global object that exists in the workfile at the time the subroutine is called. Thus, if Z is a series in the workfile, the subroutine may refer to and, if desired, alter the series Z. Similarly, if Y is a global matrix that has been created by another subroutine, the current subroutine may use the matrix Y.

The rules for variables in global subroutines are:

- Newly created objects are global and will be included in the workfile when the 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.
- The global objects corresponding to arguments may be used and updated by referring to the arguments.

Here is a simple program that calls a global subroutine:
subroutine z_square
    series x = z^2
endsub
load mywork
fetch z
call z_square

Z_SQUARE is a global subroutine which has access to the global series Z. The new global
series X contains the square of the series Z. Both X and Z remain in the workfile when
Z_SQUARE is finished.

If one of the arguments of the subroutine has the same name as a global variable, the argu-
ment name takes precedence. Any reference to the name in the subroutine will refer to the
argument, not to the global variable. For example,

subroutine sqseries(series z,string %name)
    series {%name} = z^2
endsub
load mywork
fetch z
fetch y
call sqseries(y,"y2")

In this example, there is a series Z in the original workfile and Z is also an argument of the
subroutine. Calling SQSERIES with the argument set to Y tells EViews to use the argument
rather than the global Z. Upon completion of the routine, a new series Y2 will contain the
square of the series Y, not the square of the series Z.

Global subroutines may call global subroutines. You should make certain to pass along any
required arguments when you call a subroutine from within a subroutine. For example,

subroutine wgtols(series y, series wt)
    equation eq1
    call ols(eq1, y)
    equation eq2
    series temp = y/sqr(wt)
    call ols(eq2,temp)
    delete temp
endsub
subroutine ols(equation eq, series y)
    eq.ls y c y(-1) y(-1)^2 y(-1)^3
endsub

can be run by the program:

    load mywork
    fetch cpi
    fetch cs
    call wgtols(cs,cpi)

In this example, the subroutine WGTOLS explicitly passes along the arguments for EQ and Y to the subroutine OLS; otherwise those arguments would not be recognized by OLS. If EQ and Y were not passed, OLS would try to find a global series named Y and a global equation named EQ, instead of using EQ1 and CS or EQ2 and TEMP.

You cannot use a subroutine to change the object type of a global variable. Suppose that we wish to declare new matrices X and Y by using a subroutine NEWXY. In this example, the declaration of the matrix Y works, but the declaration of matrix X generates an error because a series named X already exists:

    subroutine newxy
        matrix(2,2) x = 0
        matrix(2,2) y = 0
    endsub
    load mywork
    series x
    call newxy

EViews will return an error indicating that the global series X already exists and is of a different type than a matrix.

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.

The rules for variables in local subroutines are:

- 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 vanish when the subroutine finishes.

If you want to save results from a local subroutine, you have to explicitly include them in the arguments. For example,

```eviews
subroutine local ols(series y, series res, scalar ssr)
    equation temp_eq.ls y c y(-1) y(-1)^2 y(-1)^3
    temp_eq.makeresid res
    ssr = temp_eq.@ssr
endsub
```

This local subroutine takes the series Y as input and returns the series RES and scalar SSR as output. The equation object TEMP_EQ is local to the subroutine and will vanish when the subroutine finishes.

Here is an example program that calls this local subroutine:

```eviews
load mywork
fetch hsf
equation eq1.ls hsf c hsf(-1)
eq1.makeresid rres
scalar rssr = eq1.@ssr
series ures
scalar ussr
call ols(hsf, ures, ussr)
```

Note how we first declare the series URES and scalar USSR before calling the local subroutine. These objects are global since they are declared outside the local subroutine. When we call the local subroutine by passing these global objects as arguments, the subroutine will update these global variables.

There is one exception to the general inaccessibility of global variables in local subroutines. When a global group is passed as an argument to a local subroutine, any series in the group is accessible to the local routine.

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. For example, the subroutine

```eviews
subroutine newols(series y, series res)
    include ols
    equation eq1.ls y c y(-1)
```
and the program
load mywork
fetch hsf
series rres
call newols(hsf, rres)

produce equivalent results. Note that the subroutine NEWOLS still does not have access to any of the temporary variables in the local routine OLS, even though OLS is called from within NEWOLS.

Programming Summary

Support Commands
- open opens a program file from disk (p. 342).
- output redirects print output to objects or files (p. 347).
- poff turns off automatic printing in programs (p. 565).
- pon turns on automatic printing in programs (p. 565).
- program declares a program.
- run runs a program (p. 391).
- statusline sends message to the status line (p. 566).
- tic reset the timer (p. 469).
- toc display elapsed time (since timer reset) in seconds (p. 470).

Program Statements
- call calls a subroutine within a program (p. 559).
- else denotes start of alternative clause for IF (p. 560).
- endif marks end of conditional commands (p. 560).
- endsub marks end of subroutine definition (p. 560).
- exitloop exits from current loop (p. 561).
- for start of FOR execution loop (p. 562).
- if conditional execution statement (p. 562).
include ................ include subroutine in programs (p. 563).
next ........................ end of FOR loop (p. 563).
return ...................... exit subroutine (p. 566).
step ........................ (optional) step size of a FOR loop (p. 567).
stop ........................ halts execution of program (p. 568).
subroutine .............. declares subroutine (p. 568).
then ........................ part of IF statement (p. 569).
to ........................... upper limit of FOR loop (p. 570).
wend ........................ end of WHILE loop (p. 571).
while ........................ start of WHILE loop (p. 571).

Support Functions

@date ........................ string containing the current date (p. 559).
@errorcount ................. number of errors encountered (p. 561).
@evpath ........................ string containing the directory path for the EViews executable (p. 561).
@isobject ..................... checks for existence of object (p. 563).
@obsrage ........................ returns number of observations in the current active workfile range (0 if no workfile in memory).
@obssmpl ........................ returns number of observations in the current active workfile sample (0 if no workfile in memory).
@temppath ........................ string containing the directory path for EViews temporary files (p. 568).
@time ........................ string containing the current time (p. 569).
@toc ........................ calculates elapsed time (since timer reset) in seconds (p. 570).
Chapter 7. Sample Programs

In this chapter, we provide extended examples showing the use of the EViews programming language to perform the following tasks:

- Computing descriptive statistics by year.
- Rolling window unit-root (ADF) tests.
- Calculating cumulative sums.
- Performing time series operations on a sample of observations.
- Creating dummy variables with a loop.
- Extracting test statistics in a loop.
- Between group estimation for pooled data.
- Hausman test for fixed versus random effects.
- Reformatted regression output table.

In addition, there are a number of example files for creating and estimating user defined likelihood functions. These example files are as described in “Additional Examples” on page 661.

We have included program and data files for each example in your EViews examples directory. You may open each file by clicking on File/Open/Program… and selecting the appropriate file. To run the program, either click on Run in the program window, or enter the command run, followed by the program name, in the command window.

Note that there may be additional programs in your EViews directory that were added after the manual went to press and therefore are not listed here. If you find that the examples here do not cover an issue of interest, you may wish to browse the contents of your EViews examples directory.

Descriptive Statistics by Year

(descr1.prg)

Suppose that you wish to compute descriptive statistics (mean, median, etc.) for each year of your monthly data in the series IP, URATE, M1, and TB10. While EViews does not provide a built-in procedure to perform this calculation, you can use the statsby view of a series to compute the relevant statistics in two steps: first create a year identifier series, and second compute the statistics for each value of the identifier.

' load the basics workfile
%evworkfile = @evpath + "\example files\basics"
load "{%evworkfile}"  
smpl 1990:1 @last

' create a series containing the year identifier
series year = @year

' compute statistics for each year and
' freeze the output from each of the tables
for %var ip urate m1 tb10
  %name = "tab" + %var
  freeze(%name){%var}.statby(min,max,mean,med) year
next

The results are saved in tables named TABIP, TABURATE, TABM1, and TABTB10. For example, TABIP contains:

<table>
<thead>
<tr>
<th>YEAR</th>
<th>Mean</th>
<th>Median</th>
<th>Max</th>
<th>Min</th>
<th>Std. Dev.</th>
<th>Obs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1990</td>
<td>106.0696</td>
<td>106.3725</td>
<td>106.8150</td>
<td>104.5380</td>
<td>0.736456</td>
<td>12</td>
</tr>
<tr>
<td>1991</td>
<td>104.2815</td>
<td>104.5610</td>
<td>105.7480</td>
<td>102.0980</td>
<td>1.230377</td>
<td>12</td>
</tr>
<tr>
<td>1992</td>
<td>107.6821</td>
<td>107.7565</td>
<td>110.4470</td>
<td>104.8810</td>
<td>1.605744</td>
<td>12</td>
</tr>
<tr>
<td>1993</td>
<td>112.0888</td>
<td>111.7070</td>
<td>114.7110</td>
<td>110.6500</td>
<td>1.168270</td>
<td>12</td>
</tr>
<tr>
<td>1994</td>
<td>118.0779</td>
<td>118.1270</td>
<td>121.0000</td>
<td>114.7440</td>
<td>1.993301</td>
<td>12</td>
</tr>
<tr>
<td>1995</td>
<td>121.6750</td>
<td>121.8000</td>
<td>122.0000</td>
<td>121.1000</td>
<td>0.427201</td>
<td>4</td>
</tr>
<tr>
<td>All</td>
<td>110.3922</td>
<td>108.1350</td>
<td>122.0000</td>
<td>102.0980</td>
<td>5.805422</td>
<td>64</td>
</tr>
</tbody>
</table>

(desc2.prg)

The example above saves the values in a table. You may instead wish to place the annual values in a vector or matrix so that they can be used in other calculations. The following program creates a vector to hold the median values for each year, and loops through the calculation of the median for each year:

' load the basics workfile
%evworkfile = @evpath + "\example files\basics"
load "{%evworkfile}"  
smpl 1990:1 @last

' create series with the year and find the maximum
series year = @year
!lastyear = @max(year)

' loop over the series names
for $var ip urate m1 tbl0

' create matrix to hold the results
!numrows = (!lastyear - 1990 + 1) + 1
matrix(!numrows, 7) mat%var

' loop over each year and compute values
for !i = 1990 to !lastyear
  !row = !i - 1990 + 1
  smpl if (year = !i)
  mat%var(!row,1) = !i
  mat%var(!row,2) = @mean({$var})
  mat%var(!row,3) = @med({$var})
  mat%var(!row,4) = @min({$var})
  mat%var(!row,5) = @max({$var})
  mat%var(!row,6) = @std({$var})
  mat%var(!row,7) = @obs({$var})
next

' compute the total values
mat%var(!numrows,1) = !i
mat%var(!numrows,2) = @mean({$var})
mat%var(!numrows,3) = @med({$var})
mat%var(!numrows,4) = @min({$var})
mat%var(!numrows,5) = @max({$var})
mat%var(!numrows,6) = @std({$var})
mat%var(!numrows,7) = @obs({$var})
next

Rolling ADF Test

(rollreg.prg)

The following program runs a set of rolling ADF tests on the series M1 using a moving sample of fixed size. The basic techniques for working with rolling samples may be used in a variety of settings. The program stores and displays the \( t \)-statistics together with the asymptotic critical value (C2):
' load the data
%evworkfile = @evpath + "\example files\basics"
load "{%evworkfile}"
smpl @all

' find size of workfile
series _temp = 1
!length = @obs(_temp)
delete _temp
' set fixed sample size
! ssize = 50

' initialize matrix to store results
matrix(!length-!ssize+1,2) adftstat
' run test regression for each subsample and store
' each ar(1) coefficient test includes a constant
' with no lagged first difference
for !i = 1 to !length-!ssize+1
    smpl @first+!i-1 @first+!i+!ssize-2
    equation temp.ls d(m1) c m1(-1)
adftstat(!i,1) = temp.@tstat(2)
' 5% critical value from Davidson and MacKinnon,
' Table 20.1, page 708
adftstat(!i,2) = -2.86
next

freeze(graph1) adftstat.line
graph1.name(1) ADF t-statistic
graph1.name(2) asymptotic 5% critical value
graph1.addtext(t) ADF t-statistic for window of 50 obs for M1
show graph1
Calculating Cumulative Sums

(cum_sum.prg)

EViews does not have a built-in function for calculating cumulative sums. You can easily calculate such sums with a few lines of commands.

Say we have a series X for which we would like to calculate the cumulative sum. If the series does not contain NAs, you can use:

```eviews
smpl @first @first
series sum0 = x
smpl @first+1 @last
sum0 = sum0(-1) + x
```

If the series does contain NAs, you have two options, depending on how you would like the NAs to be treated. If you would like to continue accumulating the sum, ignoring the NA observations, you can use:

```eviews
smpl @first @first
series sum1 = @nan(x, 0)
smpl @first+1 @last
sum1 = sum1(-1) + @nan(x, 0)
smpl @all if x = na
sum1 = na
```

If you would like to reset the sum to zero whenever an NA appears in the series you can use:

```eviews
smpl @all
series sum2 = x
smpl if sum2(-1)<na and x<>na
```
The following sample output shows the differences between the three methods:

```
sum2 = sum2(-1) + x

Time Series Operations on a Sample
(subset.prg)

Many operations in EViews can be performed on a subset of the workfile observations using the `smpl` command. To estimate an equation using only observations which have a positive value of X, you could simply use:

```
smpl if x>=0
eq1.ls y c x
```

Time series operations often rely on observations being adjacent, so that this method may not work as expected. In this example we demonstrate a general technique for working with a subset of observations from the workfile as though they were adjacent. There are three steps to using the method:

- create a “short version” of the data which contains only the subset of observations.
- use the “short version” of the data to get whatever results you need.
- create a “long version” of the results which match the observations in the original data.

Consider the simple case of creating a series which contains the differences between consecutive positive values of the series X, ignoring any intervening negative values.

The commands

```
smpl if x>=0
series dxp = d(x)
```
will not compute the desired values, since the $d$ function computes differences between adjacent values, not between successive values in the sample.

The SUBSET.PRG program solves this problem using the general technique outlined above. The solution is stored in the series DX. The series $X_S$ and $DX_S$ are left in the workfile so you can follow the intermediate calculations.

```
' create sample
sample ss if x<>NA and x>=0

' create short x series
smpl @all
vector temp
stomna(x, temp, ss)
series x_s
mtos(temp, x_s)
!rows = @rows(temp)  'save number of elements
delete temp

' difference short x series
series dx_s = d(x_s)

' map back into long series dx
vector temp
stomna(dx_s, temp)
vector(!rows) temp  'trim to number of elements
series dx
mtos(temp, dx, ss)
delete temp

' display results
show x dx x_s dx_s
```

Creating Dummy Variables with a Loop

(make_dum.prg)

It is sometimes useful to create a set of dummy variables for a range of observations in a workfile. You can then include one or more of these dummies in an estimation simply by adding their names to the list of exogenous variables. It is easy to create these series by hand, though somewhat tedious.
The program MAKE_DUM.PRG automates this task. It should be run from the command line with arguments for the first and the last observation for which to create dummies. For example, to create dummies in a quarterly workfile from the first quarter of 1982 to the last quarter of 1983, the command would be:

```
run make_dum 1982:1 1983:4
```

As is, the program should work for annual, semiannual, quarterly, monthly and undated workfiles. It should be modified for weekly and daily workfiles.

```eviews
'set up start and end dates
%start = %0
%end = %1

'generate dummy variables from 'start' to 'end'
for !i = @dtoo(%start) to @dtoo(%end)
    %obsstr = @otod(!i)
    if (@mid(%obsstr, 5, 1) = ":") then
        %name = "d_" + @left(%obsstr, 4) + ":" + @mid(%obsstr, 6)
    else
        %name = "d_" + %obsstr
    endif
    smpl @all
    series %name = 0
    smpl %obsstr %obsstr
    series %name = 1
next
```

**Extracting Test Statistics in a Loop**

(omitted.prg)

Most of the output from equation or system estimation can be accessed directly using EViews regression statistics functions (see Chapter, “”, beginning on page 135). This is not true of all of the output from EViews’ diagnostic tests.

If you would like to work with statistics from these views in a program, you should use the following general method that allows you to extract and store output from any object view. The basic steps in each step through the loop are:

- **freeze** the view into a temporary table.
- **extract** the cell containing the statistic and assign its contents to an element of a table or vector.
- **delete** the temporary table.
The following subroutine uses this approach to capture the $F$-statistic and probability value from an omitted variables test. The subroutine OMIT_TEST cycles through the series contained in a group, testing whether the first four lags of each series are jointly significant when added to the chosen equation. The output is stored in a table.

```plaintext
' test whether the first four lags of an omitted variable
' are jointly significant (eq1: the equation to test;
' g: a list of the series to include; results: output)
subroutine omit_test(equation eq1, group g, table results)
!n = g.@count
table(3, !n) results
setcolwidth(results, 1, 12)
setcell(results, 1, 1, "Four lag F-test for omitted variables:", "l")
results(2,2) = "F-stat"
results(2,3) = "Probability"
for !i = 1 to !n
  %name = g.@seriesname(!i)
  series temp_s = %name
  freeze(temp_t) eq1.testadd temp_s(-1) temp_s(-2) temp_s(-3) temp_s(-4)
  results(!i+2, 1) = %name
  results(!i+2, 2) = temp_t(3, 2)
  results(!i+2, 3) = temp_t(3, 5)
  delete temp_t
  delete temp_s
next
endsub
```

The following program uses this subroutine to carry out some tests on the data from the demo in Chapter 2, “A Demonstration”, on page 21 of the User’s Guide. Both the subroutine and the program can be found in the file OMITTED.PRG.

```plaintext
open demo.wf1
smpl @all
equation eq1.ls dlog(m1) c dlog(m1(-1)) dlog(m1(-2)) dlog(m1(-3)) dlog(m1(-4))
group g1 dlog(gdp) dlog(rs) dlog(pr)
table tab1
call omit_test(eq1, g1, tab1)
show tab1
```

The program produces the following output:
Between Group Estimation for Pooled Data
(between.prg)

Consider a pooled regression specification:

$$\log(y_{it}) = \alpha_i + \beta \log(x_{it}) + \varepsilon_{it},$$  \hspace{1cm} (7.1)

where \(i\) is the cross-section index and \(t\) represents the time period. EViews provides built-in procedures to perform fixed effects (within-groups) estimation of \(\beta\):

$$\log(y_{it}) - \bar{\log}(y_{it}) = \beta \bar{\log}(x_{it}) - \log(x_{it}) + (\varepsilon_{it} - \bar{\varepsilon}_{it}),$$  \hspace{1cm} (7.2)

where a bar over a variable name represents the average over time for the cross-section.

Alternatively, you may wish to estimate the between groups model:

$$\log(y_{it}) = \alpha_i + \beta_B \log(x_{it}) + \bar{\varepsilon}_{it}$$  \hspace{1cm} (7.3)

A quick way to obtain the between groups estimate is to use auto series in a pooled regression. If we run the pool regression

```
pool1.ls(c) @mean(log(ivm?)) @mean(log(mm?))
```

in the workfile POOL.WF1, we obtain:

```
Dependent Variable: @MEAN(LOG(IVM?))
Method: Pooled Least Squares
Date: 08/24/00   Time: 10:56
Sample: 1968:01 1995:12
Included observations: 336

Number of cross-sections used: 8

Total panel (balanced) observations: 2688

<table>
<thead>
<tr>
<th>Variable</th>
<th>Coefficient</th>
<th>Std. Error</th>
<th>t-Statistic</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0.855345</td>
<td>0.092523</td>
<td>9.244701</td>
<td>0.0000</td>
</tr>
<tr>
<td>@MEAN(LOG(MM?))</td>
<td>0.959626</td>
<td>0.009793</td>
<td>97.99203</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

R-squared          0.781421  Mean dependent var 9.890818
Adjusted R-squared 0.781339  S.D. dependent var 0.847896
S.E. of regression 0.396486  Sum squared resid 422.2431
Log likelihood     -1326.401 F-statistic 9602.439
Prob(F-statistic)  0.000000  
```

Although the point estimates are correct, the standard errors are not since EViews uses multiple observations for each cross-section unit.
We can, however, use a program to obtain the between estimator. The following program stores the cross-section specific means of each series in a matrix, creates a new workfile, converts the matrix to series, and runs the between groups regression:

```plaintext
' set number of cross-sections
%evworkfile = @evpath + "\example files\pool1"
load "{%evworkfile}"
smpl @all
!ncross = pool1.@ncross

' create group with variables
pool1.makegroup(tempgrp) log(ivm?) log(mm?)

' store means of two series in matrix
matrix(!ncross,2) means
series tempser
for !i = 1 to !ncross
   tempser = tempgrp(!i)
   means(!i,1) = @mean(tempser)
   tempser = tempgrp(!ncross+!i)
   means(!i,2) = @mean(tempser)
next
store(i) means

' create new workfile and fetch means matrix
workfile between u 1 !ncross
fetch(i) means

' convert matrix to series
series lc_mean
series ly_mean
group g1 lc_mean ly_mean
mtos(means,g1)

' run between groups regression and cleanup
equation eq_bet.ls lc_mean c ly_mean
delete tempgrp tempser
show eq_bet
```

The equation results for the between estimator are given by:
Chapter 7. Sample Programs

128

Note that the temporary group TEMPGRP and temporary series TEMPSER are deleted at the end of the program.

Hausman Test for Fixed Versus Random Effects

(hausman.prg)

The following program computes the Hausman test statistic for testing the null hypothesis of random effects against the alternative of fixed effects. The program estimates a fixed and random effects model with two slope regressors and stores the estimated coefficients and its covariance matrix. (The Grunfeld data used in the program is transcribed from Greene (1997), Table 15.1.).

' load the data
%evworkfile = @evpath + "\example files\grunfeld"
load "({%evworkfile})"
smpl @all

' estimate fixed effects and store results
pool1.ls(f) log(inv?) log(val?) log(cap?)
vector beta = pool1.@coefs
matrix covar = pool1.@cov

' keep only slope coefficients
vector b_fixed = @subextract(beta,1,1,2,1)
matrix cov_fixed = @subextract(covar,1,1,2,2)

' estimate random effects and store results
pool1.ls(r) log(inv?) log(val?) log(cap?)
beta = pool1.@coefs

Dependent Variable: LC_MEAN
Method: Least Squares
Date: 08/24/00   Time: 10:55
Sample: 1 8
Included observations: 8

<table>
<thead>
<tr>
<th>Variable</th>
<th>Coefficient</th>
<th>Std. Error</th>
<th>t-Statistic</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0.855345</td>
<td>1.957609</td>
<td>0.436934</td>
<td>0.6774</td>
</tr>
<tr>
<td>LY_MEAN</td>
<td>0.959626</td>
<td>0.207199</td>
<td>4.631412</td>
<td>0.0036</td>
</tr>
</tbody>
</table>

R-squared 0.781421  Mean dependent var 9.890818
Adjusted R-squared 0.744991  S.D. dependent var 0.906271
S.E. of regression 0.457653  Akaike info criterion 1.486906
Sum squared resid 1.256676  Schwarz criterion 1.508766
Log likelihood -3.947623  F-statistic 21.44997
Durbin-Watson stat 1.352621  Prob(F-statistic) 0.003572
covar = pool1.@cov

' keep only slope coefficients
vector b_gls = @subextract(beta,2,1,3,1)
matrix cov_gls = @subextract(covar,2,2,3,3)

' compute Hausman test stat
matrix b_diff = b_fixed - b_gls
matrix v_diff = cov_fixed - cov_gls
matrix qform = @transpose(b_diff)*@inverse(v_diff)*b_diff

if qform(1,1)>=0 then
  ' set table to store results
  table(4,2) result
  setcolwidth(result, 1, 17)
  setcell(result,1,1,"Hausman test","l")
  setcell(result,2,1,"(fixed versus random effects)","l")
  setline(result,3)
  !df = @rows(b_diff)
  setcell(result,4,1,"Chi-square ("+@str(!df)+" d.f.)","l")
  setcell(result,4,2,qform(1,1))
  setcell(result,5,1,"p-value","l")
  setcell(result,5,2,1-@cchisq(qform(1,1),!df))
  show result
else
  statusline "Quadratic form is negative"
endif

Note that the program stops if the computed test statistic is negative, which can happen in finite samples. Running this program yields the table

<table>
<thead>
<tr>
<th>Hausman test</th>
</tr>
</thead>
<tbody>
<tr>
<td>(fixed versus random effects)</td>
</tr>
<tr>
<td>Chi-square (2 d.f.)</td>
</tr>
<tr>
<td>p-value</td>
</tr>
</tbody>
</table>
Regression Output Table

(regrun.prg / regtab.prg)

EViews presents regression output in tabular form, with separate columns for the coefficient values and the standard errors. For example, the output view from the equation EQ1 in the workfile BASICS.WF1 is given by:

A commonly used alternative display format places the standard errors or $t$-statistics, enclosed in parentheses, below the coefficient estimates. The program file REGTAB.PRG contains a subroutine which takes an equation object and creates a table with this alternative output:

```
' subroutine to reformat regression output
subroutine regtab(equation eq1, table tab1, scalar format)

' assign useful values
!ncoef = eq1.@ncoef
!obs = eq1.@regobs

' create a temporary table with the results of the estimation
' and declare a new table
table(1,4) temp_table
freeze(temp_table) eq1.results

setcolwidth(temp_table, 1, 19)
setcolwidth(temp_table, 2, format+4)
setcolwidth(temp_table, 3, 19)
setcolwidth(temp_table, 4, format+4)
```
' get the original header information
!line = 1
while temp_table(!line, 2) <> "Coefficient"
    setcell(tab1, !line, 1, temp_table(!line, 1), "l")
    !line = !line +1
wend
setline(tab1, !line-1)
setcell(tab1, !line, 1, "Variable", "c")
setcell(tab1, !line, 2, "Estimate", "r")
setcell(tab1, !line, 3, "Estimate", "r")
setcell(tab1, !line+1, 2, "(s.e.)", "r")
setcell(tab1, !line+1, 3, "(t-stat)", "r")
!line = !line + 2
setline(tab1, !line)
!line = !line + 1
!vline = !line

' fill all of the coefficients and standard errors
' (or t-statistics)
for !i = 1 to !ncoef
    %variable = temp_table(!vline-1, 1)
    ' write coefficients
    setcell(tab1, !line, 1, %variable, "c")
    !vline = !vline + 1
    ' write coefficients
    !est=eql.@coefs(!i)
    setcell(tab1, !line, 2, !est, "r", format)
    setcell(tab1, !line, 3, !est, "r", format)
    !line = !line + 1
    ' compute statistics
    !se = sqr(eql.@covariance(!i, !i))
    !tstat = !est/!se
    !tprob = @tdist(!tstat, !obs-!ncoef)
    ' write standard error output
    setcell(tab1, !line, 2, !se, "r", format)
    %str_se = tab1(!line, 2)
    %str_se = "(" + %str_se + ")"
    tab1(!line, 2) = %str_se
' write t-statistic output
setcell(tab1, !line, 3, !tstat, "r", format)
%str_t = "(" + tab1(!line, 3) + ")"
if !tprob < .01 then
  %str_t = "**" + %str_t
else
  if !tprob < .05 then
    %str_t = "*" + %str_t
  endif
endif
!line = !line + 1
next
setline(tab1, !line)
!line = !line + 1

' original results at bottom of table
setcell(tab1, !line, 1, "R-squared", "r")
setcell(tab1, !line, 2, eq1.@r2, "r", format)
setcell(tab1, !line, 3, "Mean dependent var", "l")
setcell(tab1, !line, 4, eq1.@meandep, "r", format)
!line = !line + 1
setcell(tab1, !line, 1, "Adjusted R-squared", "l")
setcell(tab1, !line, 2, eq1.@rbar2, "r", format)
setcell(tab1, !line, 3, "S.D. dependent var", "l")
setcell(tab1, !line, 4, eq1.@sddep, "r", format)
!line = !line + 1
setcell(tab1, !line, 1, "S.E. of regression", "l")
setcell(tab1, !line, 2, eq1.@se, "r", format)
setcell(tab1, !line, 3, "Akaike info criterion", "l")
setcell(tab1, !line, 4, eq1.@aic, "r", format)
!line = !line + 1
setcell(tab1, !line, 1, "Sum squared resid", "l")
setcell(tab1, !line, 2, eq1.@ssr, "r", format)
setcell(tab1, !line, 3, "Schwarz criterion", "l")
setcell(tab1, !line, 4, eq1.@schwarz, "r", format)
!line = !line + 1

setcell(tab1, !line, 1, "Log likelihood", "l")
setcell(tab1, !line, 2, eq1.@logl, "r", format)
setcell(tab1, !line, 3, " F-statistic", "l")
setcell(tab1, !line, 4, eq1.@f, "r", format)
!line = !line + 1

setcell(tab1, !line, 1, "Durbin-Watson stat", "l")
setcell(tab1, !line, 2, eq1.@dw, "r", format)
setcell(tab1, !line, 3, " Prob(F-statistic)", "l")
setcell(tab1, !line, 4, @fdist(eq1.@f,(!ncoef-1),(!obs-!ncoef)),
               "r", format)
!line = !line + 1
setline(tab1, !line)
endsub

To call the subroutine, you can open the program REGRUN.PRG, which contains the following commands:

include regtab.prg
%evworkfile = @evpath + "\example files\basics"
load "{%evworkfile}"
table tab1
call regtab(eq1, tab1, 3)
show tab1

The first argument to the REGTAB subroutine is the equation object, the second is the table into which you wish to place the results, and the third argument is a numeric format code. Here we indicate that we wish to display three digits after the decimal. Table TAB1 will contain the reformatted output:
Dependent Variable: LOG(M1)
Method: Least Squares
Date: 08/18/97   Time: 14:02
Sample: 1959:01 1989:12
Included observations: 372

<table>
<thead>
<tr>
<th>Variable</th>
<th>Estimate (s.e.)</th>
<th>Estimate (t-stat)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>-1.700 (0.165)</td>
<td>**(-10.305)</td>
</tr>
<tr>
<td>LOG(IP)</td>
<td>1.766 (0.044)</td>
<td>**(40.552)</td>
</tr>
<tr>
<td>TB3</td>
<td>-0.012 (0.005)</td>
<td>*(-2.570)</td>
</tr>
</tbody>
</table>

R-squared 0.886
Adjusted R-squared 0.886
S.E. of regression 0.187
Sum squared resid 12.929
Log likelihood 97.010
Durbin-Watson stat 0.009

Mean dependent var 5.664
S.D. dependent var 0.554
Akaike info criterion -0.505
Schwarz criterion -0.474
F-statistic 1439.848
Prob(F-statistic) 0.000
Appendix A. Object, View and Procedure Reference

The following is a reference guide to the views, procedures, and data members for each of the objects found in EViews:

- Alpha (p. 137)
- Coef (p. 137)
- Equation (p. 139)
- Graph (p. 143)
- Group (p. 145)
- Logl (p. 148)
- Matrix (p. 150)
- Model (p. 152)
- Pool (p. 153)
- Rowvector (p. 156)
- Sample (p. 157)
- Scalar (p. 158)
- Series (p. 159)
- Sspace (p. 161)
- System (p. 166)
- Table (p. 169)
- Text (p. 169)
- Table (p. 169)
- Var (p. 171)
- Vector (p. 174)
- Valmap (p. 171)

To use these views, procedures, and data members you should list the name of the object followed by a period, and then enter the name of the view, procedure, or data member, along with any options or arguments:

- object_name.view_name(options) arguments
- object_name.proc_name(options) arguments
- object_name.data_member

For example, to display the line graph view of the series object CONS, you can enter the command:

cons.line

To perform a dynamic forecast using the estimates in the equation object EQ1, you may enter:

eq1.forecast y_f

To save the coefficient covariance matrix from EQ1, you can enter:

sym cov1=eq1.@coefcov

Each of the views and procedures is documented more fully in the alphabetical listing found in Appendix B, “Command Reference”, on page 177.
Alpha

Alpha (alphanumeric) series. An EViews alpha series contains a set of observations on a variable containing string values

Alpha Declaration

```
alp hab ................... declare alpha series (p. 185).
frml ..................... create alpha series object with a formula for auto-updating (p. 278).
genr ..................... create alpha or numeric series object (p. 281).
```

To declare an alpha series, use the keyword `alpha`, followed by a name, and optionally, by an “=” sign and a valid series expression:

```
alpha y
alpha x = "initial strings"
```

If there is no assignment, the series will be initialized to contain blank values, “”:

Alpha Views

```
freq ..................... one-way tabulation (p. 276).
label .................... label information for the alpha (p. 302).
sheet ..................... spreadsheet view of the alpha (p. 432).
```

Alpha Procs

```
displayname ............ set display name (p. 251).
map ..................... assign or remove value map setting (p. 334).
setformat ............. set the display format for the alpha series spreadsheet (p. 419).
setindent ............. set the indentation for the alpha series spreadsheet (p. 424).
setjust ................. set the justification for the alpha series spreadsheet (p. 425).
setwidth ............... set the column width for alpha series spreadsheet (p. 431).
```

Alpha Data Members

```
i ...................... i-th element of the alpha series from the beginning of the workfile (when used on the left-hand side of an assignment, or when the element appears in a table or string variable assignment).
```

Alpha Element Functions

```
@elem(ser, j) ....... function to access the j-th observation of the alpha series, where j identifies the date or observation.
```

Alpha Examples

```
alpha val = "initial string"
```

initializes an alpha series VAL using a string literal.

If FIRST is an alpha series containing first names, and LAST is an alpha containing last names, then

\[ \text{alpha name} = \text{first} + " \" + \text{last} \]

creates an alpha series containing the full names.

**Coef**

Coefficient vector. Coefficients are used to represent the parameters of equations and systems.

**Coef Declaration**

\[
\text{coef} \ldots \text{declare coefficient vector (p. 223).}
\]

There are two ways to create a coef. First, enter the `coef` keyword, followed by a name to be given to the coefficient vector. The dimension of the coef may be provided in parentheses after the keyword:

\[
\text{coef alpha} \\
\text{coef(10) beta}
\]

If no dimension is provided, the resulting coef will contain a single element.

You may also combine a declaration with an assignment statement. If you do not provide an explicit assignment statement, new coefs are initialized to zero.

See also `param` (p. 368) for information on initializing coefficients, and the entries for each of the estimation objects (equation, pool, sspace, system, and var) for additional methods of accessing coefficients.

**Coef Views**

\[
\text{area} \ldots \text{area graph of the coefficient vector (p. 191).} \\
\text{bar} \ldots \text{bar graph of coefficient vector plotted against the coefficient index (p. 199).} \\
\text{label} \ldots \text{label view (p. 302).} \\
\text{line} \ldots \text{line graph of coefficient vector plotted against the coefficient index (p. 306).} \\
\text{sheet} \ldots \text{spreadsheet view of the coefficient (p. 432).} \\
\text{spike} \ldots \text{spike graph (p. 442).} \\
\text{stats} \ldots \text{descriptive statistics (p. 449).}
\]
Coef Procs

- **displayname** ........ set display name (p. 251).
- **fill** ................ fill the elements of the coefficient vector (p. 267).
- **read** ................. import data into coefficient vector (p. 377).
- **setformat** ............ set the display format for the coefficient vector spreadsheet (p. 419).
- **setindent** ............ set the indentation for the coefficient vector spreadsheet (p. 424).
- **setjust** .............. set the justification for the coefficient vector spreadsheet (p. 425).
- **setwidth** ............. set the column width for the coefficient vector spreadsheet (p. 431).
- **write** ................ export data from coefficient vector (p. 498).

Coef Data Members

- **(i)** .................. i-th element of the coefficient vector. Simply append “(i)” to the matrix name (without a “.”).

Coef Examples

The coefficient vector declaration

```
coef(10) coef1=3
```

creates a 10 element coefficient vector COEF1, and initializes all values to 3.

Suppose MAT1 is a $10 \times 1$ matrix, and VEC1 is a 20 element vector. Then

```
coef mycoef1=coef1
coef mycoef2=mat1
coef mycoef3=vec1
```

create, size, and initialize the coefficient vectors MYCOEF1, MYCOEF2 and MYCOEF3.

Coefficient elements may be referred to by an explicit index. For example:

```
vector(10) mm=beta(10)
scalar shape=beta(7)
```

fills the vector MM with the value of the tenth element of BETA, and assigns the seventh value of BETA to the scalar SHAPE.
Equation

Equation object. Equations are used for single equation estimation, testing, and forecasting.

Equation Declaration

equation .................. declare equation object (p. 260).

To declare an equation object, enter the keyword equation, followed by a name:

equation eq01

and an optional specification:

equation r4cst.ls r c r(-1) div
equation wcd.ls q=c(1)*n^c(2)*k^c(3)

Equation Methods

arch ............................ autoregressive conditional heteroskedasticity (ARCH and GARCH) (p. 187).
binary .......................... binary dependent variable models (includes probit, logit, gompit) models (p. 202).
censored ........................ censored and truncated regression (includes tobit) models (p. 217).
count .......................... count data modeling (includes poisson, negative binomial and quasi-maximum likelihood count models) (p. 233).
gmm ............................ generalized method of moments (p. 282).
logit ........................... logit (binary) estimation (p. 314).
ls ............................... linear and nonlinear least squares regression (includes weighted least squares and ARMAX) models (p. 315).
ordered ........................ ordinal dependent variable models (includes ordered probit, ordered logit, and ordered extreme value models) (p. 345).
probit ........................... probit (binary) estimation (p. 374).
ts1s ............................. linear and nonlinear two-stage least squares (TSLS) regression models (includes weighted TSLS, and TSLS with ARMA errors) (p. 474).

Equation Views

archtest ...................... LM test for the presence of ARCH in the residuals (p. 190).
arma .......................... Examine ARMA structure of estimated equation (p. 194).
auto .......................... Breusch-Godfrey serial correlation Lagrange Multiplier (LM) test (p. 196).
cellipse .......... Confidence ellipses for coefficient restrictions (p. 215).
chow ............... Chow breakpoint and forecast tests for structural change (p. 220).
coefcov ................ coefficient covariance matrix (p. 224).
correl ............... correlogram of the residuals (p. 231).
correlsq ............. correlogram of the squared residuals (p. 232).
derivs .............. derivatives of the equation specification (p. 249).
garch ................ conditional standard deviation graph (only for equations estimated using ARCH) (p. 280).
grads ............... examine the gradients of the objective function (p. 287).
hist .................. histogram and descriptive statistics of the residuals (p. 294).
label ............... label information for the equation (p. 302).
means ............... descriptive statistics by category of the dependent variable (only for binary, ordered, censored and count equations) (p. 336).
output ................ table of estimation results (p. 347).
predict ............. prediction (fit) evaluation table (only for binary and ordered equations) (p. 372).
representations .... text showing specification of the equation (p. 381).
reset .................. Ramsey’s RESET test for functional form (p. 383).
resids ............. display, in tabular form, the actual and fitted values for the dependent variable, along with the residuals (p. 385).
results ............ table of estimation results (p. 386).
rls ................ recursive residuals least squares (only for non-panel equations estimated by ordinary least squares, without ARMA terms) (p. 387).
testadd ........... likelihood ratio test for adding variables to equation (p. 461).
testdrop ........... likelihood ratio test for dropping variables from equation (p. 464).
testfit ........ performs Hosmer and Lemeshow and Andrews goodness-of-fit tests (only for equations estimated using binary) (p. 466).
wald ........ Wald test for coefficient restrictions (p. 490).
white ................ White test for heteroskedasticity (p. 495).

Equation Procs

displayname ....... set display name (p. 251).
fit .................... static forecast (p. 270).
forecast ........... dynamic forecast (p. 272).
makederivs .......... make group containing derivatives of the equation specification (p. 320).
makegarch.............create conditional variance series (only for ARCH equations) (p. 322).
makegrads.............make group containing gradients of the objective function (p. 322).
makelimits.............create vector of estimated limit points (only for ordered models) (p. 326).
makemodel.............create model from estimated equation (p. 327).
makeregs..............make group containing the regressors (p. 327).
makeresids...........make series containing residuals from equation (p. 328).
updatecoefs...........update coefficient vector(s) from equation (p. 480).

Equation Data Members

Scalar Values

@aic..................Akaike information criterion.
@coefcov(i,j) .........covariance of coefficient estimates \( i \) and \( j \).
@coefs(i)...............\( i \)-th coefficient value.
@dw ....................Durbin-Watson statistic.
@f .....................\( F \)-statistic.
@hq ....................Hannan-Quinn information criterion.
@jstat ...................\( J \)-statistic — value of the GMM objective function (for GMM).
@logl ...................value of the log likelihood function.
@meandep ..............mean of the dependent variable.
@ncoef ..................number of estimated coefficients.
@r2 .....................R-squared statistic.
@rbar2 .................adjusted R-squared statistic.
@regobs ..............number of observations in regression.
@schwarz ..............Schwarz information criterion.
@sddep ..................standard deviation of the dependent variable.
@se .....................standard error of the regression.
@ssr .....................sum of squared residuals.
@stderrs(i) ............standard error for coefficient \( i \).
@tstats(i) .............\( t \)-statistic value for coefficient \( i \).
c(i) .....................\( i \)-th element of default coefficient vector for equation (if applicable).

Vectors and Matrices

@coefcov ..............covariance matrix for coefficient estimates.
@coefs.............. coefficient vector.
@stderrs............. vector of standard errors for coefficients.
@tstats............... vector of t-statistic values for coefficients.

Equation Examples

To apply an estimation method (proc) to an existing equation object:

```
equation ifunc
   ifunc.ls r c r(-1) div
```

To declare and estimate an equation in one step, combine the two commands:

```
equation value.tsls log(p) c d(x) @ x(-1) x(-2)
equation drive.logit ifdr c owncar dist income
equation countmod.count patents c rdd
```

To estimate equations by list, using ordinary and two-stage least squares:

```
equation ordinary.ls log(p) c d(x)
equation twostage.tsls log(p) c d(x) @ x(-1) x(-2)
```

You can create and use other coefficient vectors:

```
coef(10) a
coef(10) b
```

```
equation eq01.ls y=c(10)+b(5)*y(-1)+a(7)*inc
```

The fitted values from EQ01 may be saved using

```
series fit = eq01.@coefs(1) + eq01.@coefs(2)*y(-1) + eq01.@coefs(3)*inc
```

or by issuing the command

```
eq01.fit fitted_vals
```

To perform a Wald test:

```
eq01.wald a(7)=exp(b(5))
```

You can save the t-statistics and covariance matrix for your parameter estimates:

```
vector eqstats=eq01.@tstats
matrix eqcov=eq01.@coefcov
```
Graph

Graph object. Specialized object used to hold graphical output.

Graph Declaration

freeze ................... freeze graphical view of object (p. 275).
graph .................... create graph object using graph command or by merging existing graphs (p. 288).

Graphs may be created by declaring a graph using one of the graph commands described below, or by freezing the graphical view of an object. For example:

graph myline.line ser1
graph myscat.scat ser1 ser2
graph myxy.xyline grp1

declare and create the graph objects MYLINE, MYSCAT and MYXY. Alternatively, you can use the freeze command to create graph objects:

freeze(myline) ser1.line
freeze(myscat) grp2.scat
freeze(myxy) grp1.xyline

which are equivalent to the declarations above.

Graph Type Commands

area ..................... area graph (p. 191).
bar ..................... bar graph (p. 199).
errbar ........................ error bar graph (p. 261).
hilo ........................ high-low(-open-close) graph (p. 292).
line ........................ line-symbol graph (p. 306).
pie ........................ pie chart (p. 370).
scat ........................ scatterplot (p. 399).
spike ........................ spike graph (p. 442).
xy .......................... XY graph with one or more X plotted against one or more Y (p. 510).
xyline .......................... XY line graph (p. 512).
xypair .......................... XY pairs graph (p. 510).

Graph View

label ...................... label information for the graph (p. 302).
Graph Procs

- `addtext`................. place arbitrary text on the graph (p. 183).
- `align`.................... align the placement of multiple graphs (p. 184).
- `axis` ..................... sets axis display characteristics for the graph (p. 197).
- `dates` ..................... controls labeling of the bottom date/time axis in time plots (p. 239).
- `datelabel` ................ controls labeling of the bottom date/time axis in time plots (p. 237).
- `displayname`.............. set display name (p. 251).
- `draw` ...................... draw lines and shaded areas on the graph (p. 252).
- `legend` .................... control the appearance and placement of legends (p. 304).
- `merge` ..................... merge graph objects (p. 336).
- `metafile` .................. save graph to a Windows metafile (p. 338).
- `name` ..................... change the series name for legends or axis labels (p. 340).
- `options` .................. change the option settings of the graph (p. 343).
- `save` ...................... save graph to a Windows metafile or PostScript file (p. 393).
- `scale` ..................... set the axis scaling for the graph (p. 397).
- `setbpelem` ............... set options for element of a boxplot graph (p. 407).
- `setelem` .................. set individual line, bar and legend options for each series in the graph (p. 412).
- `template` ............... use template graph (p. 460).

Graph Examples

You can declare your graph

```
graph abc.xyline(m) unemp gnp inf
graph bargraph.bar(d,l)unemp gnp
```

or freeze any graphical view:

```
freeze(mykernel) ser1.kdensity
```

You can change the graph type,

```
graph mygraph.line ser1
mygraph.hist
```

or combine multiple graphs,
Group

Group of series. Groups are used for working with collections of series objects (series, alphas, links).

Group Declaration

group....................create a group object (p. 290).

To declare a group, enter the keyword group, followed by a name, and optionally, a list of series or expressions:

```plaintext
group salesvrs

group nipa cons(-1) log(inv) g x
```

Additionally, a number of object procedures will automatically create a group.

Group Views

area ......................area graph of the series in the group (p. 191).
bar .......................single or multiple bar graph view of all series (p. 199).
boxplot ...................boxplot of each series in the group (p. 204).
cause ......................pairwise Granger causality tests (p. 212).
cdfplot ..................distribution (cumulative, survivor, quantile) graphs (p. 214).
coint .....................Johansen cointegration test (p. 224).
cor .......................correlation matrix between series (p. 230).
correl ....................correlogram of the first series in the group (p. 231).
cov ........................covariance matrix between series (p. 234).
cross .....................cross correlogram of the first two series (p. 235).
dtable ...................dated data table (p. 255).
errbar ......................error bar graph view (p. 261).
freq ........................frequency table n-way contingency table (p. 276).
hilo .......................high-low(-open-close) chart (p. 292).
kerfit ....................scatter of the first series against the second series with kernel fit (p. 301).
label ......................label information for the group (p. 302).
line .......................single or multiple line graph view of all series (p. 306).
linefit ....................scatter of the first series against the second series with regression line (p. 308).
nnfit ......................scatter of the first series against the second series with nearest neighbor fit (p. 341).
pie ....................... pie chart view (p. 370).
pcomp .................... principal components analysis (p. 369).
qqplot ..................... quantile-quantile plots (p. 375).
scat ....................... single scatter diagram of the series in the group (p. 399).
scatmat ................... matrix of all pairwise scatter plots (p. 401).
sheet ....................... spreadsheet view of the series in the group (p. 432).
spike ....................... spike graph (p. 442).
stats ....................... descriptive statistics (p. 449).
testbtw ..................... tests of equality for mean, median, or variance, between series in group (p. 462).
uroot ....................... unit root test on the series in the group (p. 481).
xy ......................... XY graph with one or more X series plotted against one or more Y (p. 510).
xyline ..................... XY line graph (p. 512).
xypair ..................... XY pairs graph (p. 510).

Group Procs
add ........................ add one or more series to the group (p. 179).
displayname ............... set display name (p. 251).
drop ........................ drop one or more series from the group (p. 255).
resample ..................... resample from rows of group (p. 381).
setformat ................. set the display format in the group spreadsheet for the specified series (p. 419).
setindent ................... set the indentation in the group spreadsheet for the specified series (p. 424).
setjust ..................... set the justification in the group spreadsheet for the specified series (p. 425).
setwidth ..................... set the column width in the group spreadsheet for the specified series (p. 431).

Group Data Members
(i) ........................ i-th series in the group. Simply append “(i)” to the group name (without a “.”).
@comobs ................... number of observations in the current sample for which each series in the group has a non-missing value (observations in the common sample).
@count ........................ number of series in the group.
@minobs ..........number of non-missing observations in the current sample for the shortest series in the group.

@maxobs ..........number of non-missing observations in the current sample for the longest series in the group.

@seriesname(i) ....string containing the name of the i-th series in the group.

**Group Examples**

To create a group G1, you can enter

```plaintext
group g1 gdp income
```

To change the contents of an existing group, you can repeat the declaration, or use the add and drop commands:

```plaintext
group g1 x y
g1.add w z
g1.drop y
```

The following commands produce a cross-tabulation of the series in the group, display the covariance matrix, and test for equality of variance:

```plaintext
g1.freq
g1.cov
g1.testbtw(var,c)
```

You can index selected series in the group:

```plaintext
show g1(2).line
series sum=g1(1)+g1(2)
```

To create a scalar containing the number of series in the group, use

```plaintext
scalar nsers=g1.@count
```

**Link**

Link object. Series or alpha link used to frequency converted or match merge data from another workfile page.

Once created, links may be used just like “Series” (p. 159) or “Alpha” (p. 136) objects.

**Link Declaration**

```plaintext
link .........................link object declaration (p. 309).
```

To declare a link object, enter the keyword `link`, followed by a name:
link newser

and an optional link specification:

\[\text{link altser.linkto(c=obs,nacat) indiv::x @src ind1 ind2 @dest ind1 ind2}\]

Link Procs

\texttt{linkto} \ldots \ldots \ldots \ldots \ldots specify link object definition (p. 310).

Logl

Likelihood object. Used for performing maximum likelihood estimation of user-specified likelihood functions.

Logl Declaration

\texttt{logl} \ldots \ldots \ldots likelihood object declaration (p. 314).

To declare a logl object, use the \texttt{logl} keyword, followed by a name to be given to the object.

Logl Method

\texttt{ml} \ldots \ldots \ldots maximum likelihood estimation (p. 338).

Logl Views

\texttt{append} \ldots \ldots \ldots \ldots add line to the specification (p. 186).
\texttt{ellipse} \ldots \ldots \ldots Confidence ellipses for coefficient restrictions (p. 215).
\texttt{checkderivs} \ldots \ldots \ldots compare user supplied and numeric derivatives (p. 219).
\texttt{coefcov} \ldots \ldots \ldots coefficient covariance matrix (p. 224).
\texttt{grads} \ldots \ldots \ldots examine the gradients of the log likelihood (p. 287).
\texttt{label} \ldots \ldots \ldots \ldots label view of likelihood object (p. 302).
\texttt{output} \ldots \ldots \ldots \ldots table of estimation results (p. 347).
\texttt{results} \ldots \ldots \ldots \ldots estimation results (p. 386).
\texttt{spec} \ldots \ldots \ldots \ldots likelihood specification (p. 441).
\texttt{wald} \ldots \ldots \ldots \ldots Wald coefficient restriction test (p. 490).

Logl Procs

\texttt{displayname} \ldots \ldots set display name (p. 251).
\texttt{makegrads} \ldots \ldots make group containing gradients of the log likelihood (p. 322).
\texttt{makemodel} \ldots \ldots make model (p. 327).
\texttt{updatecoefs} \ldots \ldots update coefficient vector(s) from likelihood (p. 480).
Logl Statements

The following statements can be included in the specification of the likelihood object. These statements are optional, except for "@logl" which is required. See Chapter 22, “The Log Likelihood (LogL) Object”, on page 641 of the User’s Guide for further discussion.

- @byeqn: evaluate specification by equation.
- @byobs: evaluate specification by observation (default).
- @deriv: specify an analytic derivative series.
- @derivstep: set parameters to control step size.
- @logl: specify the likelihood contribution series.
- @param: set starting values.
- @temp: remove temporary working series.

Logl Data Members

Scalar Values (system data)

- @aic: Akaike information criterion.
- @coefcov(i,j): covariance of coefficients i and j.
- @coefs(i): coefficient i.
- @hq: Hannan-Quinn information criterion.
- @logl: value of the log likelihood function.
- @ncoefs: number of estimated coefficients.
- @regobs: number of observations used in estimation.
- @sc: Schwarz information criterion.
- @stderrs(i): standard error for coefficient i.
- @tstats(i): t-statistic value for coefficient i.
- coef_name(i): i-th element of default coefficient vector for likelihood.

Vectors and Matrices

- @coefcov: covariance matrix of estimated parameters.
- @coefs: coefficient vector.
- @stderrs: vector of standard errors for coefficients.
- @tstats: vector of t-statistic values for coefficients.

Logl Examples

To declare a likelihood named LL1:

    logl ll1

To define a likelihood function for OLS (not a recommended way to do OLS!):

    ll1.append @logl logl1
ll1.append res1 = y-c(1)-c(2)*x
ll1.append logl1 = log(@dnorm(res1/@sqrt(c(3))))-log(c(3))/2

To estimate LL1 by maximum likelihood (the “showstart” option displays the starting values):

ll1.ml(showstart)

To save the estimated covariance matrix of the parameters from LL1 as a named matrix COV1:

matrix cov1=ll1.@coefcov

Matrix (two-dimensional array).

Matrix Declaration

matrix................. declare matrix object (p. 335).

There are several ways to create a matrix object. You can enter the `matrix` keyword (with an optional row and column dimension) followed by a name:

matrix scalarmat
matrix(10,3) results

Alternatively, you can combine a declaration with an assignment statement, in which case the new matrix will be sized accordingly.

Lastly, a number of object procedures create matrices.

Matrix Views

area..................... area graph of the columns in the matrix (p. 191).
bar....................... single or multiple bar graph of each column against the row index (p. 199).
cor......................... correlation matrix by columns (p. 230).
cov........................ covariance matrix by columns (p. 234).
errbar.................... error bar graph view (p. 261).
hilo....................... high-low(-open-close) chart (p. 292).
label...................... label information for the matrix (p. 302).
line....................... single or multiple line graph of each column by the row index (p. 306).
pie......................... pie chart view (p. 370).
crat....................... scatter diagrams of the columns of the matrix (p. 399).
Matrix—151

sheet ..................spreadsheet view of the matrix (p. 432).
spike ..................spike graph (p. 442).
stats....................descriptive statistics by column (p. 449).
xy......................XY graph with one or more X columns plotted against one or more Y (p. 510).
xyline .................XY line graph (p. 512).
xypair ...............XY pairs graph (p. 510).

Matrix Procs

displayname ..........set display name (p. 251).
fill ....................fill the elements of the matrix (p. 267).
read ....................import data from disk (p. 377).
setformat .............set the display format for the matrix spreadsheet (p. 419).
setindent .............set the indentation for the matrix spreadsheet (p. 424).
setjust ................set the justification for the matrix spreadsheet (p. 425).
setwidth ..............set the column width in the matrix spreadsheet (p. 431).
write ....................export data to disk (p. 498).

Matrix Data Members

(i,j) ...............(i,j)-th element of the matrix. Simply append “(i, j)” to the matrix name (without a “.”).

Matrix Examples

The following assignment statements create and initialize matrix objects:

```plaintext
matrix copymat=results
matrix covmat1=eq1.@coefcov
matrix(5,2) count
count.fill 1,2,3,4,5,6,7,8,9,10
```

as does the procedure

```plaintext
eq1.makecoefcov covmat2
```

You can declare and initialize a matrix in one command:

```plaintext
matrix(10,30) results=3
matrix(5,5) other=results1
```

Graphs and covariances may be generated for the columns of the matrix:

```plaintext
copymat.line
copymat.cov
```
and statistics computed for the rows of a matrix

\[
\text{matrix } \text{rowmat}=@\text{transpose}(\text{copymat}) \\
\text{rowmat.stats}
\]

You can use explicit indices to refer to matrix elements:

\[
\text{scalar } \text{diagsum}=\text{cov1}(1,1)+\text{cov1}(2,2)+\text{cov}(3,3)
\]

**Model**

Set of simultaneous equations used for forecasting and simulation.

**Model Declaration**

\[
\text{model } \ldots \ldots \ldots \text{ declare model object (p. 339)}.
\]

Declare an object by entering the keyword `model`, followed by a name:

\[
\text{model mymod}
\]

declares an empty model named MYMOD. To fill MYMOD, open the model and edit the specification view, or use the `append` view. Note that models are not used for estimation of unknown parameters.

See also the section on model keywords in “Text View” on page 772 of the *User’s Guide*.

**Model Views**

\[
\text{block } \ldots \ldots \text{ display model block structure (p. 203)}.
\]
\[
\text{eqs} \ldots \ldots \text{ view of model organized by equation (p. 260)}.
\]
\[
\text{label} \ldots \ldots \text{ view or set label information for the model (p. 302)}.
\]
\[
\text{msg} \ldots \ldots \text{ display model solution messages (p. 339)}.
\]
\[
\text{text} \ldots \ldots \text{ show text showing equations in the model (p. 469)}.
\]
\[
\text{trace} \ldots \ldots \text{ view of trace output from model solution (p. 471)}.
\]
\[
\text{vars} \ldots \ldots \text{ view of model organized by variable (p. 488)}.
\]

**Model Procs**

\[
\text{addassign} \ldots \text{ assign add factors to equations (p. 180)}.
\]
\[
\text{addinit} \ldots \text{ initialize add factors (p. 181)}.
\]
\[
\text{append} \ldots \text{ append a line of text to a model (p. 186)}.
\]
\[
\text{control} \ldots \text{ solve for values of control variable so that target matches trajectory (p. 227)}.
\]
\[
\text{displayname} \ldots \text{ set display name (p. 251)}.
\]
\[
\text{exclude} \ldots \text{ specifies (or merges) excluded series to the active scenario (p. 263)}.
\]
makegraph makes graph object showing model series (p. 323).
makegroup makes group out of model series and display dated data table (p. 324).
merge merge objects into the model (p. 336).
override specifies (or merges) override series to the active scenario (p. 349).
scenario set the active, alternate, or comparison scenario (p. 401).
solve solve the model (p. 438).
solveopt set solve options for model (p. 439).
spec display the text specification view (p. 441).
unlink break links in specification (p. 479).
update update model specification (p. 480).

Model Examples
The commands

\begin{verbatim}
model mod1
mod1.append y=324.35+x
mod1.append x=-234+7.3*z
mod1.solve(m=100,c=.008)
\end{verbatim}

create, specify, and solve the model MOD1. The command

\begin{verbatim}
mod1(g).makegraph gr1 x y z
\end{verbatim}

plots the endogenous series X, Y, and Z, in the active scenario for model MOD1.

Pool

Pooled time series, cross-section object. Used when working with data with both time series and cross-section structure.

Pool Declaration

\begin{verbatim}
pool declare pool object (p. 372).
\end{verbatim}

To declare a pool object, use the pool keyword, followed by a pool name, and optionally, a list of pool members. Pool members are short text identifiers for the cross section units:

\begin{verbatim}
pool mypool
pool g7_can_fr_ger_ita_jpn_us_uk
\end{verbatim}
Pool Methods

**ls** ................. estimate linear regression models including cross-section weighted least squares, and fixed and random effects models (p. 315).

**tsls** ................... linear two-stage least squares (TSLS) regression models (p. 474).

Pool Views

**cellipse** ........... Confidence ellipses for coefficient restrictions (p. 215).

**coefcov** ............ coefficient covariance matrix (p. 224).

**describe** .......... calculate pool descriptive statistics (p. 250).

**label** ............... label information for the pool object (p. 302).

**output** ............. table of estimation results (p. 347).

**representations** ..... text showing equations in the model (p. 381).

**residcor** .......... residual correlation matrix (p. 384).

**residcov** .......... residual covariance matrix (p. 385).

**resids** ............. table or graph of residuals for each pool member (p. 385).

**results** ............ table of estimation results (p. 386).

**sheet** ............... spreadsheet view of series in pool (p. 432).

**uroot** ............... unit root test on a pool series (p. 481).

**wald** ................. Wald coefficient restriction test (p. 490).

Pool Procs

**add** ................ add cross section members to pool (p. 179).

**define** ............. define cross section identifiers (p. 247).

**delete** .............. delete pool series (p. 248).

**displayname** ........ set display name (p. 251).

**drop** ................ drop cross section members from pool (p. 255).

**fetch** ............... fetch series into workfile using a pool (p. 264).

**genr** ................ generate pool series using the “?” (p. 281).

**makegroup** ........ create a group of series from a pool (p. 324).

**makemodel** ........ creates a model object from the estimated pool (p. 327).

**makeresids** ......... make series containing residuals from pool (p. 328).

**makestats** ........... make descriptive statistic series (p. 332).

**makesystem** ......... creates a system object from the pool for other estimation methods (p. 333).

**read** ................ import pool data from disk (p. 377).

**store** ............... store pool series in database/bank files (p. 452).

**updatecoefs** ....... update coefficient vector from pool (p. 480).
write  export pool data to disk (p. 498).

Pool Data Members

String Values

@idname(i) ..........i-th cross-section identifier.

Scalar Values

@aic ..................Akaike information criterion.
@coeffcov(i,j) ........covariance of coefficients i and j.
@coefs(i) ..............coefficient i.
@d2w ..................Durbin-Watson statistic.
@effects(i) ............estimated fixed or random effect for the i-th cross-section member (only for fixed or random effects).
@f .....................F-statistic.
@logl ..................log likelihood.
@meandep ...............mean of the dependent variable.
@ncross .................total number of cross sectional units.
@ncrossest .............number of cross sectional units in last estimated pool equation.
@r2 ....................R-squared statistic.
@rbar2 ..................adjusted R-squared statistic.
@regobs .................total number of observations in regression.
@schwarz ...............Schwarz information criterion.
@sddep ..................standard deviation of the dependent variable.
@se .....................standard error of the regression.
@ssr ...................sum of squared residuals.
@stderrs(i) .............standard error for coefficient i.
@totalobs ..............total number of observations in the pool. For a balanced sample this is “@regobs*ncrossest”.
@tstats(i) .............t-statistic value for coefficient i.
c(i) .....................i-th element of default coefficient vector for the pool.

Vectors and Matrices

@coeffcov ..............covariance matrix for coefficients of equation.
@coefs .................coefficient vector.
@effects .................vector of estimated fixed or random effects (only for fixed or random effects estimation).
@stderrs ...................vector of standard errors for coefficients.
@tstats ................ vector of \( t \)-statistic values for coefficients.

Pool Examples

To read data using the pool object

\[
\text{mypool1.read(b2) data.xls x? y? z?}
\]

and to delete and store pool series you may enter

\[
\text{mypool1.delete x? y?}
\]
\[
\text{mypool1.store z?}
\]

Descriptive statistics may be computed using the command

\[
\text{mypool1.describe(m) z?}
\]

To estimate a pool equation using least squares and to access the \( t \)-statistics, enter

\[
\text{mypool1.ls y? c z? @ w?}
\]
\[
\text{vector tstat1 = mypool1.@tstats}
\]

Rowvector

Row vector. (One dimensional array of numbers).

Rowvector Declaration

\[
\text{rowvector} \ldots \ldots \text{declare rowvector object (p. 390)}.
\]

There are several ways to create a rowvector object. First, you can enter the rowvector keyword (with an optional dimension) followed by a name:

\[
\text{rowvector scalarmat}
\]
\[
\text{rowvector(10) results}
\]

The resulting rowvector will be initialized with zeros.

Alternatively, you may combine a declaration with an assignment statement. The new vector will be sized and initialized accordingly:

\[
\text{rowvector(10) y=3}
\]
\[
\text{rowvector z=results}
\]

Rowvector Views

\[
\text{area} \ldots \ldots \text{area graph of the vector (p. 191)}.
\]
\[
\text{bar} \ldots \ldots \text{bar graph of each column (element) of the data against the row index (p. 199)}.
\]
\[
\text{errbar} \ldots \ldots \text{error bar graph view (p. 261)}.
\]
label .......................label information for the rowvector (p. 302).
line .......................line graph of each column (element) of the data against the row index (p. 306).
scat .......................scatter diagrams of the columns of the rowvector (p. 399).
sheet ......................spreadsheet view of the vector (p. 432).
spike .....................spike graph (p. 442).
stats ......................(trivial) descriptive statistics (p. 449).

Rowvector Procs

displayname ........set display name (p. 251).
fill .......................fill elements of the vector (p. 267).
read ......................import data from disk (p. 377).
setformat ..........set the display format for the vector spreadsheet (p. 419).
setindent ............set the indentation for the vector spreadsheet (p. 424).
setjust ..............set the justification for the vector spreadsheet (p. 425).
setwidth ................set the column width in the vector spreadsheet (p. 431).
write ....................export data to disk (p. 498).

Rowvector Data Members

(i) ......................i-th element of the vector. Simply append “(i)” to the matrix name (without a “.”).

Rowvector Examples

To declare a rowvector and to fill it with data read from an Excel file:

rowvector(10) mydata
mydata.read(b2) thedata.xls

To access a single element of the vector using direct indexing:

scalar result1=mydata(2)

The rowvector may be used in standard matrix expressions:

vector transdata=@transpose(mydata)

Sample

Sample of observations. Description of a set of observations to be used in operations.

Sample Declaration

sample ....................declare sample object (p. 392).
To declare a sample object, use the keyword `sample`, followed by a name and a sample string:

```plaintext
samplemysample 1960:1 1990:4
sample altsample 120 170 300 1000 if x>0
```

**Sample Procs**

- **set** ......................... reset the sample range (p. 407).

**Sample Example**

To change the observations in a sample object, you can use the `set` proc:

```plaintext
mysample.set 1960:1 1980:4 if y>0
sample thesamp 1 10 20 30 40 60 if x>0
thesamp.set @all
```

To set the current sample to use a sample, enter a `smpl` statement, followed by the name of the sample object:

```plaintext
smpl mysample
```

**Scalar**

Scalar (single number). A scalar holds a single numeric value. Scalar values may be used in standard EViews expressions in place of numeric values.

**Scalar Declaration**

- **scalar** ...................... declare scalar object (p. 396).

To declare a scalar object, use the keyword `scalar`, followed by a name, an “=” sign and a scalar expression or value.

Scalar objects have no views or procedures, and do not open windows. The value of the scalar may be displayed in the status line at the bottom of the EViews window.

**Scalar Examples**

You can declare a scalar and examine its contents in the status line:

```plaintext
scalar pi=3.14159
scalar shape=beta(7)
show shape
```

or you can declare a scalar and use it in an expression:

```plaintext
scalar inner=@transpose(mydata)*mydata
```
series x=1/@sqrt(inner)*y

Series

Series of numeric observations. An EViews series contains a set of observations on a numeric variable.

Series Declaration

frml ......................create numeric series object with a formula for auto-updating (p. 278).
genr ......................create numeric series object (p. 281).
series ....................declare numeric series object (p. 405).

To declare a series, use the keyword series or alpha followed by a name, and optionally, by an “=” sign and a valid numeric series expression:

series y
genr x=3*z

If there is no assignment, the series will be initialized to contain NAs.

Series Views

area ......................area graph of the series (p. 191).
bar .......................bar graph of the series (p. 199).
bdstest ..................BDS independence test (p. 201).
boxplotby .............boxplot by classification (p. 205).
cdfplot ..................distribution (cumulative, survivor, quantile) functions (p. 214).
correl ....................correlogram, autocorrelation and partial autocorrelation functions (p. 231).
edftest ..................empirical distribution function tests (p. 257).
freq ...................one-way tabulation (p. 276).
hist ....................descriptive statistics and histogram (p. 294).
kdensity ...............kernel density estimate (p. 300).
label ....................label information for the series (p. 302).
line .....................line graph of the series (p. 306).
qqplot ..................quantile-quantile plot (p. 375).
seasplot .................seasonal line graph (p. 404).
sheet .....................spreadsheet view of the series (p. 432).
spike ....................spike graph (p. 442).
statby ...................statistics by classification (p. 444).
stats ......................descriptive statistics table (p. 294).
testby. equality test by classification (p. 462).

teststat. simple hypothesis tests (p. 468).

uroot. unit root test on an ordinary or panel series (p. 481).

Series Procs

displayname. set display name (p. 251).

fill. fill the elements of the series (p. 267).


map. assign or remove value map setting (p. 334).

resample. resample from the observations in the series (p. 381).

seas. seasonal adjustment for quarterly and monthly time series (p. 403).

setconvert. set default frequency conversion method (p. 410).

setformat. set the display format for the series spreadsheet (p. 419).

setindent. set the indentation for the series spreadsheet (p. 424).

setjust. set the justification for the series spreadsheet (p. 425).

setwidth. set the column width in the series spreadsheet (p. 431).

smooth. exponential smoothing (p. 434).

tramoseats. seasonal adjustment using Tramo/Seats (p. 471).

x11. seasonal adjustment by Census X11 method for quarterly and monthly time series (p. 502).

x12. seasonal adjustment by Census X12 method for quarterly and monthly time series (p. 504).

Series Data Members

(i). i-th element of the series from the beginning of the workfile (when used on the left-hand side of an assignment, or when the element appears in a matrix, vector, or scalar assignment).

Series Element Functions

@elem(ser, j) function to access the j-th observation of the series SER, where j identifies the date or observation.

Series Examples

You can declare a series in the usual fashion:

series b=income*@mean(z)

series blag=b(1)
Note that the last example, above, involves a series expression so that B(1) is treated as a one-period lead of the entire series, not as an element operator. In contrast,

scalar blag1=b(1)

evaluates the first observation on B in the workfile.

Once a series is declared, views and procs are available:

a.qqplot
a.statby(mean, var, std) b

To access individual values:

scalar quarterlyval = @elem(y, "1980:3")
scalar undatedval = @elem(x, 323)

Sspace

State space object. Estimation and evaluation of state space models using the Kalman filter.

Sspace Declaration

sspace........................create sspace object (p. 444).

To declare a sspace object, use the sspace keyword, followed by a valid name.

Sspace Method

ml ........................maximum likelihood estimation or filter initialization (p. 338).

Sspace Views

cellipse ..................Confidence ellipses for coefficient restrictions (p. 215).
coeffcov..................coefficient covariance matrix (p. 224).
endog ....................table or graph of actual signal variables (p. 259).
grads .....................examine the gradients of the log likelihood (p. 287).
label .....................label information for the state space object (p. 302).
output ....................table of estimation results (p. 347).
residcor ..................standardized one-step ahead residual correlation matrix (p. 384).
residcov ..................standardized one-step ahead residual covariance matrix (p. 385).
resids ...................one-step ahead actual, fitted, residual graph (p. 385).
results ..................table of estimation and filter results (p. 386).
signalgraphs ............display graphs of signal variables (p. 433).
spec ......................text representation of state space specification (p. 441).
statefinal ...............display the final values of the states or state covariance (p. 447).
stategraphs display graphs of state variables (p. 446).
stateinit display the initial values of the states or state covariance (p. 448).
structure examine coefficient or variance structure of the specification (p. 454).
wald Wald coefficient restriction test (p. 490).

Sspace Procs

append add line to the specification (p. 186).
displayname set display name (p. 251).
forecast perform state and signal forecasting (p. 272).
makeendog make group containing actual values for signal variables (p. 321).
makefilter make new Kalman Filter (p. 321).
makegrads make group containing the gradients of the log likelihood (p. 322).
makemodel make a model object containing equations in sspace (p. 327).
makesignals make group containing signal and residual series (p. 330).
makestates make group containing state series (p. 331).
sspace declare sspace object (p. 444).
updatecoefs update coefficient vector(s) from sspace (p. 480).

Sspace Data Members

Scalar Values
@coefcov(i,j) covariance of coefficients i and j.
@coefs(i) coefficient i.
@eqregobs(k) number of observations in signal equation k.
@sddep(k) standard deviation of the signal variable in equation k.
@ssr(k) sum-of-squared standardized one-step ahead residuals for equation k.
@stderrs(i) standard error for coefficient i.
@tstats(i) t-statistic value for coefficient i.

Scalar Values (system level data)
@aic Akaike information criterion for the system.
@hq Hannan-Quinn information criterion for the system.
@logl value of the log likelihood function.
@ncoefs total number of estimated coefficients in the system.
@neqns number of equations for observable variables.
@regobs number of observations in the system.
@sc Schwarz information criterion for the system.
Vectors and Matrices

@totalobs .......... sum of "@eqregobs" from each equation.

@coefcov .......... covariance matrix for coefficients of equation.
@coefs ............ coefficient vector.
@stderrs .......... vector of standard errors for coefficients.
@tstats ............ vector of $t$-statistic values for coefficients.

State and Signal Results

The following functions allow you to extract the filter and smoother results for the estimation sample and place them in matrix objects. In some cases, the results overlap those available thorough the sspace procs, while in other cases, the matrix results are the only way to obtain the results.

Note also that since the computations are only for the estimation sample, the one-step-ahead predicted state and state standard error values will not match the final values displayed in the estimation output. The latter are the predicted values for the first out-of-estimation sample period.

@pred_signal ...... matrix or vector of one-step ahead predicted signals.
@pred_signalcov .. matrix where every row is the @vech of the one-step ahead predicted signal covariance.
@pred_signalse ... matrix or vector of the standard errors of the one-step ahead predicted signals.
@pred_err .......... matrix or vector of one-step ahead prediction errors.
@pred_errcov ...... matrix where every row is the @vech of the one-step ahead prediction error covariance.
@pred_errcovinv .. matrix where every row is the @vech of the inverse of the one-step ahead prediction error covariance.
@pred_errse ...... matrix or vector of the standard errors of the one-step ahead prediction errors.
@pred_errstd ...... matrix or vector of standardized one-step ahead prediction errors.
@pred_state ...... matrix or vector of one-step ahead predicted states.
@pred_statecov ... matrix where each row is the @vech of the one-step ahead predicted state covariance.
@pred_statese ...... matrix or vector of the standard errors of the one-step ahead predicted states.
@pred_stateerr .... matrix or vector of one-step ahead predicted state errors.
@curr_err .......... matrix or vector of filtered error estimates.
@curr_gain ......... matrix or vector where each row is the @vec of the Kalman gain.
@curr_state ........ matrix or vector of filtered states.
@curr_statecov .... matrix where every row is the @vech of the filtered state covariance.
@curr_statese ...... matrix or vector of the standard errors of the filtered state estimates.
@sm_signal .......... matrix or vector of smoothed signal estimates.
@sm_signalcov .... matrix where every row is the @vech of the smoothed signal covariance.
@sm_state .......... matrix or vector of smoothed states.
@sm_statecov ...... matrix where each row is the @vech of the smoothed state covariances.
@sm_stateerr ....... matrix or vector of the smoothed state errors.
@sm_stateerrcov .... matrix where each row is the @vech of the smoothed state error covariance.
@sm_stateerrstd ... matrix or vector of the standardized smoothed state errors.
@sm_crosserrcov . matrix where each row is the @vec of the smoothed error cross-covariance.

Sspace Examples

The one-step-ahead state values and variances from SS01 may be saved using

    vector ss_state=ss01.@pred_state
    matrix ss_statecov=ss01.@pred_statecov

Sym

Symmetric matrix (symmetric two-dimensional array).

Sym Declaration

    sym ................ declare sym object (p. 458).
Declare by providing a name after the `sym` keyword, with the optionally specified dimension in parentheses:

```
sym(10) symmatrix
```

You may optionally assign a scalar, a square matrix or another sym in the declaration. If the square matrix is not symmetric, the sym will contain the lower triangle. The sym will be sized and initialized accordingly.

**Sym Views**

- **area** .................area graph of the columns of the matrix (p. 191).
- **bar** ....................single or multiple bar graph of each column against the row index (p. 199).
- **cor** .....................correlation matrix by columns (p. 230).
- **cov** ....................covariance matrix by columns (p. 234).
- **errbar** ..................error bar graph view (p. 261).
- **hilo** ....................high-low(-open-close) chart (p. 292).
- **label** ..................label information for the symmetric matrix (p. 302).
- **line** ....................single or multiple line graph of each column against the row index (p. 306).
- **pie** ......................pie chart view (p. 370).
- **scat** ....................scatter diagrams of the columns of the sym (p. 399).
- **sheet** ..................spreadsheet view of the symmetric matrix (p. 432).
- **spike** ...................spike graph (p. 442).
- **stats** ...................descriptive statistics by column (p. 449).
- **xy** ......................XY graph with one or more X columns plotted against one or more Y (p. 510).
- **xyline** .................XY line graph (p. 512).
- **xypair** ..................XY pairs graph (p. 510).

**Sym Procs**

- **displayname** ........set display name (p. 251).
- **fill** ....................fill the elements of the matrix (p. 267).
- **read** ....................import data from disk (p. 377).
- **setformat** ..............set the display format for the sym spreadsheet (p. 419).
- **setindent** ..............set the indentation for the sym spreadsheet (p. 424).
- **setjust** .................set the justification for the sym spreadsheet (p. 425).
- **setwidth** ...............set the column width in the sym spreadsheet (p. 431).
- **write** ....................export data to disk (p. 498).
Sym Data Members

\((i,j)\) \(\cdots\) \((i,j)\)-th element of the matrix. Simply append “\((i,j)\)” to the matrix name (without a “.”).

Sym Examples

The declaration

\[
\text{sym results}(10)
\]
\[
\text{results}=3
\]
creates the \(10 \times 10\) matrix RESULTS and initializes each value to be 3. The following assignment statements also create and initialize sym objects:

\[
\text{sym copymat}=\text{results}
\]
\[
\text{sym covmat1=eq1.@coefcov}
\]
\[
\text{sym(3,3) count}
\]
\[
\text{count.fill 1,2,3,4,5,6,7,8,9,10}
\]

Graphs, covariances, and statistics may be generated for the columns of the matrix:

\[
\text{copymat.line}
\]
\[
\text{copymat.cov}
\]
\[
\text{copymat.stats}
\]

You can use explicit indices to refer to matrix elements:

\[
\text{scalar diagsum}=\text{cov1}(1,1)+\text{cov1}(2,2)+\text{cov}(3,3)
\]

System

System of equations for estimation.

System Declaration

\[
\text{system} \cdots \text{declare system object (p. 459).}
\]

Declare a system object by entering the keyword \text{system}, followed by a name:

\[
\text{system mysys}
\]

To fill a system, open the system and edit the specification view, or use \text{append}. Note that systems are not used for simulation. See “Model” (p. 152).

System Methods

\text{3sls} \cdots \text{three-stage least squares (p. 178).}
\text{fiml} \cdots \text{full information maximum likelihood (p. 269).}
\text{gmm} \cdots \text{generalized method of moments (p. 282).}
ls.........................ordinary least squares (p. 315).
sur.........................seemingly unrelated regression (p. 455).
tsls.........................two-stage least squares (p. 474).
wls.........................weighted least squares (p. 497).
wtsls.......................weighted two-stage least squares (p. 501).

System Views

cellipse.................Confidence ellipses for coefficient restrictions (p. 215).
coefcov....................coefficient covariance matrix (p. 224).
derivs......................derivatives of the system equations (p. 249).
dendg......................table or graph of endogenous variables (p. 259).
grads......................examine the gradients of the objective function (p. 287).
label.....................label information for the system object (p. 302).
output....................table of estimation results (p. 347).
residcor..................residual correlation matrix (p. 384).
residcov..................residual covariance matrix (p. 385).
resids.....................residual graphs (p. 385).
results..................table of estimation results (p. 386).
spec......................text representation of system specification (p. 441).
wald.....................Wald coefficient restriction test (p. 490).

System Procs

append....................add a line of text to the system specification (p. 186).
displayname...............set display name (p. 251).
makeendog...............make group of endogenous series (p. 321).
makemodel...............create a model from the estimated system (p. 327).
makeresids...............make series containing residuals from system (p. 328).
updatecoefs.............update coefficient vector(s) from system (p. 480).

System Data Members

Scalar Values (individual equation data)

@coefcov(i, j)........covariance of coefficients i and j.
@coefs(i)..............coefficient i.
@dwr(k)...............Durbin-Watson statistic for equation k.
@eqncosf(k)...........number of estimated coefficients in equation k.
@eqreobs(k)...........number of observations in equation k.
@meandep(k).........mean of the dependent variable in equation k.
@ncoef(k) ........... total number of estimated coefficients in equation k.
@r2(k) ............... R-squared statistic for equation k.
@rbar2(k) .......... adjusted R-squared statistic for equation k.
@sddep(k) .......... standard deviation of dependent variable in equation k.
@se(k) .............. standard error of the regression in equation k.
@ssr(k) ............. sum of squared residuals in equation k.
@stderrs(i) .......... standard error for coefficient i.
@tstats(i) .......... t-statistic for coefficient i.
c(i) ..................... i-th element of default coefficient vector for system (if applicable).

Scalar Values (system level data)
@aic .................. Akaike information criterion for the system (if applicable).
@detresid ............ determinant of the residual covariance matrix.
@hq .................. Hannan-Quinn information criterion for the system (if applicable).
@jstat ................ J-statistic — value of the GMM objective function (for GMM estimation).
@logl ................ value of the log likelihood function for the system (if applicable).
@ncoefs ............. total number of estimated coefficients in system.
@neqn ............... number of equations.
@regobs ............. number of observations in the sample range used for estimation
(“@regobs” will differ from “@eqregobs” if the unbalanced sample is non-overlapping).
@schwarz ............ Schwarz information criterion for the system (if applicable).
@totalobs ........... sum of “@eqregobs” from each equation.

Vectors and Matrices
@coefcov ............ covariance matrix for coefficients of equation.
@coefs ............... coefficient vector.
@stderrs ............. vector of standard errors for coefficients.
@tstats .............. vector of t-statistic values for coefficients.

System Examples
To estimate a system using GMM and to create residual series for the estimated system:
sys1.gmm(i,m=7,c=.01,b=v)
sys1.makeresids consres incres saveres
To test coefficients using a Wald test:
sys1.wald c(1)=c(4)

To save the coefficient covariance matrix:

```r
sym covs=sys1.@coefcov
```

**Table**

*Table object. Formatted two-dimensional table for output display.*

**Table Declaration**

```r
freeze .....................freeze tabular view of object (p. 275).
table ......................create table object (p. 444).
```

To declare a table object, use the keyword `table`, followed by an optional row and column dimension, and then the object name:

```r
table onelement
table(10,5) outtable
```

If no dimension is provided, the table will contain a single element.

Alternatively, you may declare a table using an assignment statement. The new table will be sized and initialized, accordingly:

```r
table newtable=outtable
```

Lastly, you may use the `freeze` command to create tables from tabular views of other objects:

```r
freeze(newtab) ser1.freq
```

**Table Views**

```r
label .....................label information for the table object (p. 302).
sheet ......................view the table (p. 432).
```

**Table Procs**

```r
comment ....................adds or removes a comment in a table cell (p. 226).
setfont .....................set the font for the text in a set of table cells (p. 418).
setheight ...................set the row height in a set of table cells (p. 423).
setfillcolor ................set the fill (background) color of a set of table cells (p. 416).
```

```r
setformat ..................set the display format of a set of table cells (p. 419).
save ......................save table as CSV, tab-delimited ASCII text, RTF, or HTML file on disk (p. 393).
```
setindent .............. set the indentation for a set of table cells (p. 424).
setjust ............... set the justification for a set of table cells (p. 425).
setlines ............... set the line characteristics and borders for a set of table cells (p. 427).
setmerge .............. merge or unmerge a set of table cells (p. 428).
settextcolor .......... set the text color in a set of table cells (p. 430).
setwidth .............. set the column width for a set of table cells (p. 431).

Table Data Members
(i,j) ...................... the (i,j)-th element of the table, formatted as a string.

Table Commands
setcell ................. format and fill in a table cell (p. 408).
setcolwidth ............ set width of a table column (p. 410).
setline .................. place a horizontal line in table (p. 426).

Table Examples
\[ \text{table}(5,5) \text{ mytable} \]
\[ \% \text{strval} = \text{mytable}(2,3) \]
\[ \text{mytable}(4,4) = \text{"R2"} \]
\[ \text{mytable}(4,5) = \text{@str(eq1.@r2)} \]

Text

Text object.

Object for holding arbitrary text information.

Text Declaration
text ...................... declare text object (p. 469).

To declare a text object, use the keyword text, followed by the object name:

\[ \text{text mytext} \]

Text Views
label ..................... label information for the valmap object (p. 302).
text ....................... view contents of text object (p. 469).

Text Examples
text mytext
[add text to the object]
Valmap

Valmap (value map).

Valmap Declaration

valmap declare valmap object (p. 487).

To declare a valmap use the keyword valmap, followed by a name

valmap mymap

Valmap Views

label label information for the valmap object (p. 302).
sheet view table of map definitions (p. 432).
stats summary of map definitions (p. 449).
usage list of series and alphas which use the map (p. 486).

Valmap Procs

append append a definition to a valmap (p. 186).
displayname set display name (p. 251).

Valmap Examples

Var

Vector autoregression and error correction object.

Var Declaration

var declare var estimation object (p. 487).

To declare a var use the keyword var, followed by a name and, optionally, by an estimation specification:

var finvar
  var empvar.ls 1 4 payroll hhold gdp
  var finec.ec(e,2) 1 6 cp div r

Var Methods

ec estimate a vector error correction model (p. 256).
ls estimate an unrestricted VAR (p. 315).

Var Views

arlm serial correlation LM test (p. 193).
arroots .................. inverse roots of the AR polynomial (p. 195).
coint ................. Johansen cointegration test (p. 224).
corr .................. residual autocorrelations (p. 231).
decomp ............... variance decomposition (p. 245).
endog ................. table or graph of endogenous variables (p. 259).
impulse .............. impulse response functions (p. 296).
jbera ................ residual normality test (p. 299).
label ................ label information for the var object (p. 302).
laglen ............... lag order selection criteria (p. 303).
output ............... table of estimation results (p. 347).
qstats ................ residual portmanteau tests (p. 376).
representations .... text describing var specification (p. 381).
residcor ............. residual correlation matrix (p. 384).
residcov ............. residual covariance matrix (p. 385).
resids ............... residual graphs (p. 385).
results ............. table of estimation results (p. 386).
testexog ............ exogeneity (Granger causality) tests (p. 465).
testlags ............. lag exclusion tests (p. 467).
white ............... White heteroskedasticity test (p. 495).

Var Procs
append ............... append restriction text (p. 186).
cleartext ............ clear restriction text (p. 221).
displayname ........ set display name (p. 251).
makecoint ........... make group of cointegrating relations (p. 319).
makeendog .......... make group of endogenous series (p. 321).
makemodel .......... make model from the estimated var (p. 327).
makeresids .......... make residual series (p. 328).
makesystem .......... make system from var (p. 333).
svar ................. estimate structural factorization (p. 456).

Var Data Members
Scalar Values (individual level data)
@eqlogl(k) .......... log likelihood for equation k.
@eqncoef(k) ........ number of estimated coefficients in equation k.
@eqregobs(k) ....... number of observations in equation k.
@meandep(k) ....... mean of the dependent variable in equation k.
@r2(k) ..........R-squared statistic for equation k.
@rbar2(k) ..........adjusted R-squared statistic for equation k.
@sdep(k) ..........std. dev. of dependent variable in equation k.
@se(k) ..........standard error of the regression in equation k.
@ssr(k) ..........sum of squared residuals in equation k.

a(i,j) ..............adjustment coefficient for the j-th cointegrating equation in the i-th equation of the VEC (where applicable).
b(i,j) ..............coefficient of the j-th variable in the i-th cointegrating equation (where applicable).
c(i,j) ..............coefficient of the j-th regressor in the i-th equation of the var, or the coefficient of the j-th first-difference regressor in the i-th equation of the VEC.

Scalar Values (system level data)

@aic ................Akaike information criterion for the system.
@determinant ..........determinant of the residual covariance matrix.
@hq ..................Hannan-Quinn information criterion for the system.
@logl ................log likelihood for system.
@ncoefs ..........total number of estimated coefficients in the var.
@neqn .................number of equations.
@regobs ..........number of observations in the var.
@sc ...................Schwarz information criterion for the system.
@svarcvgtype ........Returns an integer indicating the convergence type of the structural decomposition estimation: 0 (convergence achieved), 2 (failure to improve), 3 (maximum iterations reached), 4 (no convergence—structural decomposition not estimated).
@svaroverid ........over-identification LR statistic from structural factorization.
@totalobs ..........sum of “@eqregobs” from each equation (“@regobs*@neqn”).

Vectors and Matrices

@coefmat ..........coefficient matrix (as displayed in output table).
@coefse .............matrix of coefficient standard errors (corresponding to the output table).
@cointse ..........standard errors of cointegrating vectors.
@cointvec ..........cointegrating vectors.
@impfact ..........factorization matrix used in last impulse response view.
@lrrsp ..........accumulated long-run responses from last impulse response view.
@lrrspse ..........standard errors of accumulated long-run responses.
@residcov .......... (sym) covariance matrix of the residuals.
@svaramat .......... estimated A matrix for structural factorization.
@svarbmat .......... estimated B matrix for structural factorization.
@svarcovab ........ covariance matrix of stacked A and B matrix for structural factorization.
@svarrcov .......... restricted residual covariance matrix from structural factorization.

**Var Examples**

To declare a var estimate a VEC specification and make a residual series:

```
var finec.ec(e,2) 1 6 cp div r
finec.makeresids
```

To estimate an ordinary var, to create series containing residuals, and to form a model based upon the estimated var:

```
var empvar.ls 1 4 payroll hhold gdp
empvar.makeresids payres hholdres gdpres
empvar.makemodel(inmdl) cp fcp div fdiv r fr
```

To save coefficients in a scalar:

```
scalar coef1=empvar.b(1,2)
```

**Vector**

Vector. (One dimensional array of numbers).

**Vector Declaration**

```
vector .................. declare vector object (p. 487).
```

There are several ways to create a vector object. Enter the `vector` keyword (with an optional dimension) followed by a name:

```
vector scalarmat
vector(10) results
```

Alternatively, you may declare a vector using an assignment statement. The vector will be sized and initialized, accordingly:

```
vector(10) myvec=3.14159
vector results=myvec
```

**Vector Views**

```
area .................... area graph of the vector (p. 191).
bar ...................... bar graph of data against the row index (p. 199).
```
Vector—175

label .................... label information for the vector object (p. 302).
line ...................... line graph of the data against the row index (p. 306).
sheet ..................... spreadsheet view of the vector (p. 432).
spike ..................... spike graph (p. 442).
stats ...................... descriptive statistics (p. 449).

Vector Procs

displayname .......... set display name (p. 251).
fill ..................... fill elements of the vector (p. 267).
read .................... import data from disk (p. 377).
setformat .......... set the display format for the vector spreadsheet (p. 419).
setindent .......... set the indentation for the vector spreadsheet (p. 424).
setjust ............. set the justification for the vector spreadsheet (p. 425).
setwidth ........... set the column width for the vector spreadsheet (p. 431).
write ................ export data to disk (p. 498).

Vector Data Members

(i) ..................... (i)-th element of the vector. Simply append “(i)” to the matrix name (without a “.”).

Vector Examples

To declare a vector and to fill it with data read in from an Excel file:

vector(10) mydata
mydata.read(b2) thedata.xls

To access a single element of the vector using direct indexing:

scalar result1=mydata(2)

The vector may be used in standard matrix expressions:

rowvector transdata=@transpose(mydata)
scalar inner=@transpose(mydata)*mydata
Appendix B. Command Reference

This chapter provides a complete alphabetical listing of the commands, views, and procedures in EViews.

Each entry is comprised of a keyword, followed by a listing of the ways in which the keyword may be used. The entry also provides a summary of behavior, as well as a description of the syntax. If appropriate, we also provide a listing of additional arguments, short examples, and cross-references.

For example, the listing for `correl` is given by:

```
Keyword: correl

Command: `correl(n, options) series_name`
          `correl(n, options) object_name.correl(n, options)`

View: `correl(n, options) series_name`
      `correl(n, options) object_name.correl(n, options)`

Syntax:
- `n` Specify the number of lags of the correlogram to display
- `p` Print the correlograms

Examples:
- `correl(24) m1`
- `m1.correl(24)`
  Both examples display the correlograms of the M1 series for up to 24 lags.

Cross-references:
See Chapter 7 of the User’s Guide for a discussion of autocorrelation and partial correlation functions.
```

Note that the keyword `correl` may be used in three distinct ways: as a command to compute the correlogram of a specified series, as a series view to display the correlogram for a series object, and as an equation view to display the correlogram of the residuals from the estimated equation. The syntax for each of these uses is documented separately.

In addition to the dictionary-style alphabetical listing provided here, there are several other places in the Command and Programming Reference that contain reference material:

- “Basic Command Summary” beginning on page 16, provides a listing of commands commonly used when managing objects in workfiles and databases.
In “Matrix Function and Command Summary” on page 44, we list the basic functions and commands used when working with matrix objects.

In “Programming Summary” on page 115, we list the commands, functions and keywords used in the EViews programming and string processing.

Appendix A, “Object, View and Procedure Reference”, beginning on page 135, lists all of the views and procedures classified by object, making it easy to see which views and procedures are available for a given object.

Appendix C, “Special Expression Reference”, beginning on page 519 lists special expressions that may be used in series assignment and generation, or as terms in estimation specifications.

Appendix D, “Matrix and String Reference”, beginning on page 527 documents the matrix, vector, scalar, and string functions and commands.

Appendix E, “Programming Language Reference”, on page 559 contains the primary documentation for the commands and keywords used in the programming language. These entries may only be used in batch programs.

Appendix F, “Operator and Function Reference”, on page 573 contains a complete listing of the functions and mathematical operators used in forming series expressions, and in performing matrix element operations.

<table>
<thead>
<tr>
<th>3sls</th>
<th>System Method</th>
</tr>
</thead>
</table>

Estimate a system of equations by three-stage least squares.

Syntax

System Method: system_name.3sls(options)

Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Iterate simultaneously over the weighting matrix and coefficient vector.</td>
</tr>
<tr>
<td>s</td>
<td>Iterate sequentially over the weighting matrix and coefficient vector.</td>
</tr>
<tr>
<td>o (default)</td>
<td>Iterate the coefficient vector to convergence following one-iteration of the weighting matrix.</td>
</tr>
<tr>
<td>c</td>
<td>One step (iteration) of the coefficient vector following one-iteration of the weighting matrix.</td>
</tr>
</tbody>
</table>
Examples

sys1.3sls(i)

Estimates SYS1 by the 3SLS method, iterating simultaneously on the weighting matrix and the coefficient vector.

nlsys.3sls(showopts,m=500)

Estimates NLSYS by 3SLS with up to 500 iterations. The “showopts” option displays the starting values and other estimation options.

Cross-references


add

Add series to a group or add cross section members to a pool.

Syntax

Group Proc:  group_name.add arg1 [arg2 arg3 ...]

Pool Proc:  pool_name.add id1 [id2 id3 ...]
List the names of series or group of series to add to the group, or list the cross-section identifiers to add to the pool.

Examples

    dummy.add d11 d12

Adds the two series D11 and D12 to the group DUMMY.

    countries.add us gr

Adds US and GR as cross-section members of the pool object COUNTRIES.

Cross-references
See also drop (p. 255).

<table>
<thead>
<tr>
<th>addassign</th>
<th>Model Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Assign add factors to equations.

Syntax

    Model Proc: model_name.addassign(options) equation_spec

where equation_spec identifies the equations for which you wish to assign add factors. You may either provide a list of endogenous variables, or you can use one of the following shorthand keywords:

| @all               | All equations. |
| @stochastic       | All stochastic equations (no identities). |
| @identity         | All identities. |

The options identify the type of add factor to be used, and control the assignment behavior for equations where you have previously assigned add factors. addassign may be called multiple times to add different types of add factors to different equations. addassign may also be called to remove existing add factors.

Options

<table>
<thead>
<tr>
<th>i</th>
<th>Intercept shifts (default).</th>
</tr>
</thead>
</table>
Examples

\[
m1.addassign(v) @all
\]
assigns a variable shift to all equations in the model while
\[
m1.addassign(c, i) @stochastic
\]
changes the stochastic equation add factors to intercept shifts.
\[
m1.addassign(v) @stochastic
m1.addassign(v) y1 y2 y2
m1.addassign(i) @identity
\]
assigns variable shifts to the stochastic equations and the equations for Y1, Y2, and Y3, and assigns intercept shifts to the identities.

Cross-references


See \texttt{addinit (p. 181)}.
The options control the type of initialization and the scenario for which you want to perform the initialization. `addinit` may be called multiple times to initialize various types of add factors in the different scenarios.

**Options**

<table>
<thead>
<tr>
<th>v = arg</th>
<th>Initialize add factors: “z” (set add factor values to zero), “n” (set add factor values so that the equation has no residual when evaluated at actuals), “b” (set add factors to the values of the baseline; override = actual).</th>
</tr>
</thead>
<tbody>
<tr>
<td>s = arg</td>
<td>Scenario selection: “a” (set active scenario add factors), “b” (set baseline scenario/actuals add factors), “o” (set active scenario override add factors).</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
m1.addinit(v=b) @all
```

sets all of the add factors in the active scenario to the values of the baseline.

```plaintext
m1.addinit(v=z) @stochastic
m1.addinit(v=n) y1 y1 y2
```

first sets the active scenario stochastic equation add factors to zero, and then sets the Y1, Y2, and Y3 equation residuals to zero (evaluated at actuals).

```plaintext
m1.addinit(s=b, v=z) @stochastic
```

sets the baseline scenario add factors to zero.

**Cross-references**

See “Using Add Factors” on page 775 of the User’s Guide. See also Chapter 26 of the User’s Guide for a general discussion of models.

See also `addassign` (p. 180).
addtext—Place text in graphs.

Syntax

Graph Proc: graph_name.addtext(options) text

Follow the addtext keyword with the text to be placed in the graph.

Options

- **font = n**: Set the size of the font.
- **t**: Top (above the graph and centered).
- **l**: Left rotated.
- **r**: Right rotated.
- **b**: Below and centered.
- **x**: Enclose text in box.

To place text within a graph, you can use explicit coordinates to specify the position of the upper left corner of the text.

Coordinates are set by a pair of numbers $h$, $v$ in virtual inches. Individual graphs are always $4 \times 3$ virtual inches (scatter diagrams are $3 \times 3$ virtual inches) or a user-specified size, regardless of their current display size.

The origin of the coordinate is the upper left hand corner of the graph. The first number $h$ specifies how many virtual inches to offset to the right from the origin. The second number $v$ specifies how many virtual inches to offset below the origin. The upper left hand corner of the text will be placed at the specified coordinate.

Coordinates may be used with other options, but they must be in the first two positions of the options list. Coordinates are overridden by other options that specify location.

When addtext is used with a multiple graph, the text is applied to the whole graph, not to each individual graph.

Examples

freeze(g1) gdp.line
Places the text “Fig 1: Monthly GDP (78.1-95.12)” centered above the graph G1.

Places the text “Seasonally Adjusted” in a box within the graph, slightly indented from the upper left corner.

Cross-references
See also `legend` (p. 304).

### align

<table>
<thead>
<tr>
<th>align</th>
<th>Graph Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Align placement of multiple graphs.</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Syntax**

Graph Proc: `graph_name.align(n,h,v)`

**Options**

You must specify three numbers (each separated by a comma) in parentheses in the following order: the first number `n` is the number of columns in which to place the graphs, the second number `h` is the horizontal space between graphs, and the third number `v` is the vertical space between graphs. Spacing is specified in virtual inches.

**Examples**

```plaintext
mygraph.align(3,1.5,1)
```

Aligns MYGRAPH with graphs placed in three columns, horizontal spacing 1.5 inches, and vertical spacing 1 inch.

```plaintext
var var1.ls 1 4 m1 gdp
freeze(impgra) var1.impulse(m,24) gdp @ gdp m1
impgra.align(2,1,1)
```

Estimates a VAR, freezes the impulse response functions as multiple graphs, and realigns the graphs. By default, the graphs are stacked in one column, and the realignment places the graphs in two columns.

**Cross-references**

For a detailed discussion of customizing graphs, see Chapter 14, “Graphs, Tables, and Text Objects”, on page 387 of the User’s Guide.
See also graph (p. 288).

<table>
<thead>
<tr>
<th>alpha</th>
<th>Object Declaration</th>
</tr>
</thead>
</table>

Declare an alpha series object.

The **alpha** command creates and optionally initializes an alpha series, or modifies an existing series.

**Syntax**

- **Command:** `alpha ser_name`
- **Command:** `alpha ser_name = formula`

The **alpha** command should be followed by either the name of a new series, or an explicit or implicit expression for generating a series. If you create a series and do not initialize it, the series will be filled with the blank string “”.

**Examples**

- `alpha x = "initial value"`
  
  creates a series named X filled with NAs.

Once an alpha is declared, you need not include the **alpha** keyword prior to entering the formula (alternatively, you may use **genr** (p. 281)). The following example generates an alpha series named VAL that takes value “Low” if either INC is less than or equal to 5000 or EDU is less than 13, and “High” otherwise.

- `alpha val
  
  val = @recode(inc<=5000 or edu<13, "High", "Low")`

If FIRST is an alpha series containing first names, and LAST is an alpha containing last names, then

- `alpha name = first + " " + last`

creates an alpha series containing the full names.

**Cross-references**

See “Alpha Series” on page 135 of the *User’s Guide* for additional discussion.

“Alpha” (p. 136) provides details on the alpha object. See also **genr** (p. 281).
Append a specification line to a model, system, sspace, valmap, or var.

Syntax

Object Proc: object_name.append text

Var Proc: var_name.append(options) text

Type the text to be added after the append keyword. For vars, you must specify the text type in the options argument.

Options for Vars

One of the following options is required when using append as a var proc:

- `svar` Text for identifying restrictions for structural VAR.
- `coint` Text for restrictions on the cointegration relations and/or adjustment coefficients.

Examples

```
model macro2
macro2.merge eq_ml
macro2.merge eq_gdp
macro2.append assign @all f
macro1.append @trace gdp
macro2.solve
show macro1.results
```

The first line declares a model object. The second and third lines merge existing equations into the model. The fourth and fifth line appends an assign statement and a trace of GDP to the model. The last line solves the model.

```
macro1.append cons=c(1)+c(2)*gdp+c(3)*cons(-1)
macro1.append inv=c(4)+c(5)*tb3+c(6)*d(gdp)
macro1.append gdp=cons+inv+gov
macro1.append inst tb3 gov cons(-1) gdp(-1)
macro1.gmm
show macro1.results
```
The first line declares a system. The next three lines append the specification of each endogenous variable in the system. The fifth line appends the list of instruments to be used in estimation. The last two lines estimate the model by GMM and display the estimation results.

```plaintext
vector(2) svec0=0
sspace1.append @mprior svec0
```

This command appends a line in the state space object SSPACE1 to use the zero vector SVEC0 as initial values for the state vector.

**Cross-references**

See also cleartext (p. 221). See “System Estimation Methods” on page 666, and “Models” on page 749, and “Value Maps” on page 145 of the *User’s Guide* for details.

---

**arch**

<table>
<thead>
<tr>
<th>Command</th>
<th>Equation Method</th>
</tr>
</thead>
</table>

Estimate generalized autoregressive conditional heteroskedasticity (GARCH) models.

**Syntax**

- Command: `arch(p,q,options) y [x1 x2 x3] [@ p1 p2 [@ t1 t2]]`
- Command: `arch(p,q,options) y = expression [@ p1 p2 [@ t1 t2]]`
- Equation Method: `eq_name.arch(p,q,options) y [x1 x2 x3] [@ p1 p2 [@ t1 t2]]`
- Equation Method: `eq_name.arch(p,q,options) y = expression [@ p1 p2 [@ t1 t2]]`

The ARCH command or method estimates a model with \( p \) ARCH terms and \( q \) GARCH terms. Note the order of the arguments in which the ARCH and GARCH terms are entered, which gives precedence to the ARCH term. The maximum value for \( p \) or \( q \) is 9; values above will be set to 9. The minimum value for \( p \) is 1; the minimum value for \( q \) is 0. If either \( p \) or \( q \) is not specified, EViews will assume a corresponding order of 1. Thus, a GARCH(1, 1) is assumed by default.

After the “ARCH” keyword, specify the dependent variable followed by a list of regressors in the mean equation.

By default, no exogenous variables (except for the intercept) are included in the conditional variance equation. If you wish to include variance regressors in addition to the GARCH terms, list them after the mean equation using an “@”-sign to separate the two lists.

When estimating component ARCH models, you may specify exogenous variance regressors for the permanent and transitory components separately. After the mean equation
regressors, first list the regressors for the permanent component, followed by an “@”-sign, then the regressors for the transitory component. A constant term is always included as a permanent component.

Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>egarch</td>
<td>Exponential GARCH.</td>
</tr>
<tr>
<td>parch[ = arg]</td>
<td>Power ARCH. If the optional arg is provided then the power parameter will be set to that value, otherwise it will be estimated.</td>
</tr>
<tr>
<td>cgarch</td>
<td>Component (permanent and transitory) ARCH.</td>
</tr>
<tr>
<td>asy = integer (default = 1)</td>
<td>Number of asymmetric terms in the Power ARCH or EGARCH model. The maximum number of terms allowed is 9.</td>
</tr>
<tr>
<td>thrsh = integer (default = 0)</td>
<td>Number of threshold terms for GARCH and Component models. The maximum number of terms allowed is 9. For Component model “thrsh” must take a value of 0 or 1.</td>
</tr>
<tr>
<td>mean = arg</td>
<td>ARCH-M (ARCH in mean) specification with the conditional standard deviation (“mean = sd”), conditional variance (“mean = var”) or log of the conditional variance (“mean = log”) entered as a regressor in the mean equation.</td>
</tr>
<tr>
<td>tdist [= number]</td>
<td>Estimate the model assuming that the residuals follow a conditional Student’s t-distribution (the default is the conditional normal distribution). Providing the optional number greater than two will fix the degrees-of-freedom. If the argument is not provided, the degrees-of-freedom will be estimated.</td>
</tr>
<tr>
<td>ged [= number]</td>
<td>Estimate the model assuming that the residuals follow a conditional GED (the default is the conditional normal distribution). Providing a positive value for the optional argument will fix the GED parameter. If the argument is not provided, the parameter will be estimated.</td>
</tr>
<tr>
<td>h</td>
<td>Bollerslev-Wooldridge robust quasi-maximum likelihood (QML) covariance/standard errors.</td>
</tr>
</tbody>
</table>
Saved results

Most of the results saved for the `ls` command are also available after ARCH estimation; see `ls (p. 315)` for details.

Examples

The command

```plaintext
arch(4, 0, m=1000, h) sp500 c
```

estimates an ARCH(4) model with a mean equation consisting of SP500 regressed on a constant. The procedure will perform up to 1000 iterations and, upon convergence, will report Bollerslev-Wooldridge robust QML standard errors.

The commands

```plaintext
c = 0.1

equation arcl.arch(thrsh=1, s, mean=var) @pch(nys) c ar(1)
```

estimates a TARCH(1, 1)-in-mean specification with the mean equation the percent change of NYS depending upon a constant, AR order 1, and a conditional variance (GARCH) term.
The first line sets the default coefficient vector to 0.1 and the “s” option uses these values as coefficient starting values.

The command:

\[
\text{arch}(1, 2, \text{asy}=0, \text{parch}=1.5, \text{ged}=1.2) \, \text{dlog}(ibm)=c(1)+c(2)\times\text{dlog}(sp500) @ r
\]

estimates a symmetric Power ARCH(2, 1) (autoregressive GARCH of order 2, and moving average ARCH of order 1) model with GED errors. The power of model is fixed at 1.5 and the GED parameter is fixed at 1.2. The mean equation is the first log difference of IBM regressed on a constant and the first log difference of SP500. The conditional variance equation includes an exogenous regressor R.

Following estimation, we can save the estimated conditional variance as a series named GARCH1.

\[
\text{arch1.makegarch garch1}
\]

Cross-references

See Chapter 20 of the *User's Guide* for a discussion of ARCH models. See also *garch* (p. 280) and *makegarch* (p. 322).

```plaintext
archtest Command || Equation View

Test for autoregressive conditional heteroskedasticity (ARCH).
Carries out Lagrange Multiplier (LM) tests for ARCH in the residuals.

Syntax

Command: archtest(options)
Equation View: eq_name.archtest(options)

Options

You must specify the order of ARCH to test for. The number of lags to include should be provided in parentheses after the arch keyword.

*Other Options:*

\[
p\quad \text{Print output from the test.}
\]

Examples

```
ls output c labor capital
```
archtest(4)

Regresses OUTPUT on a constant, LABOR, and CAPITAL and tests for ARCH up to order 4.

    equation eql.arch sp500 c
eql.archtest(4)

Estimates a GARCH(1,1) model with mean equation of SP500 on a constant and tests for additional ARCH up to order 4. Note that performing an archtest after an arch estimation uses the standardized residuals (the residual of the mean equation divided by the estimated conditional standard deviation).

Cross-references


| area   | Command || Coef View | Graph Command | Group View | Matrix View | Rowvector View | Series View | Sym View | Vector View |
|--------|----------|-------------|---------------|-------------|-------------|---------------|--------------|-----------|------------|

Display area graph of object, or change existing graph object type to area graph.

Create area or filled line graph from one or more series, or from each column of a matrix object.

Syntax

Command: area(options) arg1 [arg2 arg3 ...]
Object View: object_name.area(options)
Graph Proc: graph_name.area(options)

Options

*Template and printing options*

- o = graph_name: Use appearance options from the specified graph.
- t = graph_name: Use appearance options and copy text and shading from the specified graph.
- p: Print the area graph.
**Scale options**

- **a (default)**: Automatic single scale.
- **d**: Dual scaling with no crossing. The first series is scaled on the left and all other series are scaled on the right.
- **x**: Dual scaling with possible crossing. See the “d” option.
- **n**: Normalized scale (zero mean and unit standard deviation). May not be used with the “s” option.
- **s**: Stacked area graph. Each area represents the cumulative total of the series listed. The difference between areas corresponds to the value of a series. May not be used with the “l” option.
- **l**: Area graph for the first series listed and a line graph for all subsequent series. May not be used with the “s” option.
- **m**: Plot areas in multiple graphs (will override the “s” or “l” options). Not for use with an existing graph object.

**Panel options**

The following options apply when graphing panel structured data.

- **panel = arg**: Panel data display: “stack” (stack the cross-sections), “individual” or “1” (separate graph for each cross-section), “combine” or “c” (combine each cross-section in single graph; one time axis), “mean” (plot means across cross-sections), “mean1se” (plot mean and +/- 1 standard deviation summaries), “mean2sd” (plot mean and +/- 2 s.d. summaries), “mean3sd” (plot mean and +/- 3 s.d. summaries), “median” (plot median across cross-sections), “med25” (plot median and +/- .25 quantiles), “med10” (plot median and +/- .10 quantiles), “med05” (plot median and +/- .05 quantiles), “med025” (plot median and +/- .025 quantiles), “med005” (plot median and +/- .005 quantiles), “medmxmn” (plot median, max and min).
Examples

```plaintext
group g1 ser1 ser2 ser3
  g1.area(s)
```

defines a group G1 containing the three series SER1, SER2 and SER3, then plots a stacked area graph of the series in the group.

```plaintext
area(l, o=gra1) s1 gdp cons
```

creates an area graph of series S1, together with line graphs of GDP and CONS. The graph uses options from graph GRA1 as a template.

Cross-references

The graph object is described in greater detail in “Graph” (p. 143). See Chapter 14 of the User’s Guide for a detailed discussion of graphs in EViews, and “Graph Templates” on page 394 of the User’s Guide for a discussion of graph templates.

See also graph (p. 288) for graph declaration and other graph types.

```
test
```

Perform multivariate residual serial correlation LM test using an estimated Var.

**Syntax**

```plaintext
Var View: var_name.arlm(h, options)
```

You must specify the highest order of lag h to test for serial correlation.

**Options**

- `name=arg` : Save LM statistics in named matrix object. The matrix has h rows and one column.
- `p` : Print test output.

**Examples**

```plaintext
var var1.ls 1 6 lgdp l1m1 lcpi
  show var1.arlm(12, name=lmout)
```

The first line declares and estimates a VAR with 6 lags. The second line displays the serial correlation LM tests for lags up to 12 and stores the statistics in a matrix named LMOUT.
Cross-references

See “Diagnostic Views” on page 694 of the User’s Guide for other VAR diagnostics.

See also qstats (p. 376) for related multivariate residual autocorrelation portmanteau tests.

<table>
<thead>
<tr>
<th>arma</th>
<th>Equation View</th>
</tr>
</thead>
</table>

Examine ARMA structure of estimated equation.

Provides diagnostic graphical and tabular views that help you assess the structure of the ARMA part of an estimated equation. The view is currently available only for equations specified by list that include at least one AR or MA term that are estimated by least squares. There are three views available: roots, correlogram, and impulse response.

Syntax

Object View: `eq_name arma(type = arg [,options])`

where `eq_name` is the name of an equation object specified by list, estimated by least squares, and contains at least one ARMA term. The “type =” and other options should be specified inside the parentheses.

Options

- **type = arg** Required “type =” option selects the type of ARMA structure output: “root” displays the inverse roots of the AR/MA characteristic polynomials, “acf” displays the second moments (autocorrelation and partial autocorrelation) for the data in the estimation sample and for the estimated model, “imp” displays the impulse responses.
- **t** Displays the table view of the results for the view specified by the “type =” option. By default, EViews will display a graphical view of the ARMA results.
- **hrz = arg** Specifies the maximum lag length for “type = acf” and the maximum horizon (periods) for “type = imp”.
- **imp = arg** Specifies the size of the impulse for the impulse response (“type = imp”) view. By default, EViews will use the regression estimated standard error.
arroots—195

| save = arg | Stores the results as a matrix object with the specified name. The matrix holds the results roughly as displayed in the table view of the corresponding type. For “type = root”, roots for the AR and MA polynomials will be stored in separate matrices as NAME_AR and NAME_MA where “NAME” is the name given by the “save = ” option. |

p | Print the table or graph output. |

Examples

eq1.arma(type=root, save=root)
displays and saves the ARMA roots from the estimated equation EQ1. The roots will be placed in the matrix object ROOT.

eq1.arma(type=acf, hrz=25, save=acf)
computes the second moments (autocorrelation and partical autocorrelations) for the observations in the sample and for the estimated model, for 25 periods. We save the results in the matrix object ACF.

eq1.arma(type=imp, hrz=25, save=imp)
computes the 25 period impulse-response function implied by the estimated ARMA coefficients. EViews will use the default 1 standard error of the estimated equation as the shock, and will save the results in the matrix object IMP.

Cross-references
See “ARMA Structure” on page 480 of the User’s Guide.

arroots Var View

Inverse roots of the characteristic AR polynomial.

Syntax

Var View: var_name.arroots(options)
Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name = arg</td>
<td>Save roots in named matrix object. The matrix has two columns, where the first column is the real and the second column is the imaginary part of each root in the rows.</td>
</tr>
<tr>
<td>graph</td>
<td>Plots the roots together with a unit circle. The VAR is stable if all roots are inside the unit circle.</td>
</tr>
<tr>
<td>p</td>
<td>Print table of AR roots.</td>
</tr>
</tbody>
</table>

Examples

```bash
var var1.ls 1 6 lgdp lml lcpi
var1.arroots(graph)
```

The first line declares and estimates a VAR with 6 lags. The second line plots the AR roots of the estimated VAR.

```bash
var var1.ls 1 6 lgdp lml lcpi
'store roots
freeze(tab1) var1.arroots(name=roots)
```

The first line declares and estimates a VAR with 6 lags. The second line stores the roots in a matrix named ROOTS and the table view as a table named TAB1.

Cross-references

See “Diagnostic Views” on page 694 of the User’s Guide for other VAR diagnostics.

---

**auto**

Command || Equation View

**auto**

Compute serial correlation LM (Lagrange multiplier) test.

Carries out Breusch-Godfrey Lagrange Multiplier (LM) tests for serial correlation in the estimation residuals.

Syntax

- Command: **auto**(options)
- Equation View: **eq_name.auto**(options)

In command form, **auto** tests the residuals from the default equation.
Options
You must specify the order of serial correlation to test for. You should specify the number of lags in parentheses after the `auto` keyword, followed by any additional options.

Other Options:

| p | Print output from the test. |

Examples
To regress OUTPUT on a constant, LABOR, and CAPITAL and test for serial correlation of up to order four:

```
ls output c labor capital
auto(4)
```

The commands

```
output(t) c:esult\artest.txt
equation eql.ls cons c y y(-1)
eql.auto(12, p)
```

perform a regression of CONS on a constant, Y and lagged Y and test for serial correlation of up to order twelve. The first line redirects printed tables/text to the ARTEST.TXT file.

Cross-references

<table>
<thead>
<tr>
<th>axis</th>
<th>Graph Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Graph Proc</td>
</tr>
</tbody>
</table>

Sets axis display characteristics for the graph.

Syntax

Graph Proc: `graph_name.axis(axis_id) options_list`

The `axis_id` parameter identifies which of the axes the proc modifies. If no option is specified, the proc will modify all of the axes. `axis_id` may take on one of the following values:

| left / l | Left vertical axis. |
| right / r | Right vertical axis. |
| bottom / b | Bottom axis. |
The options list may include any of the following options:

- **top / t** Top axis.
- **all / a** All axes.
- **zeroline / -zeroline** Draw/do not draw the horizontal line at zero.

### Options

Options list may include any of the following options:

- **grid / -grid** [Draw / Do not draw] grid lines.
- **zeroline / -zeroline** Draw a line at zero on the data scale.
- **ticksout** Draw tickmarks outside the graph axes.
- **ticksin** Draw tickmarks inside the graph axes.
- **ticksboth** Draw tickmarks both outside and inside the graph axes.
- **ticksnone** Do not draw tickmarks.
- **minor / -minor** [Allow / Do not allow] minor tick marks.
- **label / -label** [Place / Do not place] labels on the axes.
- **font(arg)** Set font size of labels, where `arg` is a numeric font size.
- **mirror / -mirror** [Label / Do not label] both left and right axes with duplicate axes (single scale graphs only).

Note that the default settings are taken from the global defaults.

### Examples

```
graph1.axis(r) zeroline -minor font(12)
```

draws a horizontal line through the graph at zero on the right axis, removes minor ticks, and changes the font size of the right axis labels to 12 point.

```
graph2.axis -mirror
```

turns off mirroring of axes in single scale graphs.

### Cross-references

See also scale (p. 397), datelabel (p. 237), options (p. 343) and setelem (p. 412).

Display bar graph of object, or change existing graph object type to bar graph.

Create bar graph from one or more series, or from each column of a matrix object.

Note: when the individual bars in a bar graph become too thin, the graph will automatically be converted into an area graph (area (p. 191)).

Syntax
Command: bar(options) arg1 [arg2 arg3 ...]
Object View: object_name.bar(options)
Graph Proc: graph_name.bar(options)

When used as a graph proc, bar changes the graph type to a bar graph.

Options

Template and printing options

<table>
<thead>
<tr>
<th>o = graph_name</th>
<th>Use appearance options from the specified graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>t = graph_name</td>
<td>Use appearance options and copy text and shading from the specified graph.</td>
</tr>
<tr>
<td>p</td>
<td>Print the bar graph.</td>
</tr>
</tbody>
</table>

Scale options

<table>
<thead>
<tr>
<th>a (default)</th>
<th>Automatic single scale.</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>Dual scaling with no crossing. The first series is scaled on the left and all other series are scaled on the right.</td>
</tr>
<tr>
<td>x</td>
<td>Dual scaling with possible crossing. See the “d” option.</td>
</tr>
<tr>
<td>n</td>
<td>Normalized scale (zero mean and unit standard deviation). May not be used with the “s” option.</td>
</tr>
</tbody>
</table>
Panel options

The following options apply when graphing panel structured data.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>Stacked bar graph. Each bar represents the cumulative total of the series listed. The difference between bars corresponds to the value of a series. May not be used with the “l” option.</td>
</tr>
<tr>
<td>l</td>
<td>Bar graph for the first series listed and a line graph for all subsequent series. May not be used with the “s” option.</td>
</tr>
<tr>
<td>m</td>
<td>Plot bars in multiple graphs. Will override the “s” or the “l” options. Not for use with an existing graph object.</td>
</tr>
</tbody>
</table>

Examples

Plot a bar graph of POP together with line graphs of GDP and CONS:

```
bar(l, x,o=mybar1) pop gdp cons
```

The bar graph is scaled on the left, while the line graphs are scaled on the right. The graph uses options from graph MYBAR1 as a template.

```
group mygrp oldsales newsales
mygrp.bar(s)
```
The first line defines a group of series and the second line graphs a stacked bar graph view of the series in the group.

Cross-references

See “Graph Templates” on page 394 of the User’s Guide for a discussion of graph templates.

See graph (p. 288) for graph declaration and additional graph types.

bdtest | Series View

Perform BDS test for independence.

The BDS test is a Portmanteau test for time based dependence in a series. It can be used for testing against a variety of possible deviations from independence including linear dependence, non-linear dependence, or chaos.

Syntax

Series View: series_name.bds(options)

Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>Value for calculating $\varepsilon$.</td>
</tr>
<tr>
<td>d</td>
<td>Maximum dimension.</td>
</tr>
<tr>
<td>b</td>
<td>Number of repetitions for bootstrap $p$-values. If option is omitted, no bootstrapping is performed.</td>
</tr>
<tr>
<td>o = arg</td>
<td>Name of output vector for final BDS $z$-statistics.</td>
</tr>
<tr>
<td>p</td>
<td>Print output.</td>
</tr>
</tbody>
</table>

Cross-references

See “BDS Test” on page 305 of the User’s Guide for additional discussion.
Estimate binary dependent variable models.

Estimates models where the binary dependent variable Y is either zero or one (probit, logit, gompit).

Syntax

Command: `binary(options) y x1 x2 x3`
Equation Method: `eq_name.binary(options) y x1 x2 x3`

Options

- `d = arg` (default = "n")  Specify likelihood: “n” (normal likelihood function, probit), “l” (logistic likelihood function, logit), “x” (Type I extreme value likelihood function, Gompit).
- `q (default)` Use quadratic hill climbing for maximization algorithm.
- `r` Use Newton-Raphson for maximization algorithm.
- `b` Use Berndt-Hall-Hall-Hausman (BHHH) for maximization algorithm.
- `h` Quasi-maximum likelihood (QML) standard errors.
- `g` GLM standard errors.
- `m = integer` Set maximum number of iterations.
- `c = scalar` Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.
- `s` Use the current coefficient values in C as starting values.
- `s = number` Specify a number between zero and one to determine starting values as a fraction of EViews default values (out of range values are set to “s = 1”).
- `showopts / -showopts` [Do / do not] display the starting coefficient values and estimation options in the estimation output.
- `p` Print results.
Examples
To estimate a logit model of Y on a constant, WAGE, EDU, and KIDS with QML standard errors:

\[ \text{binary(d=l,h) } y \ c \ \text{wage edu kids} \]

This command uses the default quadratic hill climbing algorithm. The commands

\[ \text{param c(1) .1 c(2) .1 c(3) .1} \]
\[ \text{equation probit1.binary(s) } y \ c \ x2 \ x3 \]

estimate a probit model of Y on a constant, X2, and X3, using the specified starting values. The commands

\[ \text{coef beta_probit = probit1.@coefs} \]
\[ \text{matrix cov_probit = probit1.@coefcov} \]

store the estimated coefficients and coefficient covariances in the coefficient vector BETA_PROBIT, and matrix COV_PROBIT.

Cross-references

block—203

block

Model View

Display the model block structure view.

Show the block structure of the model identifying which blocks are recursive and which blocks are simultaneous.

Syntax

Model View: \text{model_name.block(options)}

Options

p Print the block structure view.

Cross-references

See also \texttt{eqs} (p. 260), \texttt{text} (p. 469) and \texttt{vars} (p. 488) for alternative representations of the model.
**boxplot**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Group View: group_name.boxplot(options)</th>
</tr>
</thead>
</table>

Display boxplot of each series in the group.

Create a boxplot graph view containing boxplots for each series in the group.

**Syntax**

Follow the group name with a period, the keyword, and any options. The default settings are to display fixed width boxplots for each series using individual samples, with all basic elements drawn (mean, med, staples, whiskers, near outliers, far outliers), and with shading representing the approximate confidence intervals for the median.

**Options**

*Options to control initial display*

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nomean</td>
<td>Do not display means.</td>
</tr>
<tr>
<td>nomed</td>
<td>Do not display medians.</td>
</tr>
<tr>
<td>nostaple</td>
<td>Do not display staples.</td>
</tr>
<tr>
<td>nowhisk</td>
<td>Do not display whiskers.</td>
</tr>
<tr>
<td>nonearout</td>
<td>Do not display near outliers.</td>
</tr>
<tr>
<td>nofarout</td>
<td>Do not display far outliers.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>width=arg</td>
<td>Boxplot width: “fixed” (fixed width boxplots), “n” (width proportional to number of observations), “rootn” (width proportional to square root of number of observations).</td>
</tr>
<tr>
<td>ci=arg</td>
<td>95% confidence interval for median: “none” (do not display), “shade” (display interval as shaded area), “notch” (display interval using notched boxes).</td>
</tr>
</tbody>
</table>

*Options to control calculation of boxplots*

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>Balance sample.</td>
</tr>
<tr>
<td>q=arg</td>
<td>Compute quantiles using the specified definition: “b” (Blom), “r” (Rankit-Cleveland), “o” (simple fraction), “t” (Tukey), “v” (van der Waerden).</td>
</tr>
<tr>
<td>(default</td>
<td></td>
</tr>
</tbody>
</table>
|         |                                  |}
**boxplotby**

Display boxplot of a series classified into categories.

Create a boxplot graph view containing boxplots of the elements of a series classified into categories by one or more series.

**Syntax**

```
Series View:  series_name.boxplotby(options) classifier_names
```

Follow the series name with a period, the keyword, and a name (or a list of names) for the series or group by which to classify. The default settings are to display fixed width boxplots for each category, with all basic elements drawn (mean, med, staples, whiskers, near outliers, far outliers), and with shading representing the approximate confidence intervals for the median.

**Options**

*Options to control display*

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nomean</td>
<td>Do not display means.</td>
</tr>
<tr>
<td>nomed</td>
<td>Do not display medians.</td>
</tr>
<tr>
<td>nostaple</td>
<td>Do not display staples.</td>
</tr>
</tbody>
</table>
Options to control calculation of boxplots

- **nowhisk**: Do not display whiskers.
- **nonearout**: Do not display near outliers.
- **nofarout**: Do not display far outliers.
- **nolabel**: Do not display axis labels.

**width = arg**

- **(default = “fixed”)**: Boxplot width: “fixed” (fixed width boxplots), “n” (width proportional to number of observations), “rootn” (width proportional to square root of number of observations).

**ci = arg**

- **(default = “shade”)**: 95% confidence interval for median: “none” (do not display), “shade” (display interval as shaded area), “notch” (display interval using notched boxes).

**dropna (default), keepna**

[Drop/Keep] NA as a category.

**total**

Create category for entire series.

**q = arg**


Options to control binning

**v = integer**

- **(default = 100)**: Bin categories if classification series take on more than the specified number of distinct values.

**nov**

Do not bin based on the number of values of the classification series.

**a = number**

- **(default = 2)**: Bin categories if average cell count is less than the specified number.

**noa**

Do not bin based on the average cell count.

**b = integer**

- **(default = 5)**: Set maximum number of binned categories.

Other options

**p**

Print the view.
Examples

- `wage.boxplotby sex race` displays boxplots for the series WAGE by (possibly binned) values of SEX and RACE.

- `income.boxplotby(total, keepna, width=n) sex race` displays boxplots for INCOME classified by SEX and RACE, with missing values in the classifier series treated as categories, and an additional boxplot drawn for the entire series INCOME. The boxes will be drawn with variable widths proportional to the number of observations in each category.

Cross-references

See “Boxplots” on page 382 of the User’s Guide.

See also `boxplot` (p. 204), and `statby` (p. 444). For customization of a boxplot graph object, see `setelem` (p. 412) and `setbpelem` (p. 407).

### bpf

**Series Proc**

Compute and display the band-pass filter of a series.

Computes, and displays a graphical view of the Baxter-King fixed length symmetric, Christiano-Fitzgerald fixed length symmetric, or the Christiano-Fitzgerald full sample asymmetric band-pass filter of the series.

The view will show the original series, the cyclical component, and non-cyclical component in a single graph. For non time-varying filters, a second graph will show the frequency responses.

**Syntax**

```
series Proc: series_name.bpf(options) [cyc_name]
```

Follow the bpf keyword with any desired options, and the optional name to be given to the cyclical component. If you do not provide `cyc_name`, the filtered series will be named BPFILTER## where ## is a number chosen to ensure that the name is unique.
Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>**type = arg</td>
<td>Specify the type of band-pass filter: “bk” is the Baxter-King fixed length symmetric filter, “cffix” is the Christiano-Fitzgerald fixed length symmetric filter, “cfasym” is the Christiano-Fitzgerald full sample asymmetric filter.</td>
</tr>
<tr>
<td>(default = “bk”)</td>
<td></td>
</tr>
<tr>
<td><strong>low = number,</strong></td>
<td>Low ( (PL) ) and high ( (PH) ) values for the cycle range to be passed through (specified in periods of the workfile frequency).</td>
</tr>
<tr>
<td><strong>high = number</strong></td>
<td>Defaults to the workfile equivalent corresponding to a range of 1.5–8 years for semi-annual to daily workfiles; otherwise sets “low = 2”, “high = 8”.</td>
</tr>
<tr>
<td></td>
<td>The arguments must satisfy ( 2 \leq PL &lt; PH ). The corresponding frequency range to be passed through will be ( (2\pi / PH, 2\pi / PL) ).</td>
</tr>
<tr>
<td><strong>lag = integer</strong></td>
<td>Fixed lag length (positive integer). Sets the fixed lead/lag length for fixed length filters (“type = bk” or “type = cffix”). Must be less than half the sample size.</td>
</tr>
<tr>
<td></td>
<td>Defaults to the workfile equivalent of 3 years for semi-annual to daily workfiles; otherwise sets “lag = 3”.</td>
</tr>
<tr>
<td><strong>iorder = [0,1]</strong></td>
<td>Specifies the integration order of the series. The default value, “0” implies that the series is assumed to be (covariance) stationary; “1” implies that the series contains a unit root.</td>
</tr>
<tr>
<td>(default = 0)</td>
<td>The integration order is only used in the computation of Christiano-Fitzgerald filter weights (“type = cffix” or “type = cfasym”). When “iorder = 1”, the filter weights are constrained to sum to zero.</td>
</tr>
<tr>
<td><strong>detrend = arg</strong></td>
<td>Detrending method for Christiano-Fitzgerald filters (“type = cffix” or “type = cfasym”). You may select the default argument “n” for no detrending, “c” to demean, or “t” to remove a constant and linear trend.</td>
</tr>
<tr>
<td>(default = “n”)</td>
<td>You may use the argument “d” to remove drift, if the option “iorder = 1” is also specified.</td>
</tr>
</tbody>
</table>
Store the filter weights as an object with the specified name. For fixed length symmetric filters ("type=bk" or "type=cffix"), the saved object will be time-invariant (fixed-length symmetric) filters, the weight matrix is of dimension \( 1 \times (q + 1) \) where \( q \) is the user-specified lag length order. For these filters, the weights on the leads and the lags are the same, so the returned matrix contains only the one-sided weights. The filtered series can be computed as

\[
z_t = \sum_{c=1}^{q+1} w(1, c) y_{t+1-c} + \sum_{c=2}^{q+1} w(1, c) y_{t+c-1}
\]

for \( t = q + 1, \ldots, n - q \).

For time-varying filters, the weight matrix is of dimension \( n \times n \) where \( n \) is the number of non-missing observations in the current sample. Row \( r \) of the matrix contains the weighting vector used to generate the \( r \)-th observation of the filtered series where column \( c \) contains the weight on the \( c \)-th observation of the original series.

\[
z_t = \sum_{c=1}^{n} w(t, c) y_c \quad t = 1, \ldots, n
\]

where \( z_t \) is the filtered series, \( y_t \) is the original series and \( w(r, c) \) is the \((r, c)\) element of the weighting matrix. By construction, the first and last rows of the weight matrix will be filled with missing values for the symmetric filter.

Suppresses plotting of the frequency response (gain) function for fixed length symmetric filters ("type=bk" or "type=cffix"). By default, EViews will plot the gain function.

Specifies a name for a series to contain the non-cyclical series (difference between the actual and the filtered series). If no name is provided, the non-cyclical series will not be saved in the workfile.

Print the graph.
Examples

Suppose we are working in a quarterly workfile and we issue the following command:

\[ \text{lgdp.bpf(type=bk,low=6,high=32) cyc0} \]

EViews will compute the Baxter-King band-pass filter of the series LGDP. The periodicity of cycles extracted ranges from 6 to 32 quarters, and the filtered series will be saved in the workfile in CYC0. The BK filter uses the default lag of 12 (3 years of quarterly data).

Since this is a fixed length filter, EViews will display both a graph of the cyclical/original/non-cyclical series, as well as the frequency response (gain) graph. To suppress the latter graph, we could enter a command containing the “nogain” option:

\[ \text{lgdp.bpf(type=bk,low=6,high=32,lag=12,nogain)} \]

In this example, we have also overridden the default by specifying a fixed lag of 12 (quarters). Since we have omitted the name for the cyclical series, EViews will create a series with a name like BPFILTER01 to hold the results.

To compute the asymmetric Christiano-Fitzgerald filter, we might enter a command of the form:

\[ \text{lgdp.bpf(type=cfasym,low=6,high=32,noncyc=non1,weight=wm) cyc0} \]

The cyclical components are saved in CYC0, the non-cyclical in NON1, and the weighting matrix in WM.

Cross-references

See “Frequency (Band-Pass) Filter” on page 333. See also hpf (p. 295).

### bplabel

<table>
<thead>
<tr>
<th>Graph Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bplabel</strong></td>
</tr>
</tbody>
</table>

Specify labeling of bottom-axis in boxplots.

Sets options that are specific to the appearance of the bottom axis of boxplots.

Additional options that affect the appearance of the axis may be set using **axis** (p. 197) using the “bottom option”. These options include tick control, label and font options, and grid lines.

**Syntax**

Graph Proc: graph_name.bplabel option_list

The **option_list** may contain one or more of the following:
interval(step_size[, steps[, align_id]]
(default step_size = "auto")

Specify the label frequency: step_size can take on one of the following values: “auto” (automatic), “ends” (only label endpoints; first and last boxplot), “cust” (use align_id and steps to determine which labels to display).

steps is an optional number indicating the number of steps between labels (a single step is defined as one box). The default value is automatically determined.

align_id is an optional integer representing a box number to be used as a base. The base box will always receive a label, and subsequent labeling will be determined using the specified steps. The default value is 1.

Note: the align_index may lie above the current number of boxes.

angle(arg)

Rotates the text label to the specified angle, where arg can be either a number, measured from -90 to 90 degrees, or “auto” (or “a”) for automatically determined angling. The angle is measured in 15 degree increments, counterclockwise from the horizontal axis.

label(id[, "text"])/ -label(id[, "text"])

Show or hide the label of the id box. text is a new label for the box, and should be enclosed in quotes.

You may optionally precede the “label” keyword with “+”.

Examples

\[\text{graph01.bplabel interval(ends) angle(45)}\]

will display the endpoint labels only at a 45 degree angle.

\[\text{graph01.bplabel interval(cust, 10, 2) label(2, "North" )}\]

displays labels on every tenth box, centered around the second. The label for the second box is also changed to “North”.

Cross-references

See `setelem` (p. 412) to modify line and symbol attributes. See also `setbpelem` (p. 407), `options` (p. 343), `axis` (p. 197), and `scale` (p. 397).

| cause | Command || Group View |
|-------|----------||-------------|

**Granger causality test.**

Performs pairwise Granger causality tests between (all possible) pairs of the listed series or group of series.

**Syntax**

Command: `cause(n, options) ser1 ser2 ser3`

Group View: `group_name.cause(n, options)`

In command form, you should list the series or group of series in which to test Granger causality.

**Options**

You must specify the number of lags $n$ to use for the test; specify an integer in parentheses after the `cause` keyword. Note that the regressors of the test equation are a constant and the specified lags of the pair of series under test.

*Other options:*

- `p` Print output of the test.

**Examples**

Compute Granger causality tests of whether GDP Granger causes M1 and whether M1 Granger causes GDP.

`cause(4) gdp m1`

The regressors of each test are a constant and four lags of GDP and M1. The commands

```plaintext
  group macro m1 gdp r
  macro.cause(12,p)
```

print the result of six pairwise Granger causality tests for the three series in the MACRO group. The regressors of each test are a constant and twelve lags of the two series under test (and do not include lagged values of the third series in the group).
Cross-references

See “Granger Causality” on page 363 of the User’s Guide for a discussion of Granger’s approach to testing hypotheses about causality.

See also var (p. 487).

**ccopy**

**Command**

Copy series from the DRI Basic Economics database to the data bank.

Copies one or more series from the DRI Basic Economics Database to the EViews data bank (.DB) files. *You must have the DRI database installed on your computer to use this feature.*

**Syntax**

Command: `ccopy series_name`

Type the name of the series you want to copy after the `ccopy` keyword. The data bank file will be stored in the default directory with the same name as the series name in the DRI database. You can supply path information to indicate the directory for the data bank file.

**Examples**

The command

```
ccopy lhur
```

copies the DRI series LHUR to LHUR.DB file in the default path directory, while

```
ccopy b:gdp c:\nipadata\gnet
```

copies the GDP series to GDP.DB file in the B drive and the GNET series to the GNET.DB file in C:\NIPADATA.

**Cross-references**

See also cfetch (p. 218), clabel (p. 221), store (p. 452), fetch (p. 264).

**cd**

**Command**

Change default directory.

The `cd` command changes the current default working directory. The current working directory is displayed as “Path = ...” at the bottom right of the EViews window.
Syntax

Command: \texttt{cd path\_name}

Examples

To change the default directory to “SAMPLE DATA” in the A drive:
\[
\text{cd "a:\sample data"}
\]

Notice that the quotes surround the entire directory name. If your name does not contain spaces, you may omit the quotes:
\[
\text{cd a:\test}
\]
changes the default directory to A:TEST.

Everything you save will be saved to the default directory, unless you specify a different directory in the \texttt{save} command.

Cross-references


\begin{tabular}{|l|l|}
\hline
\textbf{cdfplot} & \textbf{Group View | Series View} \\
\hline
\end{tabular}

Empirical distribution functions.

Displays empirical cumulative distribution functions, survivor functions, and quantiles with standard errors.

Syntax

Object View: \texttt{series\_name} cdfplot(options)

Note: due to the potential for very long computation times, standard errors will only be computed if there are fewer than 2500 observations.

Options

\begin{tabular}{|l|l|}
\hline
c (default) & Plot the empirical CDF. \\
s & Plot the empirical survivor function. \\
q & Plot the empirical quantiles. \\
a & Plot all CDF, survivor, and quantiles. \\
n & Do not include standard errors. \\
\hline
\end{tabular}
Examples

To plot the empirical cumulative distribution function of the series LWAGE:

```
lwage.cdfplot
```

Cross References


See `qqplot` (p. 375).

Confidence ellipses for coefficient restrictions.

The `cellipse` view displays confidence ellipses for pairs of coefficient restrictions for an estimation object.

Syntax

```
Object View:  object_name.cellipse(options) restrictions
```

Enter the object name, followed by a period, and the keyword `cellipse`. This should be followed by a list of the coefficient restrictions. Joint (multiple) coefficient restrictions should be separated by commas.
Options

ind = arg
Specifies whether and how to draw the individual coefficient intervals. The default is “ind = line” which plots the individual coefficient intervals as dashed lines. “ind = none” does not plot the individual intervals, while “ind = shade” plots the individual intervals as a shaded rectangle.

size = number (default = 0.95)
Set the size (level) of the confidence ellipse. You may specify more than one size by specifying a space separated list enclosed in double quotes.

dist = arg
Select the distribution to use for the critical value associated with the ellipse size. The default depends on estimation object and method. If the parameter estimates are least-squares based, the $F(2, n - 2)$ distribution is used; if the parameter estimates are likelihood based, the $\chi^2(2)$ distribution will be employed. “dist = f” forces use of the $F$-distribution, while “dist = c” uses the $\chi^2$-distribution.

p
Print the graph.

Examples
The two commands

```
eq1.ellipse c(1), c(2), c(3)
eq1.ellipse c(1)=0, c(2)=0, c(3)=0
```
both display a graph showing the 0.95-confidence ellipse for C(1) and C(2), C(1) and C(3), and C(2) and C(3).

```
pool1.ellipse(dist=c, size="0.9 0.7 0.5") c(1), c(2)
```
displays multiple confidence ellipses (contours) for C(1) and C(2).

Cross-references
See wald (p. 490).
Estimation of censored and truncated models.

Estimates models where the dependent variable is either censored or truncated. The allowable specifications include the standard Tobit model.

Syntax

Command: censored(options) y x1 x2 x3
Equation Method: eq_name.censored(options) y x1 x2 x3

Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l = number</td>
<td>Set value for the left censoring limit.</td>
</tr>
<tr>
<td>(default = 0)</td>
<td></td>
</tr>
<tr>
<td>r = number</td>
<td>Set value for the right censoring limit.</td>
</tr>
<tr>
<td>(default = none)</td>
<td></td>
</tr>
<tr>
<td>l = series_name, i</td>
<td>Set series name of the indicator variable for the left censoring limit.</td>
</tr>
<tr>
<td>r = series_name, i</td>
<td>Set series name of the indicator variable for the right censoring limit.</td>
</tr>
<tr>
<td>t</td>
<td>Estimate truncated model.</td>
</tr>
<tr>
<td>d = arg</td>
<td>Specify error distribution: “n” (normal), “l” (logistic), “x” (Type I extreme value).</td>
</tr>
<tr>
<td>(default = “n”)</td>
<td></td>
</tr>
<tr>
<td>q (default)</td>
<td>Use quadratic hill climbing for maximization algorithm.</td>
</tr>
<tr>
<td>r</td>
<td>Use Newton-Raphson for maximization algorithm.</td>
</tr>
<tr>
<td>b</td>
<td>Use Berndt-Hall-Hall-Hausman for maximization algorithm.</td>
</tr>
<tr>
<td>h</td>
<td>Quasi-maximum likelihood (QML) standard errors.</td>
</tr>
<tr>
<td>g</td>
<td>GLM standard errors.</td>
</tr>
<tr>
<td>m = integer</td>
<td>Set maximum number of iterations.</td>
</tr>
<tr>
<td>c = scalar</td>
<td>Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.</td>
</tr>
</tbody>
</table>
The command

\texttt{censored(h) hours c wage edu kids}

estimates a censored regression model of HOURS on a constant, WAGE, EDU, and KIDS with QML standard errors. This command uses the default normal likelihood, with left-censoring at HOURS = 0 and no right censoring, and the quadratic hill climbing algorithm.

Cross-references


<table>
<thead>
<tr>
<th>cfretch</th>
<th>Command</th>
</tr>
</thead>
</table>

\textit{cfretch} reads one or more series from the DRI Basic Economics Database into the active workfile. \textit{You must have the DRI database installed on your computer to use this feature.}

**Syntax**

Command: \texttt{cfretch series\_name}

**Examples**

\texttt{cfretch lhur gdp gnet}

reads the DRI series LHUR, GDP, and GNET into the current active workfile, performing frequency conversions if necessary.
Cross-references

EViews’ automatic frequency conversion is described in “Frequency Conversion” beginning on page 96 of the User’s Guide.

See also ccopy (p. 213), clabel (p. 221), store (p. 452), fetch (p. 264).

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>chdir</td>
</tr>
</tbody>
</table>

Change default directory.

See cd (p. 213).

<table>
<thead>
<tr>
<th>Logl View</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkderiv</td>
</tr>
</tbody>
</table>

Check derivatives of likelihood object.

Displays a table containing information on the numeric derivatives and user-supplied analytic derivatives (if available).

**Syntax**

Logl View: \[ \text{logl\_name}\text{.checkderiv}(\text{options}) \]

**Options**

- \[ p \]
  - Print the table of results.

**Examples**

```
ll1.ml
ll1.checkderiv
```

estimates a likelihood object named LL1 and displays a table that evaluates the numeric derivatives.

**Cross-references**


See also grads (p. 287) and makegrads (p. 322).
Chow test for stability.

Carries out Chow breakpoint and Chow forecast tests for parameter constancy.

Syntax

Command:   chow(options) obs1 obs2 obs3
Equation View: eq_name.chow(options) obs1 obs2 obs3

You must provide the breakpoint (dates or observation numbers) to be tested. To specify more than one breakpoint, separate the breakpoints by a space.

Options

f Chow forecast test. For this option, you must specify a single breakpoint to test (default performs breakpoint test).
p Print the result of test.

Examples

The commands

    ls m1 c gdp cpi ar(1)
    chow 1970:1 1980:1

perform a regression of M1 on a constant, GDP, and CPI with first order autoregressive errors, and test whether the parameters before the 1970’s, during the 1970’s, and after the 1970’s are “stable” using the Chow breakpoint test.

To regress the log of SPOT on a constant, log of P_US, and log of P_UK and carry out the Chow forecast test starting from 1973, enter the commands:

    equation ppp.ls log(spot) c log(p_us) log(p_uk)
    ppp.chow(f) 1973

You may also perform Chow tests on cross-section data. Suppose GENDER is a zero-one dummy variable and, when sorted, observations up to 533 have GENDER = 0 and for observations from 534 on GENDER = 1. Then the chow command tests whether the LWAGE equation has the same coefficients for observations with GENDER = 0 and GENDER = 1.

    sort gender
ls lwage c edu edu^2 union
chow 534

Cross-references
See “Chow’s Breakpoint Test” on page 553 of the User’s Guide for further discussion.
See also rls (p. 387).

<table>
<thead>
<tr>
<th>clabel</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>clabel</td>
<td>Display a DRI Basic Economics database series description.</td>
</tr>
<tr>
<td></td>
<td>clabel reads the description of a series from the DRI Basic Economics Database and displays it in the status line at the bottom of the EViews window.</td>
</tr>
<tr>
<td></td>
<td>Use this command to verify the contents of a given DRI database series name. You must have the DRI database installed on your computer to use this feature.</td>
</tr>
<tr>
<td></td>
<td>Syntax</td>
</tr>
<tr>
<td></td>
<td>Command: clabel series_name</td>
</tr>
<tr>
<td>Examples</td>
<td></td>
</tr>
<tr>
<td>clabel lhur</td>
<td></td>
</tr>
<tr>
<td>displays the description of the DRI series LHUR on the status line.</td>
<td></td>
</tr>
<tr>
<td>Cross-references</td>
<td></td>
</tr>
<tr>
<td>See also ccopy (p. 213), cfetch (p. 218), read (p. 377), fetch (p. 264).</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cleartext</th>
<th>Var Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td>cleartext</td>
<td>Clear restriction text of var object.</td>
</tr>
<tr>
<td>Syntax</td>
<td></td>
</tr>
<tr>
<td>Var Proc: var_name.cleartext(arg)</td>
<td></td>
</tr>
<tr>
<td>You must specify the text type you wish to clear as an argument.</td>
<td></td>
</tr>
<tr>
<td>One of the following arguments is required:</td>
<td></td>
</tr>
</tbody>
</table>
Examples

\begin{verbatim}
var1.cleartext(svar)
var1.append(svar) @lr2(@u1)=0
\end{verbatim}

The first line clears the structural VAR identifying restrictions in VAR1. The next line specifies a new long-run restriction for a structural factorization.

Cross-references

See also `append (p. 186)`.

---

**close**

Command

Close object, program, or workfile.

Closing an object eliminates its window. If the object is named, it is still displayed in the workfile as an icon, otherwise it is deleted. Closing a program or workfile eliminates its window and removes it from memory. If a workfile has changed since you activated it, you will see a dialog box asking if you want to save it to disk.

Syntax

Command: `close object_name`

Examples

\begin{verbatim}
close gdp graph1 table2
lwage.hist
close lwage
\end{verbatim}

closes the three objects GDP, GRAPH1, and TABLE2.

on opens the LWAGE window and displays the histogram view of LWAGE, then closes the window.

Cross-references

See Chapter 1 of the *User’s Guide* for a discussion of basic control of EViews.
Declare a coefficient (column) vector.

Note: vector and coefficient objects are both column vectors. However, only coef elements are allowed in specifications of models or for sspace estimation.

Syntax

Command: \texttt{coef(n) coef\_name}

Follow the \texttt{coef} keyword with the number of coefficients in parentheses and a name for the object. If you omit the number of coefficients, EViews will create a vector of length 1.

Examples

\begin{verbatim}
coef(2) slope
ls lwage=c(1)+slope(1)*edu+slope(2)*edu^2
\end{verbatim}

The first line declares a coef object of length 2 named SLOPE. The second line runs a least squares regression and stores the estimated slope coefficients in SLOPE.

\begin{verbatim}
arch(2,2) sp500 c
coef beta=c
coef(6) beta
\end{verbatim}

The first line estimates a GARCH(2,2) model using the default coef vector C. (The “C” in an equation specification refers to the constant term, a series of ones.) The second line declares a coef object named BETA and copies the contents of C to BETA. (The “C” in the assignment statement refers to the default coef vector). The third line resizes BETA to “chop off” all elements except the first six, which are the estimated parameters from the GARCH(2,2) model. Note that since EViews stores coefficients with equations for later use, you will generally not need to perform this operation to save your coefficient vectors.

Cross-references

See “Coef” on page 137 for a full description of the coef object.

See also \texttt{vector} (p. 489).
**coefcov**

<table>
<thead>
<tr>
<th>Equation View</th>
<th>Logl View</th>
<th>Pool View</th>
<th>Sspace View</th>
<th>System View</th>
</tr>
</thead>
</table>

Coefficient covariance matrix.

Displays the covariances of the coefficient estimates for objects containing an estimated equation or equations.

**Syntax**

Object View: object_name.coefcov(options)

**Options**

- **p**
  - Print the coefficient covariance matrix.

**Examples**

The set of commands

```plaintext
equation eql.ls lwage c edu edu^2 union
eql.coefcov
```

declares and estimates an equation and displays the coefficient covariance matrix in a window. To store the coefficient covariance matrix as a (symmetric) matrix object, use “@coefcov”:

```plaintext
sym eqcov = eql.@coefcov
```

**Cross-references**

See also **coef** (p. 223) and **spec** (p. 441).

---

**coint**

| Command || Group View | Var View |
|---------|------------|---------|

Johansen’s cointegration test.

**Syntax**

- **Command:** `coint(test_option,n,option) y1 y2 y3`
- **Command:** `coint(test_option,n,option) y1 y2 y3 @ x1 x2 x3`
- **Group View:** `group_name.coint(test_option,n,option)`
- **Var View:** `var_name.coint(test_option,n,option)`

In command form, you should enter the `coint` keyword followed by a list of series or group names within which you wish to test for cointegration. Each name should be sepa-
rated by a space. To use exogenous variables, such as seasonal dummy variables in the test, list the names after an "@"-sign.

**Note:** the reported critical values assume no exogenous variables other than an intercept and trend.

When used as a group or var view, `coint` tests for cointegration among the series in the group or var. If the var object contains exogenous variables, the cointegration test will use those exogenous variables. However, if you explicitly list the exogenous variables with an "@"-sign, then only those that are listed will be used in the test.

**Options**

You must specify the test option followed by the number of lags \( n \) in parentheses separated by a comma. You must choose one of the following six test options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>No deterministic trend in the data, and no intercept or trend in the cointegrating equation.</td>
</tr>
<tr>
<td>b</td>
<td>No deterministic trend in the data, and an intercept but no trend in the cointegrating equation.</td>
</tr>
<tr>
<td>c</td>
<td>Linear trend in the data, and an intercept but no trend in the cointegrating equation.</td>
</tr>
<tr>
<td>d</td>
<td>Linear trend in the data, and both an intercept and a trend in the cointegrating equation.</td>
</tr>
<tr>
<td>e</td>
<td>Quadratic trend in the data, and both an intercept and a trend in the cointegrating equation.</td>
</tr>
<tr>
<td>s</td>
<td>Summarize the results of all 5 options (a-e).</td>
</tr>
<tr>
<td>restrict</td>
<td>Impose restrictions as specified by the append (coint) proc.</td>
</tr>
</tbody>
</table>

\[ m = \text{integer} \]

Maximum number of iterations for restricted estimation (only valid if you choose the restrict option).

\[ c = \text{scalar} \]

Convergence criterion for restricted estimation. (only valid if you choose the restrict option).
Other Options:

Other Options:

Examples

coint(s,4) gdp m1 tb3

summarizes the results of the Johansen cointegration test among the three series GDP, M1, and TB3 for all five specifications of trend. The test equation uses lags of up to order four.

var1.coint(c,12) @

carryes out the Johansen test for the series in the var object named VAR1. The "@"-sign without a list of exogenous variables ensures that the test does not include any exogenous variables in VAR1.

Cross-references


<table>
<thead>
<tr>
<th>comment</th>
<th>Table Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Adds or removes a comment in a table cell.

Syntax

Table Proc: table_name.comment(cell_arg) [comment_arg]

where cell_arg, which identifies the cell, can take one of the following forms:
and where comment_arg is a string expression enclosed in double quotes. If command_arg is omitted, a previously defined comment will be removed.

Examples
To add a comment “hello world” to the cell in the second row, fourth column, you may use one of the following:

```
tabl.comment(d2) "hello world"
tabl.comment(r2c4) "hello world"
tabl.comment(2,d) "hello world"
tabl.comment(2,4) "hello world"
```

To remove a comment simply omit the comment_arg:

```
tabl.comment(d2)
```
clears the comment (if present) from the second row, fourth column.

Cross-references
For additional discussion of tables see Chapter 4, “Working with Tables”. See also setlines (p. 427) and setmerge (p. 428).

<table>
<thead>
<tr>
<th>control</th>
<th>Model Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td>solve for values of control variable so that target series matches trajectory.</td>
<td></td>
</tr>
</tbody>
</table>

**Syntax**

```
Model Proc:   model_name.control control_var target_var trajectory
```

Specify the name of the control variable, followed by the target variable, and then the trajectory you wish to achieve for the target variable. EViews will solve for the values of the control so that the target equals the trajectory over the current workfile sample.
Examples

\texttt{m1.control myvar targetvar trajvar}

will put into MYVAR the values that lead the solution of the model for TARGETVAR to match TRAJVAR for the workfile sample.

Cross-references


\begin{verbatim}
<table>
<thead>
<tr>
<th>copy</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Copy an object. Duplicate objects within and across workfiles and databases.</td>
</tr>
<tr>
<td>Syntax</td>
<td>Command: copy\texttt{(options) src_object_name dest_object_name}</td>
</tr>
<tr>
<td></td>
<td>To make copies of the object within the active workfile, list the name of the original object followed by the name for the copy.</td>
</tr>
<tr>
<td></td>
<td>To make copies of objects within a database, you can precede the name of the object with the database name and a double colon “::”.</td>
</tr>
<tr>
<td></td>
<td>You can also copy objects between a workfile and a database, or between two databases. Simply prefix the object name with the database name and double colon “::”.</td>
</tr>
<tr>
<td></td>
<td>You can copy several objects with one command by using the wild card characters “?” (to match any single character) and “*” (to match zero or more characters).</td>
</tr>
<tr>
<td>Options</td>
<td>When copying a group object from a workfile to a database</td>
</tr>
<tr>
<td></td>
<td>\texttt{g=arg} Group copy from workfile to database: “s” (copy group definition and series as separate objects), “t” (copy group definition and series as one object), “d” (copy series only as separate objects), “l” (copy group definition only).</td>
</tr>
<tr>
<td></td>
<td>When copying a group object from a database to a workfile:</td>
</tr>
</tbody>
</table>
\end{verbatim}
The following options control the frequency conversion method when copying series and group objects to a workfile, converting from low to high frequency:

```
c = arg
```

Low to high conversion methods: “r” (constant match average), “d” (constant match sum), “q” (quadratic match average), “t” (quadratic match sum), “i” (linear match last), “c” (cubic match last).

The following options control the frequency conversion method when copying series and group objects to a workfile, converting from high to low frequency:

```
c = arg
```

High to low conversion methods removing NAs: “a” (average of the nonmissing observations), “s” (sum of the nonmissing observations), “f” (first nonmissing observation), “l” (last nonmissing observation), “x” (maximum nonmissing observation), “m” (minimum nonmissing observation).

High to low conversion methods propagating NAs: “an” or “na” (average, propagating missings), “sn” or “ns” (sum, propagating missings), “fn” or “nf” (first, propagating missings), “ln” or “nl” (last, propagating missings), “xn” or “nx” (maximum, propagating missings), “mn” or “nm” (minimum, propagating missings).

Note that if no method is specified, the series specific default conversion method or the global settings will be employed.

**Examples**

```
copy good_equation best_equation
```

makes a duplicate copy of GOOD_EQUATION named BEST_EQUATION in the current workfile.
copy gdp usdat::gdp

copies GDP in the current workfile to the database named USDAT with the name GDP. To copy GDP in the default database to the database named MACRO1 with the new name GDP_US, use the command

copy ::gdp macro1::gdp_us

The command

```plaintext
copy ::gdp ::gdp2
```

makes a backup copy of GDP in the default database with a different name GDP2.

```plaintext
copy gd* findat::
```

makes a duplicate of all objects in the current workfile with name starting with GD to the database named FINDAT.

Cross-references

See “Copying Objects” on page 250 of the User’s Guide for a discussion of copying and moving objects.

See also fetch (p. 264), setconvert (p. 410), and store (p. 452).

<table>
<thead>
<tr>
<th>Command</th>
<th>Group View</th>
<th>Matrix View</th>
<th>Sym View</th>
</tr>
</thead>
<tbody>
<tr>
<td>cor</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Correlation matrix.

Syntax

- Command: `cor(options) ser1 ser2 ser3`
- Group View: `group_name.cor(options)`
- Matrix View: `matrix_name.cor(options)`

In command form, EViews will create an untitled group from the listed series, and then display the correlation matrix view for that group.

When used as a matrix view, `cor` displays the correlation among the columns of the matrix.
correl—231

Options

- i: Compute correlations using pairwise samples (default is to use the common sample).
- p: Print the correlation matrix.

Examples

```
cor height weight age
```
displays a 3 by 3 correlation matrix for the three series HEIGHT, WEIGHT, and AGE.

```
group mygroup height weight age
mygroup.cor
```
displays the equivalent view using the group MYGROUP.

Cross-references

See also `cov` (p. 234), `@cor` (p. 530), and `@cov` (p. 530).

---

**correl** | Equation View | Group View | Series View | Var View

Display autocorrelation and partial correlations.

Displays the autocorrelation and partial correlation functions of the specified series together with the Q-statistics and p-values associated with each lag.

When used with equation objects, `correl` displays the correlogram of the residuals of the equation.

**Syntax**

```
Object View: object_name.correl(n, options)
```

You must specify the largest lag `n` to compute the autocorrelations.

**Options**

- p: Print the correlograms.

**Var View Options:**

- graph: Display correlograms (graphs).
- byser: Display autocorrelations in tabular form by series.
- bylag: Display autocorrelations in tabular form by lag.
Examples

\texttt{m1.correl(24)}

Displays the correlograms of the M1 series for up to 24 lags.

Cross-references

See Chapter 11 of the \textit{User’s Guide} for a discussion of autocorrelation (p. 302) and partial correlation (p. 303) functions.

See also \texttt{correlsq} (p. 232).

\begin{center}
\begin{tabular}{|l|l|}
\hline
\textbf{correlsq} & \textit{Equation View} \\
\hline
\end{tabular}
\end{center}

Correlogram of squared residuals.

Displays the autocorrelation and partial correlation functions of the squared residuals from an estimated equation, together with the $Q$-statistics and $p$-values associated with each lag.

Syntax

View: \texttt{equation\_name.correl(n, options)}

Options

<table>
<thead>
<tr>
<th>n</th>
<th>Specify the number of lags of the correlograms to display.</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>Print the correlograms.</td>
</tr>
</tbody>
</table>

Examples

\texttt{eq1.correl(24)}

displays the correlograms of the squared residuals for up to 24 lags, for equation EQ1.

Cross-references

See Chapter 11 of the \textit{User’s Guide} for a discussion of autocorrelation (p. 302) and partial correlation (p. 303) functions.

See also \texttt{correl} (p. 231).
### Estimates models where the dependent variable is a nonnegative integer count.

#### Syntax

**Command:**  
`count(options) y x1 x2 x3`  
**Equation Method:**  
`eq_name.count(options) y x1 x2 x3`

Follow the `count` keyword by the name of the dependent variable and a list of regressors, each separated by a space.

#### Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>v = positive number (default = 1)</code></td>
<td>Specify fixed QML parameter for normal and negative binomial distributions.</td>
</tr>
<tr>
<td><code>q (default)</code></td>
<td>Use quadratic hill-climbing as the maximization algorithm.</td>
</tr>
<tr>
<td><code>r</code></td>
<td>Use Newton-Raphson as the maximization algorithm.</td>
</tr>
<tr>
<td><code>b</code></td>
<td>Use Berndt-Hall-Hall-Hausman as the maximization algorithm.</td>
</tr>
<tr>
<td><code>h</code></td>
<td>Quasi-maximum likelihood (QML) standard errors.</td>
</tr>
<tr>
<td><code>g</code></td>
<td>GLM standard errors.</td>
</tr>
<tr>
<td><code>m = integer</code></td>
<td>Set maximum number of iterations.</td>
</tr>
<tr>
<td><code>c = scalar</code></td>
<td>Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.</td>
</tr>
<tr>
<td><code>s</code></td>
<td>Use the current coefficient values in C as starting values.</td>
</tr>
</tbody>
</table>
Examples

The command

\[
\text{count(d=n,v=2,g) y c x1 x2}
\]

estimates a normal QML count model of \(Y\) on a constant, \(X1\), and \(X2\) with fixed variance parameter 2 and GLM standard errors.

\[
\text{equation eql.count arrest c job police}
\]

\[
\text{eql.makeresid(g) res_g}
\]

estimates a Poisson count model of \(ARREST\) on a constant, \(JOB\), and \(POLICE\) and stores the generalized residuals as \(RES_G\).

\[
\text{equation eql.count(d=p) y c x1}
\]

\[
\text{eql.fit yhat}
\]

estimates a Poisson count model of \(Y\) on a constant and \(X1\), and saves the fitted values (conditional mean) in the series \(YHAT\).

\[
\text{equation eql.count(d=p, h) y c x1}
\]

estimates the same model with QML standard errors and covariances.

Cross-references


| COV | Command || Group View | Matrix View | Sym View |
|-----|-----------|-------------|-------------|-----------|

Compute covariance matrix.

Syntax

- Command: \( \text{cov(options)} \ \text{arg1} \ [\text{arg2} \ \text{arg3} \ ...]\)
- Group View: \( \text{group_name.cov(options)} \)
- Matrix View: \( \text{matrix_name.cov(options)} \)

In command form, EViews will create an untitled group from the listed series or groups, and then display the covariance matrix view for that group.
When used as a matrix view, `cov` displays the covariance among the columns of the matrix.

Options

- **i**  Compute covariances using pairwise samples (default is to use the common sample).
- **p**  Print the correlation matrix.

Examples

```eviews
    group grp1 height weight age
group.grp1.cov
```

displays a \(3 \times 3\) covariance matrix for the three series in GRP1.

Cross-references

See also `cor` (p. 230), `@cor` (p. 530), and `@cov` (p. 530).

---

**create**

Create workfile.

No longer supported. This command has been replaced by `wfcreate` (p. 491).

**cross**

Displays cross correlations (correlograms) for a pair of series.

Syntax

- **Command:** `cross(n,options) arg1 [arg2 arg3 ...]`
- **Group View:** `group_name.cross(n,options)`

You must specify, as an option, the number of lags \(n\) to compute the cross correlations as an option. In command form, EViews will create an untitled group from the specified series and groups, and will display the cross correlation view for the group. When used as a group view, cross correlations will be computed for the first two series in the group.
Options

The number of lags \( n \) for which to compute the cross correlations must be specified as the first option. The following options may be specified inside the parentheses after the number of lags:

- \( p \)   Print the cross correlogram.

Examples

\[
\text{cross}(36) \ \log(m1) \ \text{dlog(cpi)}
\]
displays the cross correlogram between the log of M1 and the first difference of the log of CPI up to 36 leads and lags.

\[
\text{equation eq1.arch sp500 c}
\text{eq1.makeresid(s) res_std}
\text{cross(24) res_std^2 res_std}
\]
The first line estimates a GARCH(1,1) model and the second line retrieves the standardized residuals. The third line plots the cross correlogram up to 24 leads and lags between the squared standardized residual and the standardized residual. This correlogram provides a rough check of asymmetry in ARCH effects.

Cross-references


<table>
<thead>
<tr>
<th>data</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter data from keyboard.</td>
<td>Opens an unnamed group window to edit one or more series.</td>
</tr>
</tbody>
</table>

Syntax

Command: \texttt{data arg1 [arg2 arg3 ...]}

Follow the \texttt{data} keyword by a list of series and group names. You can list existing names or new names. Unrecognized names will cause new series to be added to the workfile. These series will be initialized with the value “NA”.
Examples

data group1 newx newy

opens a group window containing the series in group GROUP1 and the series NEWX and NEWY.

Cross-references

See “Entering Data” on page 87 of the User’s Guide for a discussion of the process of entering data from the keyboard.

datetlabel

Control labeling of the bottom date/time axis in time plots.

dates sets options that are specific to appearance of time/date labeling. Many of the options that also affect the appearance of the date axis are set by the axis (p. 197) command with the “bottom” option. These options include tick control, label and font options, and grid lines.

Syntax

Graph Proc: graph_name.datelabel option_list

Options

format(dateformat[,delimiter]) Specify date format: dateformat = “auto” or dateformat = “string”, where the string argument is one of the supported formats: “yy”, “yyyy:mm”, “yyyy:q”, “yy:q”, “mm:dd:yyyy”, “mm:dd:yy”. Each letter represents a single display digit representing year (“y”), quarter (“q”), month (“m”), and day (“d”). The optional delimiter argument allows you to control the date separator.
Examples

\texttt{graph1.datelabel format(yyyy:mm)}

will display dates with four-digit years followed by the default delimiter “:” and a two-digit month (e.g. – “1974:04”).

\texttt{graph1.datelabel format(yy:mm, q)}

will display a two-digit year followed by a “q” separator and then a two-digit month (e.g. – “74:04”)

\texttt{graph1.datelabel interval(y, 2, 1951)}

specifies labels every two years on odd numbered years.

Cross-references


See also \texttt{axis (p. 197)}, \texttt{scale (p. 397). options (p. 343)}, and \texttt{setelem (p. 412)}.
See the replacement command datelabel (p. 237).

Open or create a database.

If the specified database does not exist, a new (empty) database will be created and opened. The opened database will become the default database.

Syntax

```
Command:   db [path]db_name [as shorthand_name]
```

Follow the `db` command by the name of the database to be opened or to be created (if it does not already exist). You can include a path name to work with a database not in the default path.

You may use the “as” keyword to provide an optional `shorthand_name`, a short text label which may be used to refer to the database in commands and programs. If you leave this field blank, a default `shorthand_name` will be assigned automatically. See “Database Short-hands” on page 245 of the User’s Guide for additional discussion.

Options

See `dbopen` (p. 242) for a list of available options for working with foreign format databases.

Examples

```
db findat
```

opens the database FINDAT in the default path and makes it the default database from which to store and fetch objects. If the database FINDAT does not already exist, an empty database named FINDAT will be created and opened.

Cross-references


See also `dbcreate` (p. 240) and `dbopen` (p. 242).
**dbcopy** | **Command**

Make a copy of an existing database.

**Syntax**

Command:  

```
dbcopy [path]\source_name [path]\copy_name
```

Follow the `dbcopy` command by the name of the existing database and a name for the copy. You should include a path name to copy a database that is not in the default directory. All files associated with the database will be copied.

**Examples**

```
dbcopy usdat c:\backup\usdat
```

makes a copy of all files associated with the database USDAT in the default path and stores it in the C:\BACKUP directory under the name USDAT.

**Cross-references**


See also `dbrename` (p. 244) and `dbdelete` (p. 241).

---

**dbcreate** | **Command**

Create a new database.

**Syntax**

Command:  

```
dbcreate [path]\db_name [as shorthand_name]
```

Follow the `dbcreate` keyword by a name for the new database. You can include a path name to create a database not in the default directory. The new database will become the default database.

You may use the “as” keyword to provide an optional `shorthand_name`, a short text label which may be used to refer to the open database in commands and programs. If you leave this field blank, a default `shorthand_name` will be assigned automatically. See “Database Shorthands” on page 245 of the *User’s Guide* for additional discussion.
Examples

\texttt{dbcreate macrodat}

creates a new database named MACRODAT in the default path and makes it the default database from which to store and fetch objects. This command will issue an error message if a database named MACRODAT already exists. To open an existing database, use “dbopen” or “db”.

Options

See \texttt{dbopen}\ (p. 242) for available options for working with foreign format databases.

Cross-references


See also \texttt{db}\ (p. 239) and \texttt{dbopen}\ (p. 242).

<table>
<thead>
<tr>
<th>\textbf{dbdelete}</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete an existing database (all files associated with the specified database).</td>
<td></td>
</tr>
</tbody>
</table>

Syntax

\texttt{Command: \texttt{dbdelete}} \texttt{\{path\}db\_name}

Follow the \texttt{dbdelete} keyword by the name of the database to be deleted. You can include a path name to delete a database not in the default path.

Options

See \texttt{dbopen}\ (p. 242) for available options for working with foreign format databases.

Examples

\texttt{dbdelete c:\temp\testdat}

deletes all files associated with the TESTDAT database in the specified directory.

Cross-references


See also \texttt{dbcopy}\ (p. 240) and \texttt{dbdelete}\ (p. 241).
Open an existing database.

Syntax

Command: \texttt{dbopen [path\/]db\_name [as shorthand\_name]}

Follow the \texttt{dbopen} keyword with the name of a database. You should include a path name to open a database not in the default path. The opened database will become the default database.

You may use the “as” keyword to provide an optional \textit{shorthand\_name}, a short text label which is used to refer to the open database in commands and programs. If you leave this field blank, a default \textit{shorthand\_name} will be assigned automatically. See “Database Short-hands” on page 245 of the \textit{User’s Guide} for additional discussion.

By default, EViews will use the extension of the database file to determine type. For example, files with the extension “.EDB” will be opened as an EViews database, while files with the extension “.IN7” will be opened as a GiveWin database. You may use options to specify an explicit type.

Options

\begin{verbatim}
type = arg, t = arg
\end{verbatim}

The following options may be required when connecting to a remote server:

\begin{verbatim}
s = server\_id, server = server\_id
u = user, username = user
p = pswd, password = pswd
\end{verbatim}
Server name.
Username.
Password.
Examples

dbopen c:\data\us1

opens a database named US1 in the C:\DATA directory. The command

dbopen us1

opens a database in the default path.

If the specified database does not exist, EViews will issue an error message. You should use
db or dbcreate to create a new database.

Cross-references


See also db (p. 239) and dbcreate (p. 240).

<table>
<thead>
<tr>
<th>dbpack</th>
<th>Command</th>
</tr>
</thead>
</table>

Pack an existing database.

Syntax

Command: dbpack [path]\db_name

Follow the dbpack keyword by a database name. You can include a path name to pack a
database not in the default path.

Examples

dbpack findat

Packs the database named FINDAT in the default path.

Cross-references

See “Packing the Database” on page 268 of the User's Guide for additional discussion.

See also dbrebuild (p. 243) and dbrepair (p. 245)

<table>
<thead>
<tr>
<th>dbrebuild</th>
<th>Command</th>
</tr>
</thead>
</table>

Rebuild an existing database.

Rebuild a seriously damaged database into a new database file.
Syntax

Command: \texttt{dbrebuild}\path\source\name\ path\dest\name

Follow the \texttt{dbrebuild} keyword by the name of the database to be rebuilt, and then a new database name.

Examples

If you issue the command

\texttt{dbrebuild testdat fixed_testdat}

EViews will attempt to rebuild the database TESTDAT into the database FIXED_TESTDAT in the default directory.

Cross-references

Note that \texttt{dbrepair} may be able to repair the existing database if the damage is not particularly serious. You should attempt to repair the database before rebuilding. See “Maintaining the Database” on page 267 of the User’s Guide for a discussion.

See also \texttt{dbpack} (p. 243) and \texttt{dbrepair} (p. 245).

\begin{verbatim}
\textbf{dbrename} \hspace{1cm} \textbf{Command}
\end{verbatim}

Rename an existing database.

dbrename renames all files associated with the specified database.

Syntax

Command: \texttt{dbrename} \path\old\name\path\new\name

Follow the \texttt{dbrename} keyword with the current name of an existing database and the new name for the database.

Options

See \texttt{dbopen} (p. 242) for available options for working with foreign format databases.

Examples

\texttt{dbrename testdat mydat}

Renames all files associated with the TESTDAT database in the specified directory to MYDAT in the default directory.
Cross-references


See also db (p. 239) and dbcreate (p. 240). See also dbcopy (p. 240) and dbdelete (p. 241).

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dbrepair</strong></td>
</tr>
</tbody>
</table>

Repair an existing database.

This command is used to repair a damaged database.

**Syntax**

Command: `dbrepair [path\]db_name`

Follow the `dbrepair` keyword by the name of the database to repair. You should include a path name to repair a database not in the default path.

**Examples**

`dbrepair testdat`

EViews will attempt to repair the database TESTDAT in the default directory.

Cross-references

If the database is severely damaged, you may have to rebuild it into a new database using the `dbrebuild` command. See “Maintaining the Database” on page 267 of the User’s Guide for a discussion of EViews database maintenance.

See also dbpack (p. 243) and dbrebuild (p. 243).

<table>
<thead>
<tr>
<th>Var View</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>decomp</strong></td>
</tr>
</tbody>
</table>

Variance decomposition in VARs.

**Syntax**

Var View: `var_name.decomp(n, options) series_list [@ @ ordering]`

List the series names in the VAR whose variance decomposition you would like to compute. You may optionally specify the ordering for the factorization after two “@”-signs.

You must specify the number of periods `n` over which to compute the variance decompositions.
## Options

In addition, there are the following options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>g (default)</strong></td>
<td>Display combined graphs, with the decompositions for each variable shown in a graph.</td>
</tr>
<tr>
<td><strong>m</strong></td>
<td>Display multiple graphs, with each response-shock pair shown in a separate graph.</td>
</tr>
<tr>
<td><strong>t</strong></td>
<td>Show numerical results in table.</td>
</tr>
<tr>
<td><strong>imp=arg</strong> <em>(default = “chol”)</em></td>
<td>Type of factorization for the decomposition: “chol” (Cholesky with d.f. correction), “mlechol” (Cholesky without d.f. correction), “struct” (structural).&lt;br&gt;The structural factorization is based on the estimated structural VAR. To use this option, you must first estimate the structural decomposition; see <code>svar</code> (p. 456).&lt;br&gt;The option “imp=mlechol” is provided for backward compatibility with EViews 3.x and earlier.</td>
</tr>
<tr>
<td><strong>se=mc</strong></td>
<td>Monte Carlo standard errors. Must specify the number of replications with the “rep=” option.&lt;br&gt;Currently available only when you have specified the Cholesky factorization (using the “imp=chol” option).</td>
</tr>
<tr>
<td><strong>rep=integer</strong></td>
<td>Number of Monte Carlo replications to be used in computing the standard errors. Must be used with the “se=mc” option.</td>
</tr>
<tr>
<td><strong>matbys=name</strong></td>
<td>Save responses by shocks (impulses) in named matrix. The first column is the response of the first variable to the first shock, the second column is the response of the second variable to the first shock, and so on.</td>
</tr>
<tr>
<td><strong>matbyr=name</strong></td>
<td>Save responses by response series in named matrix. The first column is the response of the first variable to the first shock, the second column is the response of the first variable to the second shock, and so on.</td>
</tr>
<tr>
<td><strong>p</strong></td>
<td>Print results.</td>
</tr>
</tbody>
</table>
If you use the “matbys=” or “matbyr=” options to store the results in a matrix, two matrices will be returned. The matrix with the specified name contains the variance decompositions, while the matrix with “_FSE” appended to the name contains the forecast standard errors for each response variable. If you have requested Monte Carlo standard errors, there will be a third matrix with “_SE” appended to the name which contains the variance decomposition standard errors.

**Examples**

```plaintext
var var1.ls 1 4 ml gdp cpi
var1.decomp(10,t) gdp
```

The first line declares and estimates a VAR with three variables and lags from 1 to 4. The second line tabulates the variance decompositions of GDP up to 10 periods using the ordering as specified in VAR1.

```plaintext
var1.decomp(10,t) gdp @ @ cpi gdp ml
```

performs the same variance decomposition as above using a different ordering.

**Cross-references**

See “Variance Decomposition” on page 701 of the *User’s Guide* for additional details.

See also `impulse` (p. 296).

---

**define**

Define cross section members (identifiers) in a pool.

**Syntax**

```plaintext
Pool View: pool_name.define id1 [id2 id3 ...]
```

List the cross section identifiers after the `define` keyword.

**Examples**

```plaintext
pool spot uk jpn ger can
spot.def uk ger ita fra
```

The first line declares a pool object named SPOT with cross section identifiers UK, JPN, GER, and CAN. The second line redefines the identifiers to be UK, GER, ITA, and FRA.

**Cross-references**

See Chapter 27 of the *User’s Guide* for a discussion of cross-section identifiers.
See also add (p. 179), drop (p. 255) and pool (p. 372).

| delete | Command || Pool Proc |
|--------|----------|-------------|

Deletes objects from a workfile or a database, or removes identifiers from a pool.

**Syntax**

Command: `delete arg1 [arg2 arg3 ...]`

Pool Proc: `pool_name.delete pool_ser1 [pool_ser2 pool_ser3 ...]`

Follow the keyword by a list of the names of any objects you wish to remove from the current workfile. Deleting does not remove objects that have been stored on disk in EViews database files.

You can delete an object from a database by prefixing the name with the database name and a double colon. You can use a pattern to delete all objects from a workfile or database with names that match the pattern. Use the “?” to match any one character and the “*” to match zero or more characters.

*When used as a pool procedure*, `delete` allows you to delete series from the workfile using pool series names. You may use the cross-section identifier “?” in the series names.

If you use `delete` in a program file, EViews will delete the listed objects without prompting you to confirm each deletion.

**Examples**

To delete all objects in the workfile with names beginning with “temp”:

```plaintext
delete temp*
```

To delete the objects CONS and INVEST from the database MACRO1:

```plaintext
delete macro1::cons macro1::invest
```

To delete all series in the workfile with names beginning with CPI that are followed by identifiers in the pool object MYPOOL.

```plaintext
mypool.delete cpi?
```

**Cross-references**

Examine derivatives of the equation specification.

Display information about the derivatives of the equation specification in tabular, graphical or summary form.

The (default) summary form shows information about how the derivative of the equation specification was computed, and will display the analytic expression for the derivative, or a note indicating that the derivative was computed numerically. The tabular form shows a spreadsheet view of the derivatives of the regression specification with respect to each coefficient for each observation. The graphical form shows this information in a multiple line graph.

Syntax

Equation View: equation_name.derivs(options)

Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>Display multiple graph showing the derivatives of the equation specification with respect to the coefficients, evaluated at each observation.</td>
</tr>
<tr>
<td>t</td>
<td>Display spreadsheet view of the values of the derivatives with respect to the coefficients evaluated at each observation.</td>
</tr>
<tr>
<td>p</td>
<td>Print results.</td>
</tr>
</tbody>
</table>

Note that the “g” and “t” options may not be used at the same time.

Examples

To show a table view of the derivatives:

eql.derivs(t)

To display and print the summary view

eql.derivs(p)

Cross-references

See also *makederivs* (p. 320) for additional routines for examining derivatives, and *grads* (p. 287), and *makegrads* (p. 322) for corresponding routines for gradients.

### describe Pool View

Computes and displays descriptive statistics for the pooled data.

**Syntax**

Pool View: `pool_name.describe(options) pool_ser1 [pool_ser2 pool_ser3 ...]`

List the name of pool series for which to compute the descriptive statistics. You may use the cross-section identifier “?” in the series names.

By default, statistics are computed for each pool series, on the stacked data, using only observations where all of the listed series have nonmissing data. A missing observation in any one series causes that observation to be dropped for all series. You may change this default treatment of NAs using the “i” and “b” options.

EViews also allows you to compute statistics with the cross-section means removed, statistics for each cross-sectional series in a pool series, and statistics for each period, taken across all cross-section units.

**Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>Stack data and subtract cross-section specific means from each variable—this option provides the within estimators.</td>
</tr>
<tr>
<td>c</td>
<td>Do not stack data—compute statistics individually for each cross-sectional unit.</td>
</tr>
<tr>
<td>t</td>
<td>Time period specific—compute statistics for each period, taken over all cross-section identifiers.</td>
</tr>
<tr>
<td>i</td>
<td>Individual sample—includes every valid observation for the series even if data are missing from other series in the list.</td>
</tr>
<tr>
<td>b</td>
<td>Balanced sample—constrains each cross-section to have the <em>same observations</em>. If an observation is missing for any series, in any cross-section, it will be dropped for all cross-sections.</td>
</tr>
<tr>
<td>p</td>
<td>Print the descriptive statistics.</td>
</tr>
</tbody>
</table>
Examples

    pool1.describe(m) gdp? inv? cpi?

displays the “within” descriptive statistics of the three series GDP, INV, CPI for the POOL1 cross-section members.

    pool1.describe(t) gdp?

computes the statistics for GDP for each period, taken across each of the cross-section identifiers.

Cross-references

See Chapter 27 of the User’s Guide for a discussion of the computation of these statistics, and a description of individual and balanced samples.

<table>
<thead>
<tr>
<th>displayname</th>
<th>Object Proc</th>
</tr>
</thead>
</table>

Display names for objects.

Attaches a display name to an object which may be used in tables and graphs in place of the standard object name.

Syntax

    Object Proc:     object_name.displayname display_name

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in object names.

Examples

    hrs.displayname Hours Worked
    hrs.label

The first line attaches a display name “Hours Worked” to the object HRS and the second line displays the label view of HRS, including its display name.

    gdp.displayname US Gross Domestic Product
    plot gdp

The first line attaches a display name “US Gross Domestic Product” to the series GDP. The line graph view of GDP from the second line will use the display name as the legend.
Cross-references


See also label (p. 302) and legend (p. 304)

**do**  Command

Execute without opening window.

**Syntax**

Command: \texttt{do \textit{procedure}}

\texttt{do} is most useful in EViews programs where you wish to run a series of commands without opening windows in the workfile area.

**Examples**

\texttt{output(t) c:\result\junk1}
\texttt{do gdp.adf(c,4,p)}

The first line redirects table output to a file on disk. The second line carries out a unit root test of GDP and prints the results to the disk file.

**Cross-references**

See also show (p. 432).

**draw**  Graph Proc

Place horizontal or vertical lines and shaded areas on the graph.

**Syntax**

Graph Proc: \texttt{graph\_name.draw(object\_type, axis\_id [,options]) position1 [position2]}

where \texttt{object\_type} may be one of the following:

- \texttt{line / l} A solid line
- \texttt{dashline / d} A dashed line
- \texttt{shade} A shaded area

and where \texttt{axis\_id} may take the values:
If drawing a line, the drawing position is taken from position1. If drawing a shaded area, you must provide a position1 and position2 to define the vertical or horizontal boundaries of the shaded region.

Line/Shade Options

The following options may be provided to change the characteristics of the specified line or shade:

- **left / l** Draw a horizontal line or shade using the left axis to define the drawing position
- **right / r** Draw a horizontal line or shade using the right axis to define the drawing position
- **bottom / b** Draw a vertical line or shade using the bottom axis to define the drawing position

**Examples**

The command
graphs a horizontal blue line at the value “5.25” as measured on the left axis while

draws a shaded horizontal region bounded by the right axis values “7.1” and “9.7”. You
can also draw vertical regions by using the “bottom” axis_id:

draws a shaded vertical region bounded by the dates “1980:1” and “1990:2”.

Cross-references


See also “Graph” (p. 143) for a summary of the graph object command language.

driconvert  Command

Convert the entire DRI Basic Economics database into an existing EViews database.

You must create an EViews database to store the converted DRI data before you use this
call. This command may be very time-consuming.

Syntax

Command:   driconvert db name

Follow the command by listing the name of an existing EViews database into which you
would like to copy the DRI data. You can include a path name to specify a database not in
the default path.

Examples

dbcreate dribasic
driconvert dribasic
driconvert c:\mydata\dribase

The first line creates a new (empty) database named DRIBASIC in the default directory.
The second line reads all the data in the DRI Basic Economics database and copies them
into the DRIBASIC database. The last example copies the DRI data into the database
DRIBASIC that is located in the C:\MYDATA directory.

Cross-references

See also `dbcreate` (p. 240) and `db` (p. 239).

<table>
<thead>
<tr>
<th>drop</th>
<th>Group Proc</th>
<th>Pool Proc</th>
</tr>
</thead>
</table>

Drops series from a group or drop cross-section members from a pool.

**Syntax**

Group Proc: `group_name.drop ser1 [ser2 ser3 ...]`

Pool Proc: `pool_name.drop id1 [id2 id3 ...]`

List the series or cross-section members to be dropped from the group or pool.

**Examples**

```plaintext
group gdplags gdp(-1 to -4)
gdplags.drop gdp(-4) gdp(-3)
```

drops the two series GDP(-4) and GDP(-3) from the group GDPLAGS.

To drop the cross-section members JPN, KOR, and HK from the pool CROSSSC

```plaintext
crossc.drop jpn kor hk
```

**Cross-references**

See also `add` (p. 179).

<table>
<thead>
<tr>
<th>dtable</th>
<th>Group View</th>
</tr>
</thead>
</table>

Dated data report table.

This group view is designed to make tables for reporting and presenting data, forecasts, and simulation results. You can display various transformations and various frequencies of the data in the same table.

The `dtable` view is currently available only for annual, semi-annual, quarterly, or monthly workfiles.

**Syntax**

Group View: `group_name.dtable(options)`

**Options**

| p           | Print the report table. |
Examples

```
freeze(report) group1.table
```

freezes the default table view of GROUP1 and saves it as a table object named REPORT.

Cross-references

See “Creating and Specifying a Dated Data Table” on page 342 of the User’s Guide for a description of dated data tables and formatting options. Note that most of the options for formatting the table are only available interactively from the window.

<table>
<thead>
<tr>
<th>ec</th>
<th>Var Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Estimate a vector error correction model (VEC).

Syntax

```
Var Method: var_name.ec(trend, n) lag_pairs endog_list [@ exog_list]
```

Specify the order of the VEC by entering one or more pairs of lag intervals, then list the series or groups to be used as endogenous variables. Note that the lag orders are those of the first differences, not the levels. If you are comparing results to another software program, you should be certain that the specifications for the lag orders are comparable.

You may include exogenous variables, such as seasonal dummies, in the VEC by including an “@”-sign followed by the list of series or groups. Do not include an intercept or trend in the VEC specification, these terms should be specified using options, as described below.

You must specify the trend option and the number of cointegrating equations n to use (default is n = 1) in parentheses, separated by a comma. You must choose the trend from the following five alternatives:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>No deterministic trend in the data, and no intercept or trend in the cointegrating equation.</td>
</tr>
<tr>
<td>b</td>
<td>No deterministic trend in the data, and an intercept but no trend in the cointegrating equation.</td>
</tr>
<tr>
<td>c (default)</td>
<td>Linear trend in the data, and an intercept but no trend in the cointegrating equation.</td>
</tr>
<tr>
<td>d</td>
<td>Linear trend in the data, and both an intercept and a trend in the cointegrating equation.</td>
</tr>
<tr>
<td>e</td>
<td>Quadratic trend in the data, and both an intercept and a trend in the cointegrating equation.</td>
</tr>
</tbody>
</table>
edftest—257

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>restrict</td>
<td>Impose restrictions as specified by the <code>append</code> (coint) proc</td>
</tr>
<tr>
<td>m = integer</td>
<td>Maximum number of iterations for restricted estimation (only valid if you choose the restrict option).</td>
</tr>
<tr>
<td>c = scalar</td>
<td>Convergence criterion for restricted estimation. (only valid if you choose the restrict option).</td>
</tr>
</tbody>
</table>

Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>Print the results view.</td>
</tr>
</tbody>
</table>

Examples

```plaintext
var macrol.ec 1 4 m1 gdp tb3
```
declares a var object MACRO1 and estimates a VEC with lagged first differences, three endogenous variables and one cointegrating equation using the default trend option “c”.

```plaintext
var term.ec(b,2) 1 2 4 4 tb1 tb3 tb6 @ d2 d3 d4
```
declares a var object TERM and estimates a VEC with lagged first differences of order 1, 2, 4, three endogenous variables, three exogenous variables, and two cointegrating equations using trend option “b”.

Cross-references


See also `var` (p. 487) and `coint` (p. 224).

Computes goodness-of-fit tests based on the empirical distribution function.

Syntax

```plaintext
Series View: series_name.edftest(options)
```
Options

General Options

**Type = arg**  
*(default = "normal")*  

**p1 = number**  
Specify the value of the first parameter (as it appears in the dialog). If this option is not specified, the first parameter will be estimated.

**p2 = number**  
Specify the value of the second parameter (as it appears in the dialog). If this option is not specified, the first parameter will be estimated.

**p3 = number**  
Specify the value of the third parameter (as it appears in the dialog). If this option is not specified, the first parameter will be estimated.

**p**  
Print test results.

Estimation Options

The following options apply if iterative estimation of parameters is required:

**b**  
Berndt-Hall-Hall-Hausman (BHHH) algorithm. Default is Marquardt.

**m = integer**  
Maximum number of iterations.

**c = number**  
Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.

**showopts / -showopts**  
[Do / do not] display the starting coefficient values and estimation options in the estimation output.

**s**  
Take starting values from the C coefficient vector. By default, EViews uses distribution specific starting values that typically are based on the method of the moments.
Examples

`x.edftest`

uses the default settings to test whether the series X comes from a normal distribution. Both the location and scale parameters are estimated from the data in X.

`freeze(tab1) x.edftest(type=chisq, p1=5)`

tests whether the series x comes from a \(\chi^2\) distribution with 5 degrees of freedom. The output is stored as a table object TAB1.

Cross-references


See also `qqplot` (p. 375).

### endog

Displays a spreadsheet or graph view of the endogenous variables.

Syntax

Object View: `object_name.endog(options)`

Note that in EViews 4, `endog` and `makeendog` are no longer supported for model objects. See instead, `makegroup` (p. 324).

Options

`g`  
Multiple line graphs of the solved endogenous series.

`p`  
Print the table of solved endogenous series.

Examples

`sys1.endog(g,p)`

prints the graphs of the solved endogenous series.

Cross-references

See also `makeendog` (p. 321), `system` (p. 459), `sspace` (p. 444) and `var` (p. 487).
## equation

 Declare an equation object.

### Syntax

Command: `equation eq_name`

Command: `equation eq_name.method(options) specification`

Follow the `equation` keyword with a name and an optional specification. If you wish to enter the specification, you should follow the new equation name with a period, an estimation method, and then the equation specification. Valid estimation methods are `arch`, `binary`, `censored`, `count`, `gmm`, `ls`, `ordered`, and `tsls`.

### Examples

```plaintext
equation cobdoug.ls log(y) c log(k) log(l)
```

declares and estimates an equation object named COBDOUG.

The command

```plaintext
equation ces.ls log(y)=c(1)*log(k^c(2)+l^c(3))
```

declares an equation object named CES containing a nonlinear least squares specification.

```plaintext
equation demand.tsls q c p x @ x p(-1) gov
```

creates an equation object named DEMAND and estimates DEMAND using two-stage least squares with instruments X, lagged P, and GOV.

### Cross-references

See the object reference entry for “Equation” (p. 139) for a summary of the equation object. Chapter 15 of the User’s Guide provides basic information on estimation and equation objects. Additional estimation methods are described in the User’s Guide.

---

<table>
<thead>
<tr>
<th>eqs</th>
<th>Model View</th>
</tr>
</thead>
</table>

View of model organized by equation.

Lists the equations in the model allowing you to access the equation specifications. This view also allows you to identify which equations are entered by text, or by link.
errbar—261

Syntax

Model View: model_name.eqs

Cross-references


See also block (p. 203), text (p. 469), and vars (p. 488) for alternative representations of the model.

errbar || Command | Graph Command | Group View | Matrix View | Rowvector | View | Sym View

Display error bar graph view of object, or change existing graph object type to error bar (if possible).

Sets the graph type to error bar or displays an error bar view of the group. If there are two series in the graph or group, the error bar will show the high and low values in the bar. The optional third series will be plotted as a symbol. When used as a matrix view, the columns of the matrix are used in place of series.

Syntax

Command: errbar(options) arg1 [arg2 arg3 ...]
Graph Proc: graph_name.errbar(options)
Object View: object_name.errbar(options)

Options

Template and printing options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>o=graph_name</td>
<td>Use appearance options from the specified graph.</td>
</tr>
<tr>
<td>t=graph_name</td>
<td>Use appearance options and copy text and shading from the specified graph.</td>
</tr>
<tr>
<td>p</td>
<td>Print the error bar graph.</td>
</tr>
</tbody>
</table>

Panel options

The following options apply when graphing panel structured data.
Examples

The following set of commands

\begin{verbatim}
  group g1 x y
  g1.errbar
\end{verbatim}

displays the error bar view of G1 using the X series as the high value of the bar and the Y series as the low value.

\begin{verbatim}
  group g2 plus2se minus2se estimate
  g2.errbar
\end{verbatim}

displays the error bar view of G2 with the PLUS2SE series as the high value of the bar, the MINUS2SE series as the low value, and ESTIMATE as a symbol.

\begin{verbatim}
  group g1 x y
  freeze(graph1) g1.line
  graph1.errbar
\end{verbatim}

first creates a graph object GRAPH1 containing a line graph of the series in G1, then changes the graph type to an error bar.

Cross-references

See Chapter 14 for details on graph objects and types.

See also \texttt{graph \( (p. 288) \)} for graph declaration and other graph types.
Specifies (or merges) excluded endogenous variables in the active scenario.

Syntax

```
Model Proc:       model_name.exclude(options) ser1(smpl) ser2(smpl) ...
```

Follow the `exclude` keyword with the argument list containing the endogenous variables you wish to exclude from the solution along with an optional sample for exclusion. If a sample is not provided, the variable will be excluded for the entire solution sample.

Options

```
im                 Merge into instead of replace the existing exclude list.
```

Examples

```
mod1.exclude fedfunds goexp("1990:01 1995:02")
```

will create an exclude list containing the variables FEDFUNDS and GOEXP. FEDFUNDS will be excluded for the entire solution sample, while GOEXP will only be excluded for the specified sample.

If you then issue the command

```
mod1.exclude goexp
```

EViews will replace the original exclude list with one containing only GOEXP. To add excludes to an existing list, use the “m” option:

```
mod1.exclude(m) fedfunds
```

The excluded list now contains both GOEXP and FEDFUNDS.

Cross-references

See the discussion in “Specifying Scenarios” on page 773 of the *User’s Guide.*

See also `model` (p. 339), `override` (p. 349) and `solveopt` (p. 439).
exit

Command

Exit from EViews. Closes the EViews application.

You will be prompted to save objects and workfiles which have changed since the last time they were saved to disk. Be sure to save your workfile, if desired, since all changes that you do not save to a disk file will be lost.

Syntax

Command: exit

Cross-references

See also close (p. 222) and save (p. 393).

expand

Command

Expand a workfile.

No longer supported. See the replacement command pagestruct (p. 363).

fetch

Command || Pool Proc

Fetch objects from databases or databank files into the workfile.

fetch reads one or more objects from EViews databases or databank files into the active workfile. The objects are loaded into the workfile under the same name as in the database or with the databank file name.

When used as a pool proc, EViews will first expand the list of series using the pool operator, and then perform the fetch.

If you fetch a series into a workfile with a different frequency, EViews will automatically apply the frequency conversion method attached to the series by setconvert. If the series does not have a conversion method set by setconvert, EViews will use the method set by Options/Date-Frequency in the main menu. You can override the conversion method by explicitly specifying a conversion method option in the fetch command.

Syntax

Command: fetch(options) object_list

Pool Proc: pool_name.fetch(options) pool_ser1 [pool_ser2 pool_ser ...]
The `fetch` command keyword is followed by a list of object names separated by spaces. The default behavior is to fetch the objects from the default database. *(This is a change from versions of EViews prior to EViews 3.x where the default was to fetch from individual databank files).*

You can precede the object name with a database name and the double colon `::` to indicate a specific database source. If you specify the database name as an option in parentheses (see below), all objects without an explicit database prefix will be fetched from the specified database.

You may use wild card characters, `?` (to match a single character) or `*` (to match zero or more characters), in the object name list. All objects with names matching the pattern will be fetched.

You can optionally fetch from individual databank files or search among registered databases. To fetch from individual databank files that are not in the default path, you should include an explicit path. If you have more than one object with the same file name (for example, an equation and a series named CONS), then you should supply the full object file name.

**Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>d=db_name</code></td>
<td>Fetch from specified database.</td>
</tr>
<tr>
<td><code>d</code></td>
<td>Fetch all registered databases in registry order.</td>
</tr>
<tr>
<td><code>i</code></td>
<td>Fetch from individual databank files.</td>
</tr>
</tbody>
</table>

The following options are available for fetch of group objects:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>g=arg</code></td>
<td>Group fetch options: “b” (fetch both group definition and series), “d” (fetch only the series in the group), “l” (fetch only the group definition).</td>
</tr>
</tbody>
</table>

The database specified by the double colon `::` takes precedence over the database specified by the “d =” option.

In addition, there are a number of additional options for controlling automatic frequency conversion when performing a fetch. The following options control the frequency conversion method when copying series and group objects to a workfile, converting from low to high frequency:
The following options control the frequency conversion method when copying series and
group objects to a workfile, converting from high to low frequency:

\[ c = \text{arg} \]

- **Low to high conversion methods:**
  - “r” (constant match average)
  - “d” (constant match sum)
  - “q” (quadratic match average)
  - “t” (quadratic match sum)
  - “i” (linear match last)
  - “c” (cubic match last)

Note that if no method is specified, the global or series specific default conversion method
will be employed.

**Examples**

To fetch M1, GDP, UNEMP from the default database, use:

\[
\text{fetch m1 gdp unemp}
\]

To fetch M1 and GDP from the US1 database and UNEMP from the MACRO database, use
the command:

\[
\text{fetch(d=us1) m1 gdp macro::unemp}
\]

You can fetch all objects with names starting with SP by searching all registered databases
in the search order. The “c = f” option uses the first (nonmissing) observation to convert
the frequency of any matching series with a higher frequency than the workfile frequency:

\[
\text{fetch(d,c=f) sp*}
\]

You can fetch M1 and UNEMP from individual databank files using:
To fetch all objects with names starting with CONS from the two databases USDAT and UKDAT:

```
fetch usdat::cons* ukdat::cons*
```

The command

```
fetch a?income
```

will fetch all series beginning with the letter “a”, followed by any single character, and ending with the string “income”.

**Cross-references**


See also `setconvert` (p. 410), `store` (p. 452), and `copy` (p. 228).

---

### Fill

|-----------|-------------|----------------|-------------|---------|-------------|

**Fill object with specified values.**

`fill` can be used to set values of series, vectors and matrices in programs.

**Syntax**

```
Object Proc: object_name.\text{\texttt{fill}}(options) n1[, n2, n3 …]
```

Follow the keyword with a list of values to place in the specified object. *Each value should be separated by a comma.* By default, series `fill` ignores the current sample and fills the series from the beginning of the workfile range. You may specify sample information using options.

Running out of values before the object is completely filled is not an error; the remaining cells or observations will be unaffected. If, however, you list more values than the object can hold, EViews will not modify any of the observations and will return an error message.
Options

Options for series fill

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Loop repeatedly over the list of values as many times as it takes to fill the series.</td>
</tr>
<tr>
<td>o = [date, integer]</td>
<td>Set starting date or observation from which to start filling the series. Default is the beginning of the workfile range.</td>
</tr>
<tr>
<td>s</td>
<td>Fill the series only for the current workfile sample. The “s” option overrides the “o” option.</td>
</tr>
<tr>
<td>s = sample_name</td>
<td>fill the series only for the specified subsample. The “s” option overrides the “o” option.</td>
</tr>
</tbody>
</table>

Options for coef/vector/matrix fill

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Loop repeatedly over the list of values as many times as it takes to fill the object.</td>
</tr>
<tr>
<td>o = integer (default = 1)</td>
<td>Fill the object from the specified element. Default is the first element.</td>
</tr>
<tr>
<td>b = arg (default = “c”)</td>
<td>Matrix fill order: “c” (fill the matrix by column), “r” (fill the matrix by row).</td>
</tr>
</tbody>
</table>

Examples

To generate a series D70 that takes the value 1, 2, and 3 for all observations from 1970:1:

```plaintext
series d70=0
d70.fill(o=1970:1,l) 1,2,3
```

Note that the last argument in the fill command above is the letter “l”. The next three lines generate a dummy series D70S that takes the value one and two for observations from 1970:1 to 1979:4:

```plaintext
series d70s=0
smpl 1970:1 1979:4
d70s.fill(s,l) 1,2
smpl @all
```

Assuming a quarterly workfile, the following generates a dummy variable for observations in either the third and fourth quarter:

```plaintext
series d34
d34.fill(l) 0, 0, 1, 1
```
Note that this series could more easily be generated using \texttt{@seas}.

The following example declares a four element coefficient vector \texttt{MC}, initially filled with zeros. The second line fills \texttt{MC} with the specified values and the third line replaces from row 3 to the last row with \(-1\).

```plaintext
  coef(4) mc
  mc.fill 0.1, 0.2, 0.5, 0.5
  mc.fill(o=3,l) -1
```

The commands

```plaintext
  matrix(2,2) m1
  matrix(2,2) m2
  m1.fill 1, 0, 1, 2
  m2.fill(b=r) 1, 0, 1, 2
```

create the matrices

\[
  m_1 = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}, \quad m_2 = \begin{bmatrix} 1 & 0 \\ 1 & 2 \end{bmatrix}
\] (B.1)

Cross-references


---

**\texttt{fiml}**

<table>
<thead>
<tr>
<th>System Method</th>
</tr>
</thead>
</table>

Estimation by full information maximum likelihood.

\texttt{fiml} estimates a system of equations by full information maximum likelihood (assuming a multivariate normal distribution).

**Syntax**

```plaintext
  System Method: system_name.fiml(options)
```

**Options**

- \texttt{i} \hspace{1cm} Iterate simultaneously over the covariance matrix and coefficient vector.
- \texttt{s (default)} \hspace{1cm} Iterate sequentially over the covariance matrix and coefficient vector.
### Examples

```
sys1.fiml
```

estimates SYS1 by FIML using the default settings. The command

```
sys1.fiml(d, s)
```

sequentially iterates over the coefficients and the covariance matrix.

### Cross-references


### fit

| Command || Equation Proc |
|----------|------------------|
| fit      |                  |

Static forecasts. Computes fitted values from an estimated equation.

When the regressor contains lagged dependent values or ARMA terms, `fit` uses the actual values of the dependent variable instead of the lagged fitted values. You may instruct `fit` to compare the forecasted data to actual data, and to compute forecast summary statistics.

Not available for equations estimated using ordered methods—use `model` instead.
Syntax

Command: \texttt{fit(options) yhat y\_se}
Equation Proc: \texttt{eq\_name.fit(options) yhat y\_se}
ARCH Proc: \texttt{eq\_name.fit(options) yhat y\_se y\_var}

Following the \texttt{fit} keyword you should type a name for the forecast series and, optionally, a name for the series containing the standard errors and, for ARCH specifications, a name for the conditional variance series.

Forecast standard errors are currently not available for binary, censored, and count models.

Options

\begin{verbatim}
\begin{tabular}{ll}
g & Graph the fitted values together with the ±2 standard error bands. \\
e & Produce the forecast evaluation table. \\
s & Ignore ARMA terms and use only the structural part of the equation to compute the fitted values. \\
i & Compute the fitted values of the index. Only for binary, censored and count models. \\
d & Forecast the dependent variable without normalizing the formula. \\
f = arg \\
\texttt{(default ="actual")} & Out-of-fit-sample fill behavior: “actual” (fill observations outside the fit sample with actual values for the fitted variable), “na” (fill observations outside the fit sample with missing values). \\
\end{tabular}
\end{verbatim}

Examples

\texttt{equation eql.ls \_cons \_c \_cons(-1) \_inc \_inc(-1)}
\texttt{eql.fit \_c\_hat \_c\_se}
\texttt{genr \_c\_up=\_c\_hat+2*\_c\_se}
\texttt{genr \_c\_low=\_c\_hat-2*\_c\_se}
\texttt{line \_cons \_c\_up \_c\_low}

The first line estimates a linear regression of CONS on a constant, CONS lagged once, INC, and INC lagged once. The second line stores the static forecasts and their standard errors as \texttt{C\_HAT} and \texttt{C\_SE}. The third and fourth lines compute the upper and lower bounds of the 95\% forecast interval (assuming a normal distribution). The fifth line plots the actual series together with the 95\% forecast interval.
equation eq2.binary(d=1) y c wage edu 
eq2.fit yf 
eq2.fit(I) xbeta 
genr yhat=1-@clogit(-xbeta)

The first line estimates a logit of Y on a constant, WAGE, and EDU. The second line computes the fitted probabilities, and the third line computes the fitted values of the index. The fourth line computes the probabilities from the fitted index using the cumulative distribution function of the logistic distribution. Note that YF and YHAT should be identical.

Note that you cannot fit values from an ordered model. You must instead solve the values from a model. The following lines generate fitted probabilities from an ordered model:

\[
equation eq3.ordered y c x z 
eq3.model(oprob1) 
solve oprob1
\]

The first line estimates an ordered probit of Y on a constant, X, and Z. The second line makes a model from the estimated equation with a name OPROB1. The third line solves the model and computes the fitted probabilities that each observation falls in each category.

Cross-references


To perform dynamic forecasting, use forecast (p. 272). See model (p. 339) and solve (p. 438) for forecasting from systems of equations or ordered equations.

| forecast | Command || Equation Proc | Sspace Proc |

Dynamic forecast.

Computes (n-period ahead) dynamic forecasts of an estimated equation or forecasts of the signals and states for an estimated state space.

forecast computes the forecast for all observations in a specified sample. In some settings, you may instruct forecast to compare the forecasted data to actual data, and to compute summary statistics.
Syntax

Command: \texttt{forecast(options) yhat [y\_se]}

Equation Proc: \texttt{eq\_name.forecast(options) yhat [y\_se]}

ARCH Proc: \texttt{eq\_name.forecast(options) yhat [y\_se y\_var]}

Sspace Proc: \texttt{ss\_name.forecast(options) keyword1 names1 [keyword2 names2] [keyword3 names3] ...}

When used with an equation, you should type a name for the forecast series and, optionally, a name for the series containing the standard errors and, for ARCH specifications, a name for the conditional variance series. Forecast standard errors are currently not available for binary or censored models. \texttt{forecast} is not available for models estimated using ordered methods.

When used with a sspace, you should enter a type-keyword followed by a list of names for the target series or a wildcard expression, and if desired, additional type-keyword/target pairs. The following are valid keywords: \texttt{"@STATE"}, \texttt{"@STATESE"}, \texttt{"@SIGNAL"}, \texttt{"@SIGNALSE"}. The first two instruct EViews to forecast the state series and the values of the state standard error series. The latter two instruct EViews to forecast the signal series and the values of the signal standard error series.

If a list is used to identify the targets in sspace forecasting, the number of target series must match the number of names implied by the keyword. Note that wildcard expressions may not be used for forecasting signal variables that contain expressions. The \texttt{"*"} wildcard expression may not be used for forecasting signal variables since this would overwrite the original data.

Options

\textit{Options for Equation forecasting}

- \texttt{d} In models with implicit dependent variables, forecast the entire expression rather than the normalized variable.
- \texttt{e} Produce the forecast evaluation table.
- \texttt{g} Graph the forecasts together with the $\pm 2$ standard error bands.
Options for \textit{Sspace} forecasting

\begin{itemize}
  \item \texttt{i = arg (default = “o”)} \quad \text{State initialization options: “o” (one-step), “e” (diffuse), “u” (user-specified), “s” (smoothed).}
  \item \texttt{m = arg (default = “d”)} \quad \text{Basic forecasting method: “n” (n-step ahead forecasting), “s” (smoothed forecasting), “d” (dynamic forecasting).}
  \item \texttt{mprior = vector\_name} \quad \text{Name of state initialization (used if option “i=u”).}
  \item \texttt{n = arg (default = 1)} \quad \text{Number of n-step forecast periods (only relevant if n-step forecasting is specified using the \texttt{method} option).}
  \item \texttt{vprior = sym\_name} \quad \text{Name of state covariance initialization (used if option “i=u”).}
\end{itemize}

Examples

The following lines

\begin{verbatim}
smpl 1970:1 1990:4
equation eql.ls con c con(-1) inc
smpl 1991:1 1995:4
eql.fit con_s
eql.forecast con_d
plot con_s con_d
\end{verbatim}

estimate a linear regression over the period 1970:1–1990:4, compute static and dynamic forecasts for the period 1991:1–1995:4, and plots the two forecasts in the same graph

\begin{verbatim}
equation eql.ls m1 gdp ar(1) ma(1)
eql.forecast m1_bj bj_se
eql.forecast(s) m1_s s_se
\end{verbatim}
plot bj_se s_se

Estimates an ARMA(1,1) model, computes the forecasts and standard errors with and without the ARMA terms, and plots the two forecast standard errors.

The following command performs n-step forecasting of the signals and states from the sspace object, SS1:

```eviews
ss1.forecast(method=n,n=4) @state * @signal y1f y2f
```

Here we save the state forecasts in the names specified in the sspace object, and we save the two signal forecasts in the series Y1F and Y2F.

**Cross-references**

For more information on equation forecasting in EViews, see Chapter 18 of the *User's Guide*. Details on forecasting for specialized techniques are provided in the corresponding chapters of the *User's Guide*. For additional discussion of wildcards, see Appendix C, “Wildcards”, on page 907 of the *User's Guide*.

To perform static forecasting with equation objects see *fit* (p. 270). For multiple equation forecasting, see *model* (p. 339), and *solve* (p. 438).

<table>
<thead>
<tr>
<th>freeze</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creates graph, table, or text objects from a view.</td>
<td></td>
</tr>
</tbody>
</table>

**Syntax**

Command: `freeze(name) object_name.view_command`

If you follow the keyword `freeze` with an object name but no view of the object, `freeze` will use the default view for the object. You can provide a name for the object containing the frozen view in parentheses.

**Examples**

The command

```eviews
freeze gdp.uroot(4,t)
```

creates an untitled table that contains the results of the unit root test of the series GDP.

```eviews
group rates tbl tb3 tb6
freeze(gral) rates.line(m)
show gral.align(2,1,1)
```
freezes a graph named GRA1 that contains multiple line graphs of the series in the group RATES, realigns the individual graphs, and displays the resulting graph.

\[ \text{freeze}(\text{mygra}) \text{ gra1 gra3 gra4} \]
\[ \text{show mygra.align}(2,1,1) \]

creates a graph object named MYGRA that combines three graph objects GRA1, GRA2, and GRA3, and displays MYGRA in two columns.

Cross-references

See “Object Commands” on page 6.

See also Chapter 4, “Object Basics”, on page 59 of the User’s Guide for further discussion of objects and views of objects. Freezing views is described in “Creating Graphs” on page 387 of the User’s Guide.

freq

<table>
<thead>
<tr>
<th>Alpha View</th>
<th>Group View</th>
<th>Series View</th>
</tr>
</thead>
</table>

Compute frequency tables.

When used as a series view (or for a group containing a single series), \texttt{freq} performs a one-way frequency tabulation. The options allow you to control binning (grouping) of observations.

When used with a group containing multiple series, \texttt{freq} produces an N-way frequency tabulation for all of the series in the group.

Syntax

Object View: \[ \text{object_name.freq}(\text{options}) \]

Options

Options common to both one-way and N-way frequency tables

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{dropna} (default)</td>
<td>[Drop/Keep] NA as a category.</td>
</tr>
<tr>
<td>\texttt{v = integer} (default = 100)</td>
<td>Make bins if the number of distinct values or categories exceeds the specified number.</td>
</tr>
<tr>
<td>\texttt{nov}</td>
<td>Do not make bins on the basis of number of distinct values; ignored if you set \texttt{v = integer}.</td>
</tr>
<tr>
<td>\texttt{a = number} (default = 2)</td>
<td>Make bins if average count per distinct value is less than the specified number.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>noa</td>
<td>Do not make bins on the basis of average count; ignored if you set “a = number.”</td>
</tr>
<tr>
<td>b = integer (default = 5)</td>
<td>Maximum number of categories to bin into.</td>
</tr>
<tr>
<td>n, obs, count (default)</td>
<td>Display frequency counts.</td>
</tr>
<tr>
<td>ncount</td>
<td>Do not display frequency counts.</td>
</tr>
<tr>
<td>p</td>
<td>Print the table.</td>
</tr>
</tbody>
</table>

**Options for one-way tables**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>total (default) / nototal</td>
<td>(Display/Do not display) totals.</td>
</tr>
<tr>
<td>pct (default) / npct</td>
<td>(Display/Do not display) percent frequencies.</td>
</tr>
<tr>
<td>cum (default) / nocum</td>
<td>(Display/Do not) display cumulative frequency counts/percentages.</td>
</tr>
</tbody>
</table>

**Options for N-way tables**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table (default)</td>
<td>Display in table mode.</td>
</tr>
<tr>
<td>list</td>
<td>Display in list mode.</td>
</tr>
<tr>
<td>rowm (default) / norowm</td>
<td>(Display/Do not display) row marginals.</td>
</tr>
<tr>
<td>colm (default) / nocolm</td>
<td>(Display/Do not display) column marginals.</td>
</tr>
<tr>
<td>tabm (default) / notabm</td>
<td>(Display/Do not display) table marginals—only for more than two series.</td>
</tr>
<tr>
<td>subm (default) / nosubm</td>
<td>(Display/Do not display) sub marginals—only for “l” option with more than two series.</td>
</tr>
<tr>
<td>full (default) / sparse</td>
<td>(Full/Sparse) tabulation in list display.</td>
</tr>
<tr>
<td>totpct / nototpct (default)</td>
<td>(Display/Do not display) percentages of total observations.</td>
</tr>
<tr>
<td>tabpct / notabpct (default)</td>
<td>(Display/Do not display) percentages of table observations—only for more than two series.</td>
</tr>
</tbody>
</table>
Examples

hrs.freq(nov,noa)

tabulates each value (no binning) of HRS in ascending order with counts, percentages, and cumulatives.

inc.freq(v=20,b=10,noa)

tabulates INC excluding NAs. The observations will be binned if INC has more than 20 distinct values; EViews will create at most 10 equal width bins. The number of bins may be smaller than specified.

group labor lwage gender race
labor.freq(v=10,norowm,nocolm)

displays tables of LWAGE against GENDER for each bin/value of RACE.

Cross-references

See Chapters 11 and 12 of the User’s Guide for a discussion of frequency tables.

| frml | Command || Object Declaration (Alpha) | Object Declaration (Series) |
|------|----------|----------------------------------|-----------------------------|

Declare a series object with a formula for auto-updating, or specify a formula for an existing series.

Syntax

Command: `frml series_name = series_expression`

Command: `frml series_name = @clear`
Follow the `frml` keyword with a name for the object, and an assignment. The special keyword “@CLEAR” is used to return the auto-updating series to an ordinary numeric or alpha series.

**Examples**

To define an auto-updating numeric or alpha series, you must use the `frml` keyword prior to entering the assignment statement. The following example creates a series named `LOW` that uses a formula to compute values. The auto-updating series takes value 1 if either `INC` is less than or equal to 5000 or `EDU` is less than 13 and 0 otherwise.

```
frml low = inc<=5000 or edu<13
```

and will be re-evaluated whenever `INC` or `EDU` change.

If `FIRST_NAME` and `LAST_NAME` are alpha series, then the formula declaration

```
frml full_name = first_name + " " + last_name
```

creates an auto-updating alpha series `FULL_NAME`.

You may apply a `frml` to an existing series. The commands

```
series z
frml z = (x+y)/2
```

makes the previously created series `Z` an auto-updating series containing the average of series `X` and `Y`. Note that once a series is defined to be auto-updating, it may not be modified directly. Here, you may not edit `Z`, nor may you generate values into the series.

Note that the commands

```
series z
z = (x+y)/2
```

while similar, produce quite different results, since the absence of the `frml` keyword in the second line means that EViews will generate fixed values in the series instead of defining a formula that will be used to compute the series values. In this latter case, the series `Z` may be modified.

One particularly useful feature of auto-updating series is the ability to reference series in databases. The command:

```
frml gdp = usdata::gdp
```

creates a series called `GDP` that gets its values from the series `GDP` in the database `USDATA`. Similarly
frml lgdp = log(usdata::gdp)

creates an auto-updating series that is the log of the values of GDP in the database USDATA.

To turn off auto-updating for a series or alpha, you should use the special expression “@CLEAR” in your frml assignment. The command

frml z = @clear

sets the series to numeric or alpha value format, freezing the contents of the series at the current values.

Cross-references
See “Auto-Updating Series” on page 131 of the User’s Guide.

<table>
<thead>
<tr>
<th>garch</th>
<th>Equation View</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditional standard deviation graph of (G)ARCH equation.</td>
<td></td>
</tr>
<tr>
<td>Displays the conditional standard deviation graph of an equation estimated by ARCH.</td>
<td></td>
</tr>
</tbody>
</table>

Syntax
Equation View: eq_name.garch(options)

Options

| p | Print the graph |

Examples

equation eq1.arch sp500 c
eq1.garch

estimates a GARCH(1,1) model and displays the estimated conditional standard deviation graph.

Cross-references
ARCH estimation is described in Chapter 20 of the User’s Guide.

See also arch (p. 187), makegarch (p. 322).
Generate series.

Generate series or alphas. This procedure also allows you to generate multiple series using the cross-section identifiers in a pool.

Syntax

Command: `genr ser_name = expression`  
Pool Proc: `pool_name.genr ser_name = expression`

For Pool series generation, you may use the cross section identifier “?” in the series name and/or in the expression on the right-hand side.

Examples

The commands

```plaintext
pool pool1
pool1.add 1 2 3
pool1.genr y? = x? - @mean(x?)
```

are equivalent to generating separate series for each cross-section:

```plaintext
series y1 = x1 - @mean(x1)
series y2 = x2 - @mean(x2)
series y3 = x3 - @mean(x3)
```

Similarly,

```plaintext
pool pool2
pool2.add us uk can
pool2.genr y_? = log(x_?) - log(x_us)
```

generates three series Y_US, Y_UK, Y_CAN that are the log differences from X_US. Note that Y_US=0.

The pool `genr` command simply loops across the cross-section identifiers, performing the appropriate substitution. Thus, the command

```plaintext
pool2.genr z=y_?
```

is equivalent to entering

```plaintext
series z=y_us
```
series z=y_uk
series z=y_can

so that the ordinary series Z will contain Y_CAN, the last series associated with the “Y?”.

Cross-references

See Chapter 27 of the User’s Guide for a discussion of the computation of pools, and a description of individual and balanced samples.

See series (p. 405) for a discussion of the expressions allowed in genr.

gmm Command | Equation Method | System Method

Estimation by generalized method of moments (GMM).

The equation or system object must be specified with a list of instruments.

Syntax

Command: \texttt{gmm(options) y \, x1 \, [x2 \, x3 \, \ldots] \, @ \, z1 \, [z2 \, z3 \, \ldots]}
\texttt{gmm(options) \, \text{specification} \, @ \, z1 \, [z2 \, z3 \, \ldots]}

Equation Method: \texttt{eq\_name.gmm(options) \, y \, x1 \, [x2 \, x3 \, \ldots] \, @ \, z1 \, [z2 \, z3 \, \ldots]}
\texttt{eq\_name.gmm(options) \, \text{specification} \, @ \, z1 \, [z2 \, z3 \, \ldots]}

System Method: \texttt{system\_name.gmm(options)}

To use \texttt{gmm} as a command or equation method, follow the name of the dependent variable by a list of regressors, followed by the “@” symbol, and a list of instrumental variables which are orthogonal to the residuals. Alternatively, you can specify an expression using coefficients, an “@” symbol, and a list of instrumental variables. There must be at least as many instrumental variables as there are coefficients to be estimated.

To specify a dynamic instrument, you should tag the instrument using “@DYN”, as in “@DYN(X)”. By default, EViews will use a set of period specific instruments corresponding to lags from -2 to “-infinity”. You may also specify a restricted lag range using arguments in the “@DYN” tag. For example, to use lags from -5 to “-infinity” you may enter “@DYN(X, -5)”; to specify lags from -2 to -6, use “@DYN(X, -2, -6)” or “@DYN(X, -6, -2)”. 
Options

**General Options**

- **m = integer**
  - Maximum number of iterations.

- **c = number**
  - Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.

- **l = number**
  - Set maximum number of iterations on the first-stage iteration to get the one-step weighting matrix.

- **showopts / -showopts**
  - [Do / do not] display the starting coefficient values and estimation options in the estimation output.

- **deriv = keyword**
  - Set derivative methods. The argument `keyword` should be a one- or two-letter string. The first letter should either be “f” or “a” corresponding to fast or accurate numeric derivatives (if used). The second letter should be either “n” (always use numeric) or “a” (use analytic if possible). If omitted, EViews will use the global defaults.

- **p**
  - Print results.

**Additional Options for Non-Panel Equation and System estimation**

- **w**
  - Use White’s diagonal weighting matrix (for cross section data).

- **b = arg (default = “nw”)**
  - Specify the bandwidth: “nw” (Newey-West fixed bandwidth based on the number of observations), “number” (user specified bandwidth), “v” (Newey-West automatic variable bandwidth selection), “a” (Andrews automatic selection).

- **q**
  - Use the quadratic kernel. Default is to use the Bartlett kernel.

- **n**
  - Prewhiten by a first order VAR before estimation.

- **i**
  - Iterate simultaneously over the weighting matrix and the coefficient vector.

- **s**
  - Iterate sequentially over the weighting matrix and coefficient vector.
### Additional Options for Panel Equation estimation

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>o</strong></td>
<td>(default) Iterate only on the coefficient vector with one step of the weighting matrix.</td>
</tr>
<tr>
<td><strong>c</strong></td>
<td>One step (iteration) of the coefficient vector following one step of the weighting matrix.</td>
</tr>
<tr>
<td><strong>e</strong></td>
<td>TSLS estimates with GMM standard errors.</td>
</tr>
</tbody>
</table>

**cx = arg** Cross-section effects method: (default) none, fixed effects estimation (“cx = f”), first-difference estimation (“cx = fd”), orthogonal deviation estimation (“cx = od”).

**per = arg** Period effects method: (default) none, fixed effects estimation (“cx = f”).

**levelper** Period dummies always specified in levels (even if one of the transformation methods is used, “cx = fd” or “cx = od”).

**wgt = arg** GLS weighting: (default) none, cross-section system weights (“wgt = cxsur”), period system weights (“wgt = persur”), cross-section diagonal weighs (“wgt = cxdiag”), period diagonal weights (“wgt = perdiag”).


By default, uses the identity matrix unless estimated with first difference transformation (“cx = first”), in which case, uses difference weighting matrix (Arellano-Bond 1-step).
Note that some options are not available with some specifications or in combination with other options.

**Examples**

In a non-panel workfile, we may estimate equations using the standard GMM options. The specification

\[ \text{eq.gmm(w) } y \ c \ f \ k \ @ \ c \ z_1 \ z_2 \ z_3 \]

estimates the linear specification using a White diagonal weighting matrix (one-step, with no iteration), while
estimates the nonlinear model using two-stage least squares (instrumental variables) with GMM standard errors computed using Newey-West automatic bandwidth selected weights.

For system estimation, the command

\[ \text{sys1.gmm}(b=a, \quad q, \quad i) \]

estimates the system SYS1 by GMM with a quadratic kernel, Andrews automatic bandwidth selection, and iterates until convergence.

When working with a workfile that has a panel structure, you may use the panel equation estimation options. The command

\[ \text{eq.gmm}(cx=fd, \quad per=f) \quad dj \quad dj(-1) \quad @ \quad @\text{dyn}(dj) \]

estimates an Arellano-Bond “one-step” with differencing of the dependent variable DJ, period fixed effects, and dynamic instruments constructed using DJ with observation specific lags from period 1 to \( t-2 \).

To perform the “two-step” version of this estimator, you may use

\[ \text{eq.gmm}(cx=fd, \quad per=f, \quad gmm=perwhite, \quad iter=oneb) \quad dj \quad dj(-1) \quad @ \quad @\text{dyn}(dj) \]

where the combination of the options “gmm=perwhite” and (the default) “iter=oneb” instructs EViews to estimate the model with the difference weights, use the estimates to form period covariance GMM weights, and then re-estimates the model.

You may iterate the GMM weights to convergence using

\[ \text{eq.gmm}(cx=fd, \quad per=f, \quad gmm=perwhite, \quad iter=seq) \quad dj \quad dj(-1) \quad @ \quad @\text{dyn}(dj) \]

Alternatively,

\[ \text{eq.gmm}(cx=od, \quad gmm=perwhite, \quad iter=oneb) \quad dj \quad dj(-1) \quad x \quad y \quad @ \quad @\text{dyn}(dj,-2,-6) \quad x(-1) \quad y(-1) \]

estimates an Arellano-Bond “two-step” equation using orthogonal deviations of the dependent variable, dynamic instruments constructed from DJ from period \( t-6 \) to \( t-2 \), and ordinary instruments \( X(-1) \) and \( Y(-1) \).

Cross-references

Gradients of the objective function.

Displays the gradients of the objective function (where available) for objects containing an estimated equation or equations.

The (default) summary form shows the value of the gradient vector at the estimated parameter values (if valid estimates exist) or at the current coefficient values. This latter usage makes grads useful in examining the behavior of the objective function at starting values. The tabular form shows a spreadsheet view of the gradients for each observation. The graphical form shows this information in a multiple line graph.

Syntax

Object View: object_name.grads(options)

Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>Display multiple graph showing the gradients of the objective function with respect to the coefficients evaluated at each observation (not available for systems).</td>
</tr>
<tr>
<td>t (default)</td>
<td>Display spreadsheet view of the values of the gradients of the objective function with respect to the coefficients evaluated at each observation (not available for systems).</td>
</tr>
<tr>
<td>p</td>
<td>Print results.</td>
</tr>
</tbody>
</table>

Examples

To show a summary view of the gradients:

eq1.grads

To display and print the table view

eq1.grads(t, p)

Cross-references

See also derivs (p. 249), makederivs (p. 320), and makegrads (p. 322).
Create named graph object containing the results of a graph command, or created when merging multiple graphs into a single graph.

Syntax

Command: `graph graph_name.graph_command(options) arg1 [arg2 arg3 ...]`
Command: `graph graph_name.merge graph1 graph2 [graph3 ...]`

Follow the keyword with a name for the graph, a period, and then a statement used to create a graph. There are two distinct uses of the command:

- In the first form of the command, you create a graph using one of the graph commands, and then name the object using the specified name. The portion of the command given by

  \[\text{graph}\_\text{command}(\text{options}) \ arg1 [\arg2 \arg3 \ldots]\]

should follow the form of one of the standard EViews graph commands

<table>
<thead>
<tr>
<th>Graph Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>Area graph (area (p. 191)).</td>
</tr>
<tr>
<td>bar</td>
<td>Bar graph (bar (p. 199)).</td>
</tr>
<tr>
<td>errbar</td>
<td>Error bar graph (errbar (p. 261)).</td>
</tr>
<tr>
<td>hilo</td>
<td>High-low(open-close) graph (hilo (p. 292)).</td>
</tr>
<tr>
<td>line</td>
<td>Line graph (line (p. 306)).</td>
</tr>
<tr>
<td>pie</td>
<td>Pie graph (pie (p. 370)).</td>
</tr>
<tr>
<td>scat</td>
<td>Scatterplot—same as XY, but lines are initially turned off, symbols turned on, and a 3 × 3 frame is used (scat (p. 399)).</td>
</tr>
<tr>
<td>spike</td>
<td>Spike graph (spike (p. 442)).</td>
</tr>
</tbody>
</table>
In the second form of the command, you instruct EViews to merge the listed graphs into a single graph, and then name the graph object using the specified graph name.

**Options**

- `p` Print the graph (for use when specified with a graph command).

Additional options will depend on the type of graph chosen. See the entry for each graph type for a list of the available options (e.g. see bar (p. 199) for details on bar graphs).

**Examples**

The graph declaration

```
graph gra1.line(s, p) gdp ml inf
```

creates and prints a stacked line graph object named GRA1. This command is equivalent to running the command

```
line(s, p) gdp ml inf
```

freezing the view, and naming the graph GRA1.

```
graph mygra.merge gr_line gr_scat gr_pie
```

creates a multiple graph object named MYGRA that merges three graph objects named GR_LINE, GR SCAT, and GR PIE.

**Cross-references**

The graph object is described in greater detail in “Graph” (p. 143). See Chapter 14 of the User’s Guide for a general discussion of graphs.

See also freeze (p. 275) and merge (p. 336).
Declare a group object containing a group of series.

Syntax

Command:  `group group_name ser1 ser2 [ser3 ...]`

Follow the group name with a list of series to be included in the group.

Examples

```plaintext
group g1 gdp cpi inv
```

```plaintext
group g1 tb3 m1 gov
```

```plaintext
g1.add gdp cpi
```

The first line creates a group named G1 that contains three series GDP, CPI, and INV. The second line redeclares group G1 to contain the three series TB3, M1, and GOV. The third line adds two series GDP and CPI to group G1 to make a total of five series. See `add` (p. 179).

```plaintext
group rhs d1 d2 d3 d4 gdp(0 to -4)
```

```plaintext
ls cons rhs
```

```plaintext
ls cons c rhs(6)
```

The first line creates a group named RHS that contains nine series. The second line runs a linear regression of CONS on the nine series in RHS. The third line runs a linear regression of CONS on C and only the sixth series GDP(-1) of RHS.

Cross-references

See “Group” on page 145 for a complete description of the group object. See also Chapter 12 of the User’s Guide for additional discussion.

See also `add` (p. 179) and `drop` (p. 255).

Convert an entire Haver Analytics Database into an EViews database.

Syntax

Command:  `hconvert haver_path db_name`

**You must have a Haver Analytics database installed on your computer to use this feature.**
You must also create an EViews database to store the converted Haver data before you use this command.

Be aware that this command may be very time-consuming.

Follow the command by a full path name to the Haver database and the name of an existing EViews database to store the converted Haver data. You can include a path name to the EViews database not in the default path.

**Examples**

```
  dbcreate hdata
  hconvert d:\haver\haver hdata
```

The first line creates a new (empty) database named HDATA in the default directory. The second line converts all the data in the Haver database and stores it in the HDATA database.

**Cross-references**


See also dbcreate (p. 240), db (p. 239), hfetch (p. 291) and hlabel (p. 294).

### hfet<href\>ch

**Command**

Fetching a series from a Haver Analytics database into a workfile.

hfet<href\>ch reads one or more series from a Haver Analytics Database into the active workfile. You must have a Haver Analytics database installed on your computer to use this feature.

**Syntax**

```
  Command: hfet<href\>ch(database_name) series_name
```

hfet<href\>ch, if issued alone on a command line, will bring up a Haver dialog box which has fields for entering both the database name and the series names to be fetched. If you provide the database name (including the full path) in parentheses after the hfet<href\>ch command, EViews will read the database and copy the series requested into the current workfile. It will also display information about the series on the statusline. The database name is optional if a default database has been specified.

hfet<href\>ch can read multiple series with a single command. List the series names, each separated by a space.
Examples

```
hfetch(c:\data\us1) pop gdp xyz
```

reads the series POP, GDP, and XYZ from the US1 database into the current active workfile, performing frequency conversions if necessary.

Cross-references

Additional information on EViews frequency conversion is provided in “Frequency Conversion” on page 96 of the User’s Guide. See also Chapter 10 of the User’s Guide for a discussion of EViews and Haver databases.

See also `dbcreate` (p. 240), `db` (p. 239), `hconvert` (p. 290) and `hlabel` (p. 294).

| hilo            | Command || Graph Command | Group View | Matrix View | Sym View |
|-----------------|-----------|------------------------|------------|-------------|----------|

Display high-low[-open-close] graph view of object, or change existing graph object type to high-low[-open-close] (if possible).

Syntax

Command: `hilo(options) high_ser low_ser [close_ser]`

`hilo(options) high_ser low_ser open_ser close_ser`

`hilo(options) arg`

Graph Proc: `graph_name.hilo(options)`

Object View: `object_name.hilo(options)`

For a high-low, or a high-low-close graph, follow the command name with the name of the high series, followed by the low series, and an optional close series. If four series names are provided, EViews will use them in the following order: high-low-open-close.

When used with a matrix or group or an existing graph, EViews will use the first series as the high series, the second series as the low series, and if present, the third series as the close. If there are four or more series, EViews will use them in the following order: high-low-open-close. When used as a matrix view, the columns of the matrix are used in place of the series.

Note that if you wish to display a high-low-open graph, you should use an “NA”-series for the close values.
Options

Template and printing options

\[ o = graph\_name \]
Use appearance options from the specified graph.

\[ t = graph\_name \]
Use appearance options and copy text and shading from the specified graph.

\[ p \]
Print the bar graph.

Panel options

The following options apply when graphing panel structured data.

\[ panel = arg \]
(default taken from global settings)
Panel data display: “stack” (stack the cross-sections), “individual” or “1” (separate graph for each cross-section), “combine” or “c” (combine each cross-section in single graph; one time axis), “mean” (plot means across cross-sections), “mean1se” (plot mean and +/- 1 standard deviation summaries), “mean2sd” (plot mean and +/- 2 s.d. summaries), “mean3sd” (plot mean and +/- 3 s.d. summaries), “median” (plot median across cross-sections), “med25” (plot median and +/- .25 quantiles), “med10” (plot median and +/- .10 quantiles), “med05” (plot median and +/- .05 quantiles), “med025” (plot median and +/- .025 quantiles), “med005” (plot median and +/- .005 quantiles), “median”, “max”, “min”.

Examples

\[ hilo\ mshigh\ mslow \]
displays a high-low graph using the series MSHIGH and MSLOW.

The command

\[ hilo(p)\ mshigh\ mslow\ msopen\ msclose \]
plots and prints the high-low-open-close graph of the four series.

\[ group\ stockprice\ mshigh\ mslow\ msclose\ stockprice.hilo \]
displays the high-low-close view of the group STOCKPRICE.

The command
hilo NA NA msopen msclose

displays an open-close graph.

Cross-references

See also graph (p. 288) for graph declaration and other graph types.

| hist          | Command || Equation View | Series View |
|--------------|-----------|---------------------|-------------|

Histogram and descriptive statistics of a series.

When used as a command or a series view, hist computes descriptive statistics and displays a histogram for the series. When used as an equation view, hist displays a histogram and descriptive statistics of the residual series.

Syntax

Command: hist(options) series_name
Object View: object_name.hist(options)

Options

p Print the histogram.

Examples

lwage.hist

Displays the histogram and descriptive statistics of LWAGE.

Cross-references

See “Histogram and Stats” on page 286 of the User’s Guide for a discussion of the descriptive statistics reported in the histogram view.

<table>
<thead>
<tr>
<th>hlabel</th>
<th>Command</th>
</tr>
</thead>
</table>

Display a Haver Analytics database series description.

hlabel reads the description of a series from a Haver Analytics Database and displays it on the status line at the bottom of the EViews window. Use this command to verify the contents of a Haver database series name.
You must have a Haver Analytics database installed on your computer to use this feature.

Syntax

Command: \texttt{hlabel(database\_name)\ series\_name}

\texttt{hlabel}, if issued alone on a command line, will bring up a Haver dialog box which has fields for entering both the database name and the series names to be examined. If you provide the database name in parentheses after the \texttt{hlabel} command, EViews will read the database and find the key information about the series in question, such as the start date, end date, frequency, and description. This information is displayed in the status line at the bottom of the EViews window. Note that the \texttt{database\_name} should refer to the full path to the Haver database but need not be specified if a default database has been specified in \texttt{HAVERDIR.INI}.

If several series names are placed on the line, \texttt{hlabel} will gather the information about each of them, but the information display may scroll by so fast that only the last will be visible.

Examples

\begin{verbatim}
  hlabel(c:\data\us1) pop
\end{verbatim}

displays the description of the series POP in the US1 database.

Cross-references


See also \texttt{hfetch (p. 291)} and \texttt{hconvert (p. 290)}.

\begin{tabular}{|c|c|}
\hline
\textbf{hpf} & \textbf{Command || Series Proc} \\
\hline
\end{tabular}

Hodrick-Prescott filter. \texttt{hpf} smooths a series using the Hodrick-Prescott filter.

Syntax

Command: \texttt{hpf(n, options) series\_name filtered\_name}

Series Proc: \texttt{series\_name.hpf(n, options) filtered\_name}

Follow the \texttt{hpf} keyword with the name of the series you want to filter and a name for the filtered series.

When used as a series procedure, \texttt{hpf} smooths the series without displaying the graph of the smoothed series.
**Options**

<table>
<thead>
<tr>
<th>n</th>
<th>You may specify the smoothing parameter; a larger number results in more smoothing. The default is “100” for annual, “1600” for quarterly, and “14400” for monthly data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>Print the graph of the smoothed series and the original series.</td>
</tr>
</tbody>
</table>

**Examples**

The command

```
hpf(1000) gdp gdp_hp
```

smoothes the GDP series with a smoothing parameter “1000” and saves the smoothed series as GDP_HP.

**Cross-references**


<table>
<thead>
<tr>
<th>impulse</th>
<th>Var View</th>
</tr>
</thead>
</table>

Display impulse response functions of a VAR.

**Syntax**

```
Var View: var_name.impulse(n, options) ser1 [ser2 ser3 ...] [@ shock_series [@ ordering_series]]
```

List the series names in the var whose responses you would like to compute. You may optionally specify the sources of shocks. To specify the shocks, list the series after an “@”. By default, EViews computes the responses to all possible sources of shocks using the ordering in the Var.

If you are using impulses from the Cholesky factor, you may change the Cholesky ordering by listing the order of the series after a second “@”.

You must specify the number of periods $n$ for which you wish to compute the impulse responses.
Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>g (default)</strong></td>
<td>Display combined graphs, with impulse responses of one variable to all shocks shown in one graph. If you choose this option, standard error bands will not be displayed.</td>
</tr>
<tr>
<td><strong>m</strong></td>
<td>Display multiple graphs, with impulse response to each shock shown in separate graphs.</td>
</tr>
<tr>
<td><strong>t</strong></td>
<td>Tabulate the impulse responses.</td>
</tr>
<tr>
<td><strong>a</strong></td>
<td>Accumulate the impulse responses.</td>
</tr>
<tr>
<td><strong>imp = arg (default = “chol”)</strong></td>
<td>Type of factorization for the decomposition: unit impulses, ignoring correlations among the residuals (“imp = unit”), non-orthogonal, ignoring correlations among the residuals (“imp = nonort”), Cholesky with d.f. correction (“imp = chol”), Cholesky without d.f. correction (“imp = mlechol”), Generalized (“imp = gen”), structural (“imp = struct”), or user specified (“imp = user”). The structural factorization is based on the estimated structural VAR. To use this option, you must first estimate the structural decomposition; see <strong>svar</strong> (p. 456). For user specified impulses, you must specify the name of the vector/matrix containing the impulses using the “fname = ” option. The option “imp = mlechol” is provided for backward compatibility with EViews 3.x and earlier.</td>
</tr>
<tr>
<td><strong>fname = name</strong></td>
<td>Specify name of vector/matrix containing the impulses. The vector/matrix must have $k$ rows and 1 or $k$ columns, where $k$ is the number of endogenous variables.</td>
</tr>
</tbody>
</table>
Examples

\[ \text{var var1.ls 1 4 m1 gdp cpi} \]
\[ \text{var1.impulse(10,m) gdp @ m1 gdp cpi} \]

The first line declares and estimates a VAR with three variables. The second line displays multiple graphs of the impulse responses of GDP to shocks to the three series in the VAR using the ordering as specified in VAR1.

\[ \text{var1.impulse(10,m) gdp @ m1 @ cpi gdp m1} \]
displays the impulse response of GDP to a one standard deviation shock in M1 using a different ordering.
Cross-references
See Chapter 24 of the User’s Guide for a discussion of variance decompositions in VARs.
See also `decomp` (p. 245).

<table>
<thead>
<tr>
<th>jbera</th>
<th>Var View</th>
</tr>
</thead>
</table>

Multivariate residual normality test.

Syntax
```
Var View: var_name.jbera(options)
```

You must specify a factorization method by the “factor=” option.

Options
```
factor = chol   Factorization by the inverse of the Cholesky factor of the residual covariance matrix.
factor = cor    Factorization by the inverse square root of the residual correlation matrix (Doornik and Hansen 1994).
factor = cov    Factorization by the inverse square root of the residual covariance matrix (Urzua 1997).
factor = svar   Factorization matrix from structural VAR. You must first estimate the structural factorization to use this option; see `svar` (p. 456).
name =         Save the test statistics in a named matrix object. See below for a description of the statistics contained in the stored matrix.
p              Print the test results.
```

The “name = ” option stores the following matrix. Let the VAR have $k$ endogenous variables. Then the stored matrix will have dimension $(k + 1) \times 4$. The first $k$ rows contain statistics for each orthogonal component, where the 1st column is the third moments, the second column is the one-degree of freedom chi-square statistics for the third moments, the third column is the fourth moments, and the fourth column is the one-degree of freedom $\chi^2$ statistics for the fourth moments. The sum of the second and fourth columns are the Jarque-Bera statistics reported in the last output table.
The \((k+1)\) row contains statistics for the joint test. The second and fourth column of the \((k+1)\) row is simply the sum of all the rows above in the corresponding column and are the \(\chi^2(k)\) statistics for the joint skewness and kurtosis tests, respectively. These joint skewness and kurtosis statistics add up to the joint Jarque-Bera statistic reported in the output table, except for the “factor=\text{cov}” option. When this option is set, the joint Jarque-Bera statistic includes all cross moments (in addition to the pure third and fourth moments). The overall Jarque-Bera statistic for this option is stored in the first column of the \((k+1)\) row (which will be a missing value for all other options).

**Examples**

```plaintext
var var1.ls 1 6 lgdp lml lcpi
show var1.jbera(factor=cor,name=jb)
```

The first line declares and estimates a VAR. The second line carries out the residual multivariate normality test using the inverse square root of the residual correlation matrix as the factorization matrix and stores the results in a matrix named JB.

**Cross-references**

See Chapter 24 of the *User’s Guide* for a discussion of the test and other VAR diagnostics.

### kdensity

<table>
<thead>
<tr>
<th><strong>Series View</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>kdensity</strong></td>
</tr>
</tbody>
</table>

Kernel density plots. Displays nonparametric kernel density estimates of the specified series.

**Syntax**

```plaintext
Series View: series_name.kdensity(options)
```

**Options**

- \(s\) (default) Automatic bandwidth (Silverman).
- \(b=number\) Specify a number for the bandwidth.
- \(b\) Bracket bandwidth.
- \(n\) (default = 100) Number of points to evaluate.
- \(x\) Exact evaluation of kernel density.
Examples

\texttt{lwage.kdensity(k=n)}

plots the kernel density estimate of LWAGE using a normal kernel and the automatic bandwidth selection.

Cross-references


<table>
<thead>
<tr>
<th>kerfit</th>
<th>Group View</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fits a kernel regression of the second series in the group (vertical axis) against the first series in the group (horizontal axis).

Syntax

\texttt{Group View: group_name.kerfit(options)}

Options

<table>
<thead>
<tr>
<th>k = arg</th>
</tr>
</thead>
<tbody>
<tr>
<td>(default = “e”)</td>
</tr>
</tbody>
</table>


b = number

Specify a number for the bandwidth.

b

Bracket bandwidth.

integer

Number of grid points to evaluate.

x

Exact evaluation of the polynomial fit.
Examples

group gg weight height
gg.kerfit(s=w_fit, 200)

Fits a kernel regression of HEIGHT on WEIGHT using 200 points and saves the fitted series as W_FIT.

Cross-references


See also linefit (p. 308), nnfit (p. 341).

<table>
<thead>
<tr>
<th>label</th>
<th>Object View</th>
<th>Object Proc</th>
</tr>
</thead>
</table>

Display or change the label view of the object, including the last modified date and display name (if any).

As a procedure, label changes the fields in the object label.

Syntax

Object View: object_name.label
Object Proc: object_name.label(options) text

Options

To modify the label, you should specify one of the following options along with optional text. If there is no text provided, the specified field will be cleared:

c    Clears all text fields in the label.
d    Sets the description field to text.
s    Sets the source field to text.
Examples
The following lines replace the remarks field of LWAGE with “Data from CPS 1988 March File”:

\[ \text{l wage.label(r)} \]
\[ \text{l wage.label(r) Data from CPS 1988 March File} \]

To append additional remarks to LWAGE, and then to print the label view:

\[ \text{l wage.label(r) Log of hourly wage} \]
\[ \text{l wage.label(p)} \]

To clear and then set the units field, use:

\[ \text{l wage.label(u) Millions of bushels} \]

Cross-references
See also displayname (p. 251).

<table>
<thead>
<tr>
<th>laglen</th>
<th>Var View</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR lag order selection criteria.</td>
<td></td>
</tr>
</tbody>
</table>

Syntax
Var View: \( \text{var\_name.laglen}(m, \text{options}) \)

You must specify the maximum lag order \( m \) to test for.

Options

| vname = arg | Save selected lag orders in named vector. See below for a description of the stored vector. |
| mname = arg | Save lag order criteria in named matrix. See below for a description of the stored matrix. |
| p | Print table of lag order criteria. |
The “vname=” option stores a vector with 5 rows containing the selected lags from the following criteria: sequential modified LR test (row 1), final prediction error (row 2), Akaike information criterion (row 3), Schwarz information criterion (row 4), Hannan-Quinn information criterion (row 5).

The “mname=” option stores a $q \times 6$ matrix, where $q = m + 1$ if there are no exogenous variables in the VAR, otherwise $q = m + 2$. The first $(q - 1)$ rows contain the information displayed in the table view following the same order. In addition, the saved matrix has an additional row which contains the lag order selected from each column criterion. The first column (corresponding to the log likelihood values) of the last row is always an NA.

Examples

```eviews
var var1.ls 1 6 lgdp lml lcpi
show var1.laglen(12,vname=v1)
```

The first line declares and estimates a VAR. The second line computes the lag length criteria up to a maximum of 12 lags and stores the selected lag orders in a vector named V1.

Cross-references

See Chapter 24 of the User’s Guide for a discussion of the various criteria and other VAR diagnostics.

See also `testlags` (p. 467).

<table>
<thead>
<tr>
<th>legend</th>
<th>Graph Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Set legend appearance and placement in graphs.

When `legend` is used with a multiple graph, the legend settings apply to all graphs. See `setelem` (p. 412) for setting legends for individual graphs in a multiple graph.

Syntax

```eviews
Graph Proc:      graph_name.legend option_list
```

Note: the syntax of the `legend` proc has changed considerably from Version 3.1 of EViews. While not documented here, the EViews 3 options are still (for the most part) supported. However, we do not recommend using the old options as future support is not guaranteed.
Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>columns(arg)</td>
<td>Columns for legend: “auto” (automatically choose number of columns), int (put legend in specified number of columns).</td>
</tr>
<tr>
<td>(default = “auto”)</td>
<td></td>
</tr>
<tr>
<td>display/-display</td>
<td>Display/do not display the legend.</td>
</tr>
<tr>
<td>font(nI)</td>
<td>Size of font for legend.</td>
</tr>
<tr>
<td>inbox/-inbox</td>
<td>Put legend in box/remove box around legend.</td>
</tr>
<tr>
<td>position(arg)</td>
<td>Position for legend: “left” or “l” (place legend on left side of graph), “right” or “r” (place legend on right side of graph), “botleft” or “bl” (place left-justified legend below graph), “botcenter” or “bc” (place centered legend below graph), “btright” or “br” (place right-justified legend below graph), “(h, v)” (the first number h specifies the number of virtual inches to offset to the right from the origin. The second number v specifies the virtual inch offset below the origin. The origin is the upper left hand corner of the graph).</td>
</tr>
<tr>
<td>p</td>
<td>Print the graph.</td>
</tr>
</tbody>
</table>

The default settings are taken from the global defaults.

Examples

mygra1.legend display position(l) inbox

places the legend of MYGRA1 in a box to the left of the graph.

mygra1.legend position(.2,.2) -inbox

places the legend of MYGRA1 within the graph, indented slightly from the upper left corner with no box surrounding the legend text.

Cross-references


See also setelem (p. 412) for changing legend text and other graph options.
Display a line graph of object, or change existing graph object type to line plot.

The line graph view of a group plots all series in the group in a graph. The line graph view of a matrix plots each column in the matrix in a graph.

Syntax

Command:       line(options) arg1 [arg2 arg3 ...]
Object View:   object_name lineaptions)
Graph Proc:    graph_name.line(options)

Options

Template and printing options

- o = graph_name: Use appearance options from the specified graph.
- t = graph_name: Use appearance options and copy text and shading from the specified graph.
- p: Print the line graph.

Scale options (for multiple line views of group and matrix objects)

- a (default): Automatic single scale.
- d: Dual scaling with no crossing. The first series is scaled on the left and all other series are scaled on the right.
- x: Dual scaling with possible crossing. See the “d” option.
- n: Normalized scale (zero mean and unit standard deviation). May not be used with the “s” option.
- s: Stacked line graph. Each area represents the cumulative total of the series listed. The difference between areas corresponds to the value of a series. May not be used with the “l” option.
- m: Plot lines in multiple graphs (will override the “s” or “l” options). Not for use with an existing graph object.
Panel options

The following options apply when graphing panel structured data.

```
panel = arg
(default taken
from global set-
tings)
```

Panel data display: “stack” (stack the cross-sections), “individual” or “1” (separate graph for each cross-section), “combine” or “c” (combine each cross-section in single graph; one time axis), “mean” (plot means across cross-sections), “mean1se” (plot mean and +/- 1 standard deviation summaries), “mean2sd” (plot mean and +/- 2 s.d. summaries), “mean3sd” (plot mean and +/- 3 s.d. summaries), “median” (plot median across cross-sections), “med25” (plot median and +/- .25 quantiles), “med10” (plot median and +/- .10 quantiles), “med05” (plot median +/- .05 quantiles), “med025” (plot median +/- .025 quantiles), “med005” (plot median +/- .005 quantiles), “med-mxm” (plot median, max and min).

Examples

```
group gl gdp cons m1
g1.line(d)
```

plots line graphs of the three series in group G1 with dual scaling (no crossing). The latter two series will share the same scale.

```
matrix1.line(t=mygra)
```

displays line graphs of the columns of MATRIX1 using the graph object MYGRA as a template.

```
line(m) gdp cons m1
```

creates an untitled graph object that contains three line graphs of GDP, CONS, and M1, with each plotted separately.

Cross-references


See also graph (p. 288) for graph declaration and other graph types.
### linefit

**Scatter plot with bivariate fit.**

Displays the scatter plot of the second series (horizontal axis) and the first series (vertical axis) with a regression fit line. You can specify various transformation methods and weighting for the bivariate fit.

**Syntax**

```plaintext
Group View: group_name.linefit(options)
```

**Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>yl</code></td>
<td>Take the natural log of first series, <code>y</code>.</td>
</tr>
<tr>
<td><code>yi</code></td>
<td>Take the inverse of <code>y</code>.</td>
</tr>
<tr>
<td><code>yp = number</code></td>
<td>Take <code>y</code> to the power of the specified number.</td>
</tr>
<tr>
<td><code>yb = number</code></td>
<td>Take the Box-Cox transformation of <code>y</code> with the specified parameter.</td>
</tr>
<tr>
<td><code>xl</code></td>
<td>Take the natural log of <code>x</code>.</td>
</tr>
<tr>
<td><code>xi</code></td>
<td>Take the inverse of <code>x</code>.</td>
</tr>
<tr>
<td><code>xp = number</code></td>
<td>Take <code>x</code> to the power of the specified number.</td>
</tr>
<tr>
<td><code>xb = number</code></td>
<td>Take the Box-Cox transformation of <code>x</code> with the specified parameter.</td>
</tr>
<tr>
<td><code>xd = integer</code></td>
<td>Fit a polynomial of <code>x</code> up to the specified power.</td>
</tr>
<tr>
<td><code>m = integer</code></td>
<td>Set number of robustness iterations.</td>
</tr>
<tr>
<td><code>s = name</code></td>
<td>Save the fitted <code>y</code> series.</td>
</tr>
<tr>
<td><code>p</code></td>
<td>Print the scatter plot.</td>
</tr>
</tbody>
</table>

If the polynomial degree of `x` leads to singularities in the regression, EViews will automatically drop the high order terms to avoid collinearity.

**Examples**

```plaintext
group g1 inf unemp

G1.linefit(yl,xl,s=yfit)
```
displays a scatter plot of log UNEMP against log INF together with the fitted values from a regression of log UNEMP on the log INF. The fitted values are saved in a series named YFIT. Note that the saved fitted values are for the original UNEMP, not the log transform.

\[
gl.linefit(yb=0.5,m=10)
\]

The Box-Cox transformation of UNEMP with parameter 0.5 is regressed on INF with 10 iterations of bisquare weights.

**Cross-references**

See “Scatter with Regression” on page 374 of the User’s Guide for a discussion of scatter plot with regression fit.

See also `nnfit` (p. 341) and `kerfit` (p. 301).

<table>
<thead>
<tr>
<th>link</th>
<th>Object Declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td>link</td>
<td>Object Declaration</td>
</tr>
</tbody>
</table>

Create a series link object.

Declares a link object which may be used to refer to data in an series on a different work-file page. Links are used to create automatically updating match merges involving dates (frequency conversion) or involving index series.

**Syntax**

Command:

```
link link_name
```

Command:

```
link link_name.linkto(options) link specification
```

Follow the `link` keyword with the name to be given to the link object. If desired, you may combine the declaration with the `linkto` proc in order to provide a full link specification.

**Examples**

```
link mylink
```

creates the link MYLINK with no link specification, while,

```
link l1.linkto(c=obs,nacat) indiv::x @src ind1 ind2 @dest ind1
```

combines the link declaration with the link specification step.

**Cross-references**

See also `linkto` (p. 310) and `unlink` (p. 479).
Define the specification of a series link.

Specify the method by which the object uses data in an existing series. Links are used to perform cross-page match merging or frequency conversion.

**Syntax**

<table>
<thead>
<tr>
<th>Link Proc</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>linkto</code></td>
<td>Link Proc: <code>link_name.linkto(options) source_page::series_name [src_id dest_id]</code></td>
</tr>
<tr>
<td></td>
<td>Link Proc: <code>link_name.linkto(options) source_page::series_name [@src src_ids @dest dest_ids]</code></td>
</tr>
</tbody>
</table>

The most common use of `linkto` will be to define a link that employs general match merging. You should use the keyword `linkto` followed by any desired options, and then provide the name of the source series followed by the names of the source and destination IDs. If more than one identifier series is used, you must separate the source and destination IDs using the “@SRC” and “@DEST” keywords.

In the special case where you wish to link your data using date matching you must use the special keyword “@DATE” as an ID series for a regular frequency page. If “@DATE” is not specified as either a source or destination ID, EViews will perform an exact match merge using the specified identifiers.

The other use of `linkto` will be to define a frequency conversion link between two regular frequency pages. To specify a frequency conversion link, you should use the `linkto` keyword followed by any desired options and then the name of a numeric source series. You must not specify ID series since a frequency conversion link must use the implicit dates associated with the regular frequency pages—if ID series are specified, the link will instead employ general match merging. Note also that if ID series are not specified, but a general match merge specific conversion option is provided (e.g., “c=med”), “@DATE @DATE” will be appended to the list of IDs and a general match merge employed.

It is worth mentioning that a frequency conversion link that uses an alpha source series will generate an evaluation error.

Note that linking by frequency conversion is the same as linking by general match merge using the source and destination IDs “@DATE @DATE” with the following exceptions:

- General match merge linking offers contraction methods not available with frequency conversion (e.g., median, variance, skewness).
- General match merge linking allows you to use samples to restrict the source observations used in evaluating the link.
- General match merge linking allows you to treat NA values in the ID series as a category to be used in matching.
- Frequency conversion linking offers expansion methods other than repeat.
- Frequency conversion linking provides options for the handling of NA values.

Options

```
smpl = "sample_spec"  Sample to be used when computing contractions in a link by match merge. By default, EViews will use the entire workfile sample "@ALL".

c = arg  Set the match merge contraction or the frequency conversion method.

If you are linking a numeric source series by general match merge, the argument can be one of: "mean", "med" (median), "max", "min", "sum", "sumsq" (sum-of-squares), "var" (variance), "sd" (standard deviation), "skew" (skewness), "kurt" (kurtosis), "quant" (quantile, used with "quant=" option), "obs" (number of observations), "nas" (number of NA values), "first" (first observation in group), "last" (last observation in group), "unique" (single unique group value, if present), "none" (disallow contractions).

If linking an alpha series, only the non-summary methods "max", "min", "obs", "nas", "first", "last", "unique" and "none" are supported. For numeric links, the default contraction method is "c=mean"; for alpha links, the default is "c=unique".

If you are linking by frequency conversion, you may use this argument to specify the up- or downconversion method using the options found in fetch (p. 264). The default frequency conversion methods are taken from the series defaults.

quant = number  Quantile value to be used when contracting using the "c=quant" option (e.g. "quant=.3").
```
Most of the conversion options should be self-explanatory. As for the others: “first” and “last” give the first and last non-missing observed for a given group ID; “obs” provides the number of non-missing values for a given group; “nas” reports the number of NAs in the group; “unique” will provide the value in the source series if it is the identical for all observations in the group, and will return NA otherwise; “none” will cause the link to fail if there are multiple observations in any group—this setting may be used if you wish to prohibit all contractions.

On a match merge expansion, linking by ID will repeat the values of the source for every matching value of the destination. If both the source and destination have multiple values for a given ID, EViews will first perform a contraction in the source (if not ruled out by “c=none”), and then perform the expansion by replicating the contracted value in the destination.

Examples

General Match Merge Linking

Let us start with a concrete example. Suppose our active workfile page contains observations on the 50 states of the US, and contains a series called STATE containing the unique state identifiers. We also have a workfile page called INDIV that contains data on individuals from all over the country, their incomes (INCOME), and their state of birth (BIRTHSTATE).

Now suppose that we wish to find the median income of males in our data for each possible state of birth, and then to match merge that value into our 50 observation state page.

The following commands

```plaintext
link male_income
male_income.linkto(c=med, smpl="if male=1") indiv::income birthstate state
```

create the series link MALE_INCOME. MALE_INCOME contains links to the individual INCOME data, telling EViews to subsample only observations where MALE = 1, to compute median values for individuals in each BIRTHSTATE, and to match observations by comparing the values of BIRTHSTATE to STATE in the current page.

In this next example, we link to the series X in the INDIV page, matching values of the IND1 and the IND2 series in the two workfile pages. The link will compute the number of valid observations in the X series for each index group, with NA values in the ID series treated as a valid identifier value.

```plaintext
nacat Treat “NA” values as a category when performing link by general match merge operations.
```
You may wish to use the “@DATE” keyword as an explicit identifier, in order to gain access to our expanded date matching feature. In our annual workfile, the command

\[
\text{link gdp.linkto(c=sd) monthly::gdp @date @date}
\]

will create link that computes the standard deviation of the values of GDP for each year and then match merges these values to the years in the current page. Note that this command is equivalent to

\[
\text{link gdp.linkto(c=sd) quarterly::gdp}
\]

since the presence of the match merge option “c=sd” and the absence of indices instructs EViews to perform the link by ID using the defaults “@DATE” and “@DATE”.

**Frequency Conversion Linking**

Suppose that we are in an annual workfile page and wish to link data from a quarterly page. Then the commands

\[
\text{link gdp}
\]

\[
\text{gdp.linkto quarterly::gdp}
\]

creates a series link GDP in the current page containing a link by date to the GDP series in the QUARTERLY workfile page. When evaluating the link, EViews will automatically frequency convert the quarterly GDP to the annual frequency of the current page, using the series default conversion options. If we wish to control the conversion method, we can specify the conversion method as an option:

\[
\text{gdp.linkto(c=s) quarterly::gdp}
\]

links to GDP in the QUARTERLY page, and will frequency convert by summing the non-missing observations.

**Cross-references**

For a discussion of linking, see Chapter 8, “Series Links”, on page 159 of the User’s Guide. See also link (p. 309) and unlink (p. 479).

<table>
<thead>
<tr>
<th>load</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Load a workfile.

Provided for backward compatibility. Same as **wfopen** (p. 492).
## logit Command

Estimate binary models with logistic errors.

Provide for backward compatibility. Equivalent to issuing the command, `binary` with the option “(d=l)”.

See `binary` (p. 202).

### Logl Object Declaration

Declare likelihood object.

**Syntax**

Command: `logl logl_name`

**Examples**

```matlab
logl ll1
```

declares a likelihood object named LL1.

```matlab
ll1.append @logl logl1
ll1.append res1 = y-c(1)-c(2)*x
ll1.append logl1 = log(@dnorm(res1/@sqrt(c(3))))-log(c(3))/2
```

specifies the likelihood function for LL1 and estimates the parameters by maximum likelihood.

**Cross-references**

See Chapter 22 of the *User’s Guide* for further examples of the use of the likelihood object.

See also `append` (p. 186) for adding specification lines to an existing likelihood object, and `ml` (p. 338) for estimation.
Estimation by linear or nonlinear least squares regression.

When used as a pool proc or when the current workfile has a panel structure, `ls` also estimates cross-section weighed least squares, feasible GLS, and fixed and random effects models.

**Syntax**

**Command:**

```
ls(options) y1 [x2 x3 ...]
```

Equation Method:

```
eq_name.ls(options) y1 [x2 x3 ...]
```

```
eq_name.ls(options) specification
```

```
paneleq_name.ls(options) y1 [x2 x3 ...] [@cxreg z1 z2 ...]
```

```
[@perreg z3 z4 ...]
```

Pool Method:

```
pool_name.ls(options) y1 [x2 x3 ...] [@cxreg z1 z2 ...] [@perreg z3 z4 ...]
```

System Method:

```
system_name.ls(options)
```

**VAR Method:**

```
var_name.ls(options)
```

For linear specifications, list the dependent variable first, followed by a list of the independent variables. Use a “C” if you wish to include a constant or intercept term; unlike some other programs, EViews does not automatically include a constant in the regression. You may add AR, MA, SAR and SMA error specifications and PDL specifications for polynomial distributed lags. If you include lagged variables, EViews will adjust the sample automatically, if necessary.

Both dependent and independent variables may be created from existing series using standard EViews functions and transformations. EViews treats the equation as linear in each of the variables and assigns coefficients C(1), C(2), and so forth to each variable in the list.

Linear or nonlinear single equations can also be specified by explicit equation. You should specify the equation as a formula. The parameters to be estimated should be called “C(1)”, “C(2)”, and so forth (assuming that you wish to use the default coefficient vector “C”). You may also declare an alternative coefficient vector using `coef` and use these coefficients in your expressions.
For equation and var objects, you can declare and estimate the object in a single command. Follow the declaration of the object type with the name of the new object, then the `ls` keyword, separated by a period.

When used as a pool method, `ls` carries out panel data estimation. Type the name of the dependent variable followed by one or two lists of regressors. The first list should contain ordinary and pool series that are restricted to have the same coefficient across all members of the pool. The second list, if provided, should contain variables that have different coefficients for each member of the pool. If there is a second list, the two lists must be separated by an “@”-sign.

For pool estimation, you may include AR terms as regressors (except when estimating random effects models using the “r” option) but not MA terms.

**Options**

**General options**

| m = integer | Set maximum number of iterations. |
| c = scalar  | Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. |

| s = number  | Determine starting values for equations specified by list with AR or MA terms. Specify a number between zero and one representing the fraction of preliminary least squares estimates computed without AR or MA terms. Note that out of range values are set to “s = 1”. Specifying “s = 0” initializes coefficients to zero. By default EViews uses “s = 1”. |

| showopts / -showopts | [Do / do not] display the starting coefficient values and estimation options in the estimation output. |
deriv = keyword  Set derivative methods. The argument keyword should be a one or two letter string. The first letter should either be “f” or “a” corresponding to fast or accurate numeric derivatives (if used). The second letter should be either “n” (always use numeric) or “a” (use analytic if possible). If omitted, EViews will use the global defaults.

p  Print basic estimation results.

Additional Options for Equation, System, and Var estimation

w = series_name  Weighted Least Squares. Each observation will be weighted by multiplying by the specified series.

h  White’s heteroskedasticity consistent standard errors.

n  Newey-West heteroskedasticity and autocorrelation consistent (HAC) standard errors.

z  Turn off backcasting in ARMA models.

Note: not all options are available for all equation methods. See the User’s Guide for details on each estimation method.

Additional Options for Pool and Panel Equation estimation

cx = arg  Cross-section effects: (default) none, fixed effects (“cx = f”), random effects (“cx = r”).

per = arg  Period effects: (default) none, fixed effects (“per = f”), random effects (“per = r”).

wgt = arg  GLS weighting: (default) none, cross-section system weights (“wgt = cxsur”), period system weights (“wgt = persur”), cross-section diagonal weights (“wgt = cxdiag”), period diagonal weights (“wgt = perdiag”).

Examples

```
equation eq1.ls m1 c uemp inf(0 to -4) @trend(1960:1)
equation eq2.ls(z) d(tbill) c inf @seas(1) @seas(1)*inf ma(2)
equation eq3.ls(h) q = beta(1)+beta(2)*(l^c(1) + k^(1-c(1)))
```

estimates a linear regression of M1 on a constant, UEMP, INF (from current up to four lags), and a linear trend.

regresses the first difference of TBILL on a constant, INF, a seasonal dummy, and an interaction of the dummy and INF with an MA(2) error term. The “z” option turns off backcasting.

```
coef(2) beta
param beta(1) .2 beta(2) .5 c(1) 0.1
```

Note that random effects models currently do not permit weight iteration to convergence.

---

**keepwgts**

Keep full set of GLS weights used in estimation with object, if applicable (by default, only small memory weights are saved).

**rancalc = arg**

Random component method: Swamy-Arora ("rancalc = sa"), Wansbeek-Kapteyn ("rancalc = wk"), Wallace-Hussain ("rancalc = wh").

**nodf**

Do not perform degree-of-freedom corrections in computing coefficient covariance matrix. The default is to use degree-of-freedom corrections.

**b**

Estimate using a balanced sample (pool estimation only).

**coef = arg**

Specify the name of the coefficient vector (if specified by list); the default behavior is to use the “C” coefficient vector.

**iter = arg**

Iteration control for GLS specifications: perform one weight iteration, then iterate coefficients to convergence ("iter = onec"), iterate weights and coefficients simultaneously to convergence ("iter = sim"), iterate weights and coefficients sequentially to convergence ("iter = seq"), perform one weight iteration, then one coefficient step ("iter = oneb").

Note that random effects models currently do not permit weight iteration to convergence.
estimates the nonlinear regression starting from the specified initial values. The “h” option reports heteroskedasticity consistent standard errors.

\begin{align*}
equation \text{eq4.ls } r &= c(1)+c(2)*r(-1)+\text{div(-1)^c(3)} \\
\text{sym betacov} &= \text{eq4.@cov}
\end{align*}

declares and estimates a nonlinear equation and stores the coefficient covariance matrix in a symmetric matrix called BETACOV.

\begin{align*}
\text{pool1.ls dy? inv? edu? year}
\end{align*}
estimates pooled OLS of DY on a constant, INV, EDU, and YEAR.

\begin{align*}
\text{pool1.ls(f) dy? @ inv? edu? year ar(1)}
\end{align*}
estimates a fixed effects model without restricting any of the coefficients to be the same across pool members.

\begin{align*}
\text{group rhs c dum1 dum2 dum3 dum4} \\
\text{ls cons rhs ar(1)}
\end{align*}
uses the group definition for RHS to regress CONS on C, DUM1, DUM2, DUM3, and DUM4 with an AR(1) correction.

Cross-references


See “Equation” (p. 139), “Pool” (p. 153), “System” (p. 166), and “Var” (p. 171) for a complete description of these objects, including a list of saved results.

See also Appendix C, “Special Expression Reference”, on page 519, for special terms that may be used in \texttt{ls} specifications.

\begin{tabular}{|l|l|}
\hline
\texttt{makecoint} & Var Proc \\
\hline
\end{tabular}

Create group containing the estimated cointegrating relations from a VEC.

Syntax

- \texttt{Var Proc: var\_name.makecoint}
- \texttt{Var Proc: var\_name.makecoint group\_name}

From the command window, the first form creates an untitled group object containing the estimated cointegrating relations in the VEC. The second form allows you to name the group object by following the \texttt{makecoint} proc with a name to be given to the group. The
series contained in the group are named COINTEQ01, COINTEQ02, and so on. In batch mode (program files), the first form simply creates the series COINTEQ01, COINTEQ02, and so on.

This proc will return an error message unless you have estimated an error correction model with the var object.

**Examples**

```plaintext
var vec1.ec(b,2) 1 4 y1 y2 y3
vec1.makemcoint gcoint
```

The first line estimates a VEC with 2 cointegrating relations. The second line creates a group named GC OINT which contains the two estimated cointegrating relations. The two cointegrating relations will be stored as series named COINTEQ01 and COINTEQ02 if these names have not yet been used in the workfile. If these names are already used, the next unused number will be appended to the prefix “COINTEQ” and will be used to name the series.

**Cross-references**

See Chapter 24 of the *User’s Guide* for details.

See also `coint` (p. 224).

### makederivs

**Equation Proc**

Make a group containing individual series which hold the derivatives of the equation specification.

**Syntax**

```plaintext
Equation Proc: equation_name.makederivs(options) [ser1 ser2 ...]
```

If desired, enclose the name of the group object to contain the series as an option (in parentheses following the command name).

The argument specifying the names of the series is also optional. If not provided, EViews will name the series “DERIV##” where # is a number such that “DERIV##” is the next available unused name. If the names are provided, the number of names must match the number of target series.

**Cross-references**


See also `derivs` (p. 249), `grads` (p. 287), `makegrads` (p. 322).
**makeendog**

Make a group out of the endogenous series.

**Syntax**

Object Proc: object_name.makeendog name

Following the keyword `makeendog`, you should provide a name for the group to hold the endogenous series. If you do not provide a name, EViews will create an untitled group.

Note that in EViews 4, `endog` and `makeendog` are no longer supported for model objects. See instead, `makegroup` (p. 324).

**Examples**

```
var1.makeendog grp_v1
```

creates a group named GRP_V1 that contains the endogenous series in VAR1.

**Cross-references**

See also `endog` (p. 259) and `makegroup` (p. 324).

---

**makefilter**

Create a “Kalman filter” sspace object.

Creates a new sspace object with all estimated parameter values substituted out of the specification. This procedure allows you to use the structure of the sspace without reference to estimated coefficients or the estimation sample.

**Syntax**

Sspace Proc: sspace_name.makefilter [filter_name]

If you provide a name for the model in parentheses after the keyword, EViews will quietly create the named model in the workfile. If you do not provide a name, EViews will open an untitled model window if the command is executed from the command line.

**Examples**

```
ss1.makefilter kfilter
```

creates a new sspace object named KFILTER, containing the specification in SS1 with estimated parameter values substituted for coef elements.
Cross-references


See also makesignals (p. 330) and makestates (p. 331).

<table>
<thead>
<tr>
<th>Command</th>
<th>Proc Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>makegarch</td>
<td>Equation Proc</td>
</tr>
</tbody>
</table>

Generate conditional variance series.

Saves the estimated conditional variance (from an equation estimated using ARCH) as a named series.

**Syntax**

```
Equation Proc: eq_name.makegarch series_name
```

Following the `makegarch` keyword you should provide a name for the saved series. If you do not provide a name, EViews will name the series GARCH01 (if GARCH01 already exists, it will be named GARCH02, and so on).

**Examples**

```
equation eq1.arch sp c
eq1.makegarch cvar
plot cvar^.5
```

estimates a GARCH(1,1) model, saves the conditional variance as a series named CVAR, and plots the conditional standard deviation. If you only want to view a plot of the conditional standard deviation use the `garch view`.

**Cross-references**


See also `arch` (p. 187) `archtest` (p. 190) and `garch` (p. 280).

<table>
<thead>
<tr>
<th>Command</th>
<th>Proc Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>makegrads</td>
<td>Equation Proc</td>
</tr>
</tbody>
</table>

Make a group containing individual series which hold the gradients of the objective function.

**Syntax**

```
Object Proc: object_name.makegrads [options] [ser1 ser2 ...]
```

The argument specifying the names of the series is also optional. If the argument is not provided, EViews will name the series “GRAD##” where ## is a number such that “GRAD##” is the next available unused name. If the names are provided, the number of names must match the number of target series.

**Options**

- `n=arg` Name of group object to contain the series.

**Examples**

```plaintext
eq1.grads(n=out)
```

creates a group named OUT containing series named GRAD1, GRAD2, and GRAD3.

```plaintext
eq1.grads(n=out) g1 g2 g3
```

creates the same group, but names the series G1, G2 and G3.

**Cross-references**

See also derivs (p. 249), makederivs (p. 320), grads (p. 287).

<table>
<thead>
<tr>
<th>makegraph</th>
<th>Model Proc</th>
</tr>
</thead>
</table>

Make graph object showing model series.

**Syntax**

Model Proc: `model_name.makegraph(options) graph_name model_vars`

where `graph_name` is the name of the resulting graph object, and `models_vars` are the names of the series. The list of `models_vars` may include the following special keywords:

- `@all` All model variables.
- `@endog` All endogenous model variables.
- `@exog` All exogenous model variables
- `@addfactor` All add factor variables in the model

**Options**

- `a` Include actuals.
- `c` Include comparison scenarios.
- `d` Include deviations.
Examples

\[
\text{mod1.makegraph(a) gr1 y1 y2 y3}
\]
creates a graph containing the model series Y1, Y2, and Y3 in the active scenario and the actual Y1, Y2, and Y3.

\[
\text{mod1.makegraph(a,t=pchy) gr1 y1 y2 y3}
\]
plots the same graph, but with data displayed as 1-year percent changes.

Cross-references


See \text{makegroup} (p. 324).

<table>
<thead>
<tr>
<th>\text{makegroup}</th>
<th>Model Proc</th>
<th>Pool Proc</th>
</tr>
</thead>
</table>

Make group out of pool and ordinary series using a pool object or make group out of model series and display dated data table.

Syntax

\[
\text{Pool Proc: } \text{pool_name.makegroup(group_name) pool_series1 [pool_series2}
\]
Model Proc: model_name.makegroup(options) grp_name model_vars

When used as a pool proc, following the keyword, you should provide a name for the new group in parentheses, and then list the ordinary and pool series to be placed in the group.

When used as a model proc, the makegroup keyword should be followed by any options, the name of the destination group, and the list of model variables to be created. The options control the choice of model series, and transformation and grouping features of the resulting dated data table view. The list of model_vars may include the following special keywords:

- @all: All model variables.
- @endog: All endogenous model variables.
- @exog: All exogenous model variables.
- @addfactor: All add factor variables in the model.

Options

For Model Proc

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Include actuals</td>
</tr>
<tr>
<td>c</td>
<td>Include comparison scenarios</td>
</tr>
<tr>
<td>d</td>
<td>Include deviations</td>
</tr>
<tr>
<td>n</td>
<td>Do not include active scenario (by default the active scenario is included)</td>
</tr>
<tr>
<td>t = arg (default = level)</td>
<td>Transformation type: “level” (display levels), “pch” (percent change), “pcha” (display percent change - annual rates), “pchy” (display 1-year percent change), “dif” (display 1-period differences), “dify” (display 1-year differences).</td>
</tr>
<tr>
<td>s = arg (default = “d”)</td>
<td>Solution type: “d” (deterministic), “m” (mean of stochastic), “s” (mean and ±2 std. dev. of stochastic), “b” (mean and confidence bounds of stochastic).</td>
</tr>
<tr>
<td>g = arg (default = “v”)</td>
<td>Grouping setting for graphs: “v” (group series in graph by model variable), “s” (group series in graph by scenario).</td>
</tr>
</tbody>
</table>
Examples

```
pool1.makegroup(g1) x? z y?
```
places the ordinary series Z, and all of the series represented by the pool series X? and Y?, in the group G1.

```
model1.makegroup(a,n) group1 @endog
```
places all of the actual endogenous series in the group GROUP1.

Cross-references


See `makegraph` (p. 323).

### makelimits

<table>
<thead>
<tr>
<th>Equation Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>makelimits</strong></td>
</tr>
</tbody>
</table>

Create vector of limit points from ordered models.

`makelimits` creates a vector of the estimated limit points from equations estimated by ordered.

**Syntax**

```
Equation Proc:   eq_name.makelimits [vector_name]
```

Provide a name for the vector after the `makelimits` keyword. If you do not provide a name, EViews will name the vector with the next available name of the form LIMITS## (if LIMITS01 already exists, it will be named LIMITS02, and so on).

**Examples**

```
equation eql.ordered edu c age race gender
eql.makelimit cutoff
```
Estimates an ordered probit and saves the estimated limit points in a vector named CUTOFF.

**Cross-references**

**makemodel**

Make a model from an estimation object.

**Syntax**

Object Proc: object_name.makemodel(name) assign_statement

You should enter the name of the object, followed by a dot and the keyword. If you provide a name for the model in parentheses after the keyword, EViews will create the named model in the workfile. If you do not provide a name, EViews will open an untitled model window if the command is executed from the command line.

**Examples**

```plaintext
var var3.ls 1 4 m1 gdp tb3
var3.makemodel(varmod) @prefix s_
```

estimates a VAR and makes a model named VARMOD from the estimated var object. VARMOD includes an assignment statement “ASSIGN @PREFIX S_”. Use the command “show varmod” or “varmod.spec” to open the VARMOD window.

**Cross-references**


See also `append` (p. 186), `merge` (p. 336) and `solve` (p. 438). For sspace objects, see `makefilter` (p. 321).

**makeregs**

Create a regressor group.

**Syntax**

Equation Proc: equation_name.makeregs grp_name

Follow the keyword `makeregs` with the name of the group.
Examples

```
equation eql.ls y c x1 x2 x3 z
eql.makeregsg reggroup
```

creates a group REGGROUP containing the series Y1, X1, X2, X3, and Z.

Cross-references

See also `group` (p. 290) and `equation` (p. 260).

`makeresids` | `Equation Proc` | `Pool Proc` | `System Proc` | `Var Proc`

Create residual series.

Creates and saves residuals in the workfile from an object with estimated equation or equations.

Syntax

```
Equation Proc:   equation_name.makeresids(options) [res1]
Object Proc:     object_name.makeresids[res1 res2 res3]
Pool Proc:       pool_name.makeresids [poolser]
```

Follow the object name with a period and the `makeresid` keyword and a list of names to be given to the stored residuals. For pool residuals, you may use a cross section identifier “?” to specify a set of names.

For multiple equation objects, you should provide as many names as there are equations. If there are fewer names than equations, EViews creates the extra residual series with names RESID01, RESID02, and so on. If you do not provide any names, EViews will also name the residuals RESID01, RESID02, and so on.

[NOTE: `makeresids` is no longer supported for the sspace object— see `makesignals` (p. 330) for relevant replacement routines]

Options

The following options are available only for the residuals from single equation objects:

```
o (default)    Ordinary residuals.
s             Standardized residuals (available only after weighted estimation and GARCH, binary, ordered, censored, and count models).
```
Examples

equation eq1.ls y c m1 inf unemp
eq1.makeresids res_eq1
estimates a linear regression of Y on a constant, M1, INF, UNEMP and saves the residuals as a series named RES_EQ1.

var macro_var.ls 1 4 y m1 r
macro_var.makeresids resay res_m1riser
estimates an unrestricted VAR with four lags and endogenous variables Y, M1, R and stores the residuals as RES_Y, RES_M1, RES_R.

equation probit1.binary y c x z
probit1.makeresids(g) res_g
series score1=res_g
series score2=res_g*x
series score3=res_g*z
estimates a probit model of Y on a constant, X, Z and stores the generalized residuals as RES_G. Then the vector of scores are computed as SCORE1, SCORE2, SCORE3.

pool1.makeresids res1_?
The residuals of each pool member will have a name starting with “RES1_” and the cross-section identifier substituted for the “?”. 

Cross-references

For state space models, see makesignals (p. 330).
**makesignals**

Generate signal series or signal standard error series from an estimated sspace object.

Options allow you to choose to generate one-step ahead and smoothed values for the signals, and the signal standard errors.

**Syntax**

```
Sspace Proc: name.makesignals(options) [name_spec]
```

Follow the object name with a period and the `makesignals` keyword, options to determine the output type, and a list of names or wildcard expression identifying the series to hold the output. If a list is used to identify the targets, the number of target series must match the number of states implied in the sspace. If any signal variable contain expressions, you may not use wildcard expressions in the destination names.

**Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>n=group_name</code></td>
<td>Name of group to hold newly created series.</td>
</tr>
</tbody>
</table>

**Examples**

```
ss1.makesignals(t=smooth) sm*
```

produces smoothed signals in the series with names beginning with “sm”, and ending with the name of the signal dependent variable.

```
ss2.makesignals(t=pred, n=pred_sigs) sig1 sig2 sig3
```

creates a group named PRED_SIGS which contains the one-step ahead signal predictions in series SIG1, SIG2, and SIG3.
Cross-references


See also `forecast` (p. 272), `makefilter` (p. 321) and `makestates` (p. 331).

### makestates

**SSpace Proc**

Generate state series or state standard error series from an estimated `sspace` object.

Options allow you to choose to generate one-step ahead, filtered, and smoothed values for the states, and the state standard errors.

**Syntax**

SSpace Proc: `name.makestates(options) [name_spec]`

Follow the object name with a period and the `makestate` keyword, options to determine the output type, and a list of names or wildcard expression identifying the series to hold the output. If a list is used to identify the targets, the number of target series must match the number of names implied by the keyword.

**Options**

- **t = output_type (default = “pred”)**

- **n = group_name**
  Name of group to hold newly created series.

**Examples**

```plaintext
ss1.makestates(t=smooth) sm*
```

produces smoothed states in the series with names beginning with “sm”, and ending with the name of the state dependent variable.
ss2.makestates(t=pred, n=pred_states) sig1 sig2 sig3
creates a group named PRED_STATES which contains the one-step ahead state predictions in series SIG1, SIG2, and SIG3.

Cross-references
See also forecast (p. 272), makefilter (p. 321) and makesignals (p. 330).

<table>
<thead>
<tr>
<th>makestats</th>
<th>Pool Proc</th>
</tr>
</thead>
</table>

Create and save series of descriptive statistics computed from a pool object.

Syntax

Pool Proc: pool_name.makestats(options) pool_series1 [pool_series2 ...] @ stat_list

Follow the object name with a period, the makestat command, a list of series names, the "@"-sign, and a list of command names for the statistics to compute. The statistics series will have a name with the cross-section identifier "?" replaced by the statistic command.

Options

*Options in parentheses specify the sample to use to compute the statistics*

i Use individual sample.
c (default) Use common sample.
b Use balanced sample.

*Command names for the statistics to be computed*

obs Number of observations.
mean Mean.
med Median.
var Variance.
.sd Standard deviation.
skew Skewness.
kurt Kurtosis.
Examples

`pool1.makestats gdp_? edu_? @ mean sd`

computes the mean and standard deviation of the GDP_? and EDU_? series in each period across the cross-section members using the default common sample. The mean and standard deviation series will be named GDP_MEAN, EDU_MEAN, GDP_SD, EDU_SD.

`pool1.makestats(b) gdp_? @ max min`

Computes the maximum and minimum values of the GDP_? series in each period using the balanced sample. The max and min series will be named GDP_MAX and GDP_MIN.

Cross-references

See Chapter 27 of the User’s Guide for details on the computation of these statistics and a discussion of the use of individual, common, and balanced samples in pool.

See also `describe` (p. 250).

---

<table>
<thead>
<tr>
<th>makesystem</th>
<th>Pool Proc</th>
<th>Var Proc</th>
</tr>
</thead>
</table>

Create system from a pool object or var.

Syntax

- Pool Proc: `pool_name.makesystem(options) pool_spec`
- Var Proc: `var_name.makesystem(options)`

The first usage creates a system out of the pool specification. See `ls` (p. 315) for details on the syntax of a pool specification. Note that period specific coefficients and effects are not available in this routine.

The second form is used to create a system out of the current var specification. You may elect to order the equations by series (default) or by lags.
Options

Options for Pools:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>No intercept.</td>
</tr>
<tr>
<td>c (default)</td>
<td>Common intercept term.</td>
</tr>
<tr>
<td>f</td>
<td>Cross-section specific intercepts (fixed effects).</td>
</tr>
<tr>
<td>name</td>
<td>Name for the system object.</td>
</tr>
</tbody>
</table>

Options for VARs

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = name</td>
<td>Specify name for the system object.</td>
</tr>
<tr>
<td>bylag</td>
<td>Specify system by lags (default is to order by variables).</td>
</tr>
</tbody>
</table>

Examples

pool1.makesystem(sys1) inv? cap? @ val?

creates a system named SYS1 with INV? as the dependent variable and a common intercept for each cross-section member. The regressor CAP? is restricted to have the same coefficient in each equation, while the VAL? regressor has a different coefficient in each equation.

pool1.makesystem(sys2,cx=f) inv? @cxreg cap? @cxinst @trend inv?(-1)

This command creates a system named SYS2 with INV? as the dependent variable and a different intercept for each cross-section member equation. The regressor CAP? enters each equation with a different coefficient and each equation has two instrumental variables @TREND and INV? lagged.

Cross-references

See Chapter 23 of the User’s Guide for a discussion of system objects in EViews.

map

Assign or remove value map setting.

Syntax

Object Proc: series_name.map [valmap_name]
If the optional valmap name is provided, the procedure will assign the specified value map to the series or alpha. If no name is provided, EViews will remove an existing valmap assignment.

Examples

alpha1.map mymap
assigns the valmap object MYMAP to the alpha series ALPHA1.

series1.map
removes an existing valmap assignment from SERIES1.

Cross-references


matrix—335

Declare and optionally initializes a matrix object.

Syntax

Command: matrix(r, c) matrix_name
Command: matrix(r, c) matrix_name = assignment

The matrix keyword is followed by the name you wish to give the matrix. matrix also takes an optional argument specifying the row r and column c dimension of the matrix.

You can combine matrix declaration and assignment. If there is no assignment statement, the matrix will initially be filled with zeros.

Once declared, matrices may be resized by repeating the matrix command using the original name.

You should use sym for symmetric matrices.

Examples

matrix mom
declares a matrix named MOM with one element, initialized to zero.

matrix(3,6) coefs
declares a 3 by 6 matrix named COEFS, filled with zeros.
Cross-references


See “Matrix” (p. 150), “Rowvector” (p. 156) and “Vector” (p. 174) and “Sym” (p. 164) for full descriptions of the various matrix objects.

**means**

---

Descriptive statistics by category of dependent variable.

Computes and displays descriptive statistics of the explanatory variables (regressors) of an equation by categories/values of the dependent variable. `means` is currently available only for equations estimated by `binary`.

**Syntax**

`Equation View: eq_name.means(options)`

**Options**

- `p` Print the descriptive statistics table.

**Examples**

```
equation eql.binary(d=l) work c edu faminc
eql.means
```

estimates a logit and displays the descriptive statistics of the regressors C, EDU, FAMINC for WORK = 0 and WORK = 1.

**Cross-references**

See Chapter 21 of the User’s Guide for a discussion of binary dependent variable models.

**merge**

---

Merge objects.

When used as a model procedure, merges equations from an estimated equation, model, pool, system, or var object. When used as a graph procedure, merges graph objects.

If you supply only the object’s name, EViews first searches the current workfile for the object containing the equation. If the object is not found, EViews looks in the default direc-
tory for an equation or pool file (.DBE). If you want to merge the equations from a system file (.DBS), a var file (.DBV), or a model file (.DBL), include the extension in the command. You may include a path when merging files. You must merge objects to a model one at a time; `merge` appends the object to the equations already existing in the model.

When used as a graph procedure, `merge` combines graph objects into a single Graph object. The graph objects to merge must exist in the current workfile.

**Syntax**

Model Proc: `model_name.merge(options) object_name`

Graph Proc: `graph name.merge graph1 graph2 [graph3 ...]`

For use as a model procedure, follow the keyword with a name of an object containing estimated equation(s) to merge.

For use as a graph procedure, follow the keyword with a list of existing graph object names to merge.

**Options**

- `t` Merge an ASCII text file (only for model merge).

**Examples**

```plaintext
eq1.makemodel(mod1)
mod1.merge eq2
mod1.merge(t) c:\data\test.txt
```

The first line makes a model named MOD1 from EQ1. The second line merges (appends) EQ2 to MOD1 and the third line further merges (appends) the text file TEST from the specified directory.

```plaintext
graph mygra.merge gra1 gra2 gra3 gra4
show mygra.align(4,1,1)
```

The first line merges the four graphs GRA1, GRA2, GRA3, GRA4 into a graph named MYGRA. The second line displays the four graphs in MYGRA in a single row.

**Cross-references**

metafile

Graph Proc

Save graph to disk as an enhanced or ordinary Windows metafile.

Provided for backward compatibility, `metafile` has been replaced by the more general graph proc `save` (p. 393), which allows for saving graphs in metafile or postscript files, with additional options.

ml

Logl Method | Sspace Method

Maximum likelihood estimation of logl and state space models.

Syntax

Object Method: `object_name.ml(options)`

Options

- **b**
  - Berndt-Hall-Hall-Hausman (BHHH) algorithm (default is Marquardt).
- **m = integer**
  - Set maximum number of iterations.
- **c = scalar**
  - Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.
- **showopts / -showopts**
  - [Do / do not] display the starting coefficient values and estimation options in the estimation output.
- **p**
  - Print basic estimation results.

Examples

```
bvar.ml
```

estimates the sspace object BVAR by maximum likelihood.

Cross-references

See Chapters 22 and 25 of the User’s Guide for a discussion of user specified likelihood and state space models.
Declare a model object

Syntax

Command: \texttt{model model\_name}

The keyword \texttt{model} should be followed by a name for the model. To fill the model, you may use \texttt{append} (p. 186) or \texttt{merge} (p. 336).

Examples

\begin{verbatim}
model macro
macro.append cs=10+0.8*y(-1)
macro.append i=0.7*(y(-1)-y(-2))
macro.append y=cs+i+g
\end{verbatim}

declares an empty model named MACRO and adds three lines to MACRO.

Cross-references


See also \texttt{append} (p. 186), \texttt{merge} (p. 336) and \texttt{solve} (p. 438).

Display model solution messages.

Show view containing messages generated by the most recent model solution.

Syntax

\begin{verbatim}
Model View: model\_name.msg(options)
\end{verbatim}

Options

\begin{verbatim}
p \quad \text{Print the model solution messages.}
\end{verbatim}

Cross-references

See also `solve (p. 438)` and `solveopt (p. 439)`.

### name

<table>
<thead>
<tr>
<th>Graph Proc</th>
</tr>
</thead>
</table>

Change the names used for legends or axis labels in XY graphs.

Allows you to provide an alternative to the names used for legends or for axis labels in XY graphs. The `name` command is available only for single graphs and will be ignored in multiple graphs.

#### Syntax

Graph Proc: `graph_name.name(n) legend_text`

Provide a series number in parentheses and `legend_text` for the legend (or axis label) after the keyword. If you do not provide text, the series name will be removed from the legend/axis label.

#### Examples

```plaintext
g1.line(d) unemp gdp
  g1.name(1) Civilian_unemployment_rate
  g1.name(2) Gross_National_Product
```

The first line creates a line graph named G1 with dual scale, no crossing. The second line replaces the legend of the first series UNEMP, and the third line replaces the legend of the second series GDP.

```plaintext
g2.scat id w h
  g2.name(1)
  g2.name(2) weight
  g2.name(3) height
  g2.legend(l)
```

The first line creates a scatter diagram named G2. The second line removes the legend of the horizontal axis, and the third and fourth lines replace the legends of the variables on the vertical axis. The last line moves the legend to the left side of the graph.

#### Cross-references

See Chapter 14 of the *User’s Guide* for a discussion of working with graphs.

See also `displayname (p. 251)`.
nnfit—341

Nearest neighbor fit.

Displays the fit of the second series (vertical axis) on the first series (horizontal axis) in the group.

Syntax

Group View: group_name.nnfit(options)

Options

- **b = fraction** (default = 0.3)
  - Bandwidth as a fraction of the total sample. The larger the fraction, the smoother the fit.

- **d = integer** (default = 1)
  - Degree of polynomial to fit.

- **b**
  - Bracket bandwidth span.

- **integer** (default = 100)
  - Approximate number of data points at which to compute the fit (if performing subsampling).

- **x**
  - Exact (full) sampling. Default is Cleveland subsampling.

- **u**
  - No local weighting. Default is local weighting with tricube weights.

- **m = integer**
  - Set number of robustness iterations.

- **s = name**
  - Symmetric neighbors. Default is nearest neighbors.

- **s**
  - Save fitted series.

- **p**
  - Print the Loess fit.

Examples

```
    group gr1 gdp90 gdp50
    gr1.nnfit(x,m=3)
```

displays the nearest neighbor fit of GDP50 on GDP90 with exact (full) sampling and 3 robustness iterations. Each local regression fits the default linear regression with tricube weighting using a bandwidth of span 0.3.

```
    group gro1 weight height
    gro1.nnfit(50,d=2,m=3)
```
displays the nearest neighbor fit of HEIGHT on WEIGHT by fitting approximately 50 data points. Each local regression fits a quadratic using tricube robustness weights with 3 iterations.

Cross-references

See “Scatter with Nearest Neighbor Fit” on page 375 of the User’s Guide for a discussion of nearest neighbor regressions.

See also linefit (p. 308) and kerfit (p. 301).

open Command

Opens a program file, or text (ASCII) file.

This command should be used to open program files or text (ASCII) files for editing.

You may also use the command to open workfiles or databases. This use of the open command is provided for backward compatibility. We recommend instead that you use the new commands wfopen (p. 492) and pageload (p. 357) to open a workfile, or dbopen (p. 242) to open databases.

Syntax

Command:  open(options) [path:]file_name

You should provide the name of the file to be opened including the extension (and optionally a path) or the file name without an extension, but with an option identifying its type. Specified types always take precedence over automatic type identification. If not path is provided, EViews will look for the file in the default directory.

Files with the “.PRG” extension will be opened as program files, unless otherwise specified. Files with the “.TXT” extension will be opened as text files, unless otherwise specified.

For backward compatibility, files with extensions that are recognized as database files are opened as EViews databases, unless an explicit type is specified. Similarly, files with the extensions “.WF” and “.WF1”, and foreign files with recognized extensions will be opened as workfiles, unless otherwise specified.

All other names will be read as text files.
Options

<table>
<thead>
<tr>
<th>p</th>
<th>Open file as program file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Open file as text file.</td>
</tr>
</tbody>
</table>
| type=arg | Specify text or program file type using keywords. (
|       | “prg” or “txt”)          |

Examples

open finfile.txt
opens a text file named “FINFILE.TXT” in the default directory.

open "c:\program files\my files\test1.prg"
opens a program file named “TEST1.PRG” from the specified directory.

open a:\mymemo.tex
opens a text file named “MYMEMO.TEX” from the A: drive.

Cross-references

See wfopen (p. 492) and pageload (p. 357) for commands for opening files as workfiles or workfile pages, and dbopen (p. 242) for opening database files.

See Chapter 5 of the User’s Guide for a discussion of basic file operations.

<table>
<thead>
<tr>
<th>options</th>
<th>Graph Proc</th>
</tr>
</thead>
</table>

Set options for a graph object.

Allows you to change the option settings of an existing graph object. When options is used with a multiple graph, the options are applied to all graphs.

Syntax

Graph Proc: graph_name.options option_list

Note: the syntax of the options proc has changed considerably from version 3.1 of EViews. While not documented here, the EViews 3 options are still (for the most part) supported. However, we do not recommend using the old options, as future support is not guaranteed.
Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size(w, h)</td>
<td>Specifies the size of the plotting frame in virtual inches (w = width, h = height).</td>
</tr>
<tr>
<td>inbox / -inbox</td>
<td>[Enclose / Do not enclose] the plotting frame in a box and [include / remove] the yellow background.</td>
</tr>
<tr>
<td>indent / -indent</td>
<td>[Indent / Do not indent] lines/bars/areas/spikes in the plotting frame.</td>
</tr>
<tr>
<td>lineauto</td>
<td>Use solid lines when drawing in color and use patterns and grayscale when drawing in black and white.</td>
</tr>
<tr>
<td>linesolid</td>
<td>Always use solid lines.</td>
</tr>
<tr>
<td>linepat</td>
<td>Always use line patterns.</td>
</tr>
<tr>
<td>color / -color</td>
<td>Specifies that lines/filled areas [use / do not use] color. Note that if the “lineauto” option option is specified, this choice will also influence the type of line or filled area drawn on screen: if color is specified, solid colored lines and filled areas will be drawn; if color is turned off, lines will be drawn using black and white line patterns, and gray scales will be used for filled areas.</td>
</tr>
<tr>
<td>barlabelabove/</td>
<td>[Place / Do not place] text value of data above bar in bar graph.</td>
</tr>
<tr>
<td>-barlabelabove</td>
<td></td>
</tr>
<tr>
<td>barlabelinside /</td>
<td>[Place / Do not place] text value of data inside bar in bar graph.</td>
</tr>
<tr>
<td>-barlabelinside</td>
<td></td>
</tr>
<tr>
<td>outlinebars/</td>
<td>[Outline / Do not outline] bars in a bar graph.</td>
</tr>
<tr>
<td>-outlinebars</td>
<td></td>
</tr>
<tr>
<td>outlinearea /</td>
<td>[Outline / Do not outline] areas in an area graph.</td>
</tr>
<tr>
<td>-outlinearea</td>
<td></td>
</tr>
<tr>
<td>barspace /</td>
<td>[Put / Do not put] space between bars in bar graph.</td>
</tr>
<tr>
<td>-barspace</td>
<td></td>
</tr>
<tr>
<td>pielabel /</td>
<td>[Place / Do not place] text value of data in pie chart.</td>
</tr>
<tr>
<td>-pielabel</td>
<td></td>
</tr>
</tbody>
</table>

The options which support the “-” may be proceeded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.
Data labels in bar and pie graphs will only be visible when there is sufficient space in the graph.

Examples

```
graph1.option size(4,4) +inbox color
```

sets GRAPH1 to use a $4 \times 4$ frame enclosed in a box. The graph will use color.

```
graph1.option linepat -color size(2,8) -inbox
```

sets GRAPH1 to use a $2 \times 8$ frame with no box. The graph does not use color, with the lines instead being displayed using patterns.

Cross-references


See also `axis` (p. 197), `datelabel` (p. 237), `scale` (p. 397) and `setelem` (p. 412).

## ordered

### Command || Equation Method

Estimation of ordered dependent variable models.

**Syntax**

Command: `ordered(options) y x1 [x2 x3 ...]`

Equation Method: `equation name.ordered(options) y x1 [x2 x3 ...]`

When used as an equation procedure, `ordered` estimates the model and saves the results as an equation object with the given name.

**Options**

- `d = arg` *(default = "n")*
  Specify likelihood: “n” (normal likelihood function, ordered probit), “l” (logistic likelihood function, ordered logit), “x” (Type I extreme value likelihood function, ordered Gompit).

- `q` *(default)*
  Use quadratic hill climbing as maximization algorithm.

- `r`
  Use Newton-Raphson as maximization algorithm.

- `b`
  Use Berndt-Hall-Hall-Hausman as maximization algorithm.

- `h`
  Quasi-maximum likelihood (QML) standard errors.
If you choose to employ user specified starting values, the parameters corresponding to the limit points must be in ascending order.

**Examples**

```plaintext
ordered(d=1,h) y c wage edu kids
```

estimates an ordered logit model of Y on a constant, WAGE, EDU, and KIDS with QML standard errors. This command uses the default quadratic hill climbing algorithm.

```plaintext
param c(1) .1 c(2) .2 c(3) .3 c(4) .4 c(5) .5
equation eql.binary(s) y c x z
coeff betahat=eql.@coefs
eql.makelimit gamma
```

estimates an ordered probit model of Y on a constant, X, and Z from the specified starting values. The estimated coefficients are then stored in the coefficient vector BETAHAT, and the estimated limit points are stored in the vector GAMMA.
Cross-references


See binary (p. 202) for the estimation of binary dependent variable models. See also makelimits (p. 326).

Redirect printer output or display estimation output.

When used as a command, output redirects printer output. You may specify that any procedure that would normally send output to the printer puts output in a text file, in a Rich Text Format (RTF) file, or into frozen table or graph objects in the current workfile.

When used as a view of an estimation object, output changes the default object view to display the estimation output (equivalent to using results (p. 386)).

Syntax

Command:    output\[(f)\] base_name
Command:    output\[options\] [path\]file_name
Command:    output off
Object View: object_name.output

By default, the output command redirects the output into frozen objects. You should supply a base name after the output keyword. Each subsequent print command will create a new table or graph object in the current workfile, using the base name and an identifying number. For example, if you supply the base name of “OUT”, the first print command will generate a table or graph named OUT1, the second print command will generate OUT2, and so on.

You can also use the optional settings, described below, to redirect table and text output to a text file or all output to an RTF file. If you elect to redirect output to a file, you must specify a filename.

When followed by the optional keyword off, the output command turns off output redirection. Subsequent print commands will be directed to the printer.
### Options

#### Options for output command

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>Redirect all output to frozen objects in the default work-file, using <code>base_name</code>.</td>
</tr>
<tr>
<td>t</td>
<td>Redirect table and text output to a text file. Graphic output will still be sent to the printer.</td>
</tr>
<tr>
<td>r</td>
<td>Redirect all output to a Rich Text Format (RTF) file.</td>
</tr>
<tr>
<td>o</td>
<td>Overwrite file if necessary. If the specified filename for text or RTF output exists, overwrite the file. The default is to append to the file. Only applicable for RTF and text file output (specified using options “t” or “r”).</td>
</tr>
<tr>
<td>c</td>
<td>Command logging. Output both the output, and the command used to generate the output. Only applicable for RTF and text file output (specified using options “t” or “r”).</td>
</tr>
</tbody>
</table>

#### Options for output view

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>Print estimation output for estimation object</td>
</tr>
</tbody>
</table>

### Examples

```plaintext
output print_
```

causes the first print command to generate a table or graph object named `PRINT_1`, the second print command to generate an object named `PRINT_2`, and so on.

```plaintext
output(t) c:\data\results
equation eql.ls(p) log(gdp) c log(k) log(l)
eql.resids(g,p)
output off
```

The second line redirects printing to the RESULTS.TXT file, while the print option of the third line sends the graph output to the printer. The last line turns output redirection off and restores normal printer use.

If instead, the first line read

```plaintext
output(r) c:\data\results
```

all subsequent output would be sent to the RTF file RESULTS.RTF.
The `output` keyword may also be used to change the default view of an estimation object. Entering

    eql.output

displays the estimation output for equation EQ1.

Cross-references
See “Output Redirection” beginning on page 902 of the User’s Guide for further discussion. See also `pon` (p. 565), `poff` (p. 565).

<table>
<thead>
<tr>
<th><strong>override</strong></th>
<th><strong>Model Proc</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies (or merges) overridden exogenous variables and add factors in the active scenario.</td>
<td></td>
</tr>
</tbody>
</table>

Syntax

    Model Proc:    model_name.override(options) ser1 [ser2 ser3 ...]

Follow the keyword with the argument list containing the exogenous variables or add factors you wish to override.

Options

    m
    Merge into (instead of replace) the existing override list.

Examples

    mod1.override fed1 add1
creates an override list containing the variables FED1 and ADD1.

If you then issue the command

    mod1.override fed1
EViews will replace the original exclude list with one containing only FED1. To add overrides to an existing list, use the “m” option:

    mod1.override(m) add1

The override list now contains both series.
Cross-references

See the discussion in “Specifying Scenarios” on page 773 of the User’s Guide. See also Chapter 26 of the User’s Guide for a general discussion of models.

See also model (p. 339), exclude (p. 263) and solveopt (p. 439).

pageappend Command

Append contents of the specified workfile page to the active workfile page.

Syntax

Command: pageappend[options] wfname\pgname[\pgname] [object_list]

where wfname is the name of a workfile that is currently in memory. You may optionally provide the name of a page in wfname that you wish to used as a source, and the list of objects to be read from the source workfile page. If no wfname is provided, EViews will use the default page in the source workfile.

The command appends the contents of the source page to the active page in the default workfile. The target page is first unstructured (if necessary) and its range is expanded to encompass the combined range of the sample from the source workfile page, and the destination page.

The default behavior is to append all series and alpha objects (but not other types) from the source page, but the optional object_list may be provided to specify specific series, or to specify that non series/alpha objects be included. Command options may also be used to modify the list of objects to be included.

Note that since this operation is performed in place, the original workfile page cannot be recovered. We recommend that you consider backing up your original page using page-copy (p. 353).
**Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>smpl = smpl_spec</code></td>
<td>Specifies an optional sample identifying which observations from the source page are to be appended. The default is “@all”.</td>
</tr>
<tr>
<td><code>allobj</code></td>
<td>Specifies that all objects (including non-series and non-alpha objects) should be appended. For objects other than series and alphas, appending involves simply copying the objects from the source page to the destination page. This option may not be used with an explicit <code>object_list</code> specification.</td>
</tr>
<tr>
<td><code>match</code></td>
<td>Specifies that only series and alphas in the append page that match series and alphas of the same name in the active page should be appended. This option may not be used with “allobj” or with an explicit <code>object_list</code> specification.</td>
</tr>
<tr>
<td><code>sufix = suffix_arg</code></td>
<td>Specifies a string to be added to the end of the source object name, if necessary, to avoid name collision when creating a new object in the target page.</td>
</tr>
<tr>
<td><code>obsid = arg</code></td>
<td>Provides the name of a series used to hold the date or observation ID of each observation in the destination workfile.</td>
</tr>
<tr>
<td><code>wfid = arg</code></td>
<td>Provides the name of a (boolean) series to hold an indicator of the source for each observation in the destination workfile (0, if from the destination; 1, if from the source).</td>
</tr>
</tbody>
</table>

**Examples**

The command

```
pageappend updates
```

appends, to the default workfile page, all of observations in all of the series in the active page of the workfile `UPDATES`.

```
pageappend(match, smpl="1999 2003") updates
```

restricts the series to those which match (by name) those in the default workfile page, and restricts the observations to merge to those between 1999 and 2003.
pageappend newdat\page1 income cons
takes only the INCOME and CONS series from the PAGE1 of the NEWDATA workfile, and
appends them to the current workfile page.

pageappend(alltypes, suffix="_1") mydata
appends all objects from MYDATA, merging series with matching names, and renaming
other matching objects by appending "_1" to the end of the name.

Cross-references
See also pagecopy (p. 353).

---

<table>
<thead>
<tr>
<th>Command</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>pagecontract</td>
<td>Contract the active workfile page according to the specified sample.</td>
</tr>
<tr>
<td>Syntax</td>
<td>Pagecontract smpl_spec</td>
</tr>
<tr>
<td>where smpl_spec is a sample specification. Removes observations that are not in specified sample from the active workfile page. Note that because this operation is performed in place, you may wish to backup your original page (see pagecopy (p. 353)) prior to contracting.</td>
<td></td>
</tr>
<tr>
<td>Examples</td>
<td>Pagecontract if income&lt;50000 and race=2</td>
</tr>
<tr>
<td>removes all observations with INCOME values less than or equal to 50000 and RACE not equal to 2.</td>
<td></td>
</tr>
<tr>
<td>Pagecontract 1920 1940 1945 2000</td>
<td></td>
</tr>
<tr>
<td>removes observations for the years 1941 to 1944.</td>
<td></td>
</tr>
<tr>
<td>Cross-references</td>
<td></td>
</tr>
<tr>
<td>See also pagecopy (p. 353).</td>
<td></td>
</tr>
</tbody>
</table>
**pagecopy** | Command

Copies all or part of the active workfile page to a new workfile or to a new page within the default workfile.

**Syntax**

Command: `pagecopy(options) [object_list]`

where the optional `object_list` specifies the workfile objects to be copied. If `object_list` is not provided, all objects will be copied subject to the option restrictions discussed below.

If copying objects to a new page within the default workfile, you may choose to copy series objects (series, alphas, and links) by value or by link (by employing the “bylink” option). If you elect to copy by value, links in the source page will converted to ordinary series and alpha objects when they are copied. If you copy by link, series and alpha objects in the source page are converted to links when copied.

The default is to copy by value. If you copy objects to a new workfile, all data objects must be copied by value.

**Options**

- **bylink**
  - Specifies that series and alpha objects be copied as links to the source page. This option is not available if you use the “wf=” option since linking requires that the destination page be in the same workfile as the source page. Automatically sets the “dataonly” option so that only series, alphas, links, and valmaps will be copied.

- **smpl = smpl_spec**
  - Specifies an optional sample identifying which observations from the source page are to be copied. The default is “@all”.

- **rndobs = integer**
  - Copy only a random subsample of `integer` observations from the specified sample. Not available with “bylink” or “rnpct”.

- **rnpct = arg**
  - Copy only a random subsample of `arg` (a number between 0 and 1) of the specified sample. Not available with “bylink” or “rndobs”.
Examples

```
pagecopy(page=allvalue)
```
will first create a new UNTITLED workfile, with a page ALLVALUE that has the same structure as the current page. Next, all of the objects in the active workfile page will be copied into the new page, with the series objects copied by value. In contrast,

```
pagecopy(bylink, page=alllink)
```
will instead create a page ALLLINK in the existing workfile, and will copy all series objects by creating links in the new page.

```
pagecopy(page=partcopy, bylink, smpl="1950 2000 if gender="male"", a* *z
```
will create a new page named PARTCOPY in the existing workfile containing the specified subsample of observations, and will copy all series objects in the current page beginning with the letter “A” or ending with the letter “Z”. The objects will be copied by creating links in the new page.

```
pagecopy(page=rndcopy, smpl="1950 2000 if gender="male"",
         rndobs=200, dataonly, nolinks)
```
creates a new workfile and page RNDCOPY containing a 200 observation random sample from the specified subsample. Only series and alpha objects will be copied by value from the source page.

Cross-references

See also pageappend (p. 350).
Create a new page in the default workfile. The new page becomes the active page.

Syntax

Command: `pagecreate(options) freq start_date end_date [num_cross_sections]`

Command: `pagecreate(options) u num_observations`

Command: `pagecreate(byid[, options]) identifier_list`

The first two forms of this command are the same as `wfcreate` (p. 491) except that here, we create a new page in the default workfile, rather than creating a new workfile.

The first form of the command is used to create a regular frequency page with the specified frequency, start and end date. If you include the optional `num_cross_sections`, EViews will create a balanced panel page using integer identifiers for each of the cross-sections. Note that more complex panel structures may be created using `pagestruct` (p. 363).

The second form of the command is used to create an unstructured workfile with the specified number of observations.

The last form of the command, which uses IDs, creates a new page from the active workfile page using the unique values of the specified identifiers in the specified sample. The `identifier_list` should include all desired ID series.

If you wish to use ID series as date IDs, you should enclose the series (or the date ID component series) inside parentheses with a special “@DATE” keyword identifier. If you wish to use the existing date ID of the source workfile page, you may use “@DATE” without arguments.

Options

<table>
<thead>
<tr>
<th>byid</th>
<th>Create new page using the identifier list and optional sample in the default workfile page.</th>
</tr>
</thead>
<tbody>
<tr>
<td>smpl = smpl_spec</td>
<td>Specifies an optional sample identifying which observations to use when creating a page using the “byid” option. The default is “@all”.</td>
</tr>
<tr>
<td>page = page_name</td>
<td>Optional name for the newly created page. If not provided, EViews will use the next available name of the form given by the page structure (e.g., “Undated”, “Annual”, “Daily5”).</td>
</tr>
</tbody>
</table>
Examples

The two commands

```
pagecreate(page="annual") a 1950 2005
pagecreate(page="unstruct") u 1000
```

create new pages in the existing workfile. The first page is an annual page named ANNU-

AL, containing annual data from 1950 to 2005; the second is an unstructured page named

UNSTRUCT containing 1000 observations.

```
pagecreate(page="mypanel") a 1935 1954 10
```

creates a new workfile page named MYPANEL containing a 10 cross-section annual panel

for the years 1935 to 1954.

```
pagereate(byid, page="stateind") state industry
```

creates a new page named STATEIND using the distinct STATE/INDUSTRY values in the

active page.

```
pagereate(byid, page="stateyear") state @date(year)
pagereate(byid, page="statemonth") @date(year, month)
```

use the YEAR, and the YEAR and MONTH series to form date identifiers that will be used

in creating the new pages.

```
pagereate(byid, smpl="if sex=1) crossid @date
```

creates a new page using the CROSSID and existing date ID values of the active workfile

page. Note that only values in the subsample defined by "@all if sex = 1" are used to deter-

mine the unique values.

Cross-references

See also `wfcreate` (p. 491) and `pagedelete` (p. 356).

<table>
<thead>
<tr>
<th>pagedelete</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Delete the named page from the default workfile.</td>
</tr>
</tbody>
</table>

**Syntax**

Command:  

`pagedelete pgname`

where `pgname` is the name of a page in the default workfile.
Examples

```
pagedelete page1
```

Cross-references

See also `pagecreate` (p. 355).

---

<table>
<thead>
<tr>
<th>pageload</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pageload</strong></td>
<td><strong>Command</strong></td>
</tr>
</tbody>
</table>

Load one or more new pages in the default workfile.

**Syntax**

Command: `pageload [path]\workfile_name`

Command: `pageload(options) [path]\datasource [\@keep keep_list] [\@drop drop_list] [\@keepmap keepmap_list] [\@dropmap dropmap_list] [\@selectif condition]`

Command: `pageload(options) table_spec [\@keep keep_list] [\@drop drop_list] [\@keepmap keepmap_list] [\@dropmap dropmap_list] [\@selectif condition]`

where `workfile_name` is the name of a workfile on disk. If no path is provided, EViews will look for the file in the default directory. If a page is loaded with a name that matches an existing page, EViews will rename the new page to the next available name (e.g., “INDIVID” will be renamed “INDIVIDI1”.

**Options**

```
type = arg, t = arg
```

Examples

pageload "c:\my documents\data\panel1"
loads the workfile PANEL1.WF1 from the specified directory. All of the pages in the workfile will be loaded as new pages into the current workfile.

pageload f.wf1
loads all of the pages in the workfile F.WF1 located in the default directory.

Cross-references
See also \texttt{wfopen} (p. 492) and \texttt{pagecreate} (p. 355).

\begin{tabular}{|l|l|}
\hline
\textbf{pagerename} & \textbf{Command} \\
\hline
\end{tabular}

Load the specified workfile as one or more new pages in the default workfile.

Syntax

Command: \texttt{pagerename old\_name new\_name}
renames the page \textit{old\_name} in the default workfile to \textit{new\_name}. Page names are case-insensitive for purposes of comparison, even though they are displayed using the input case.

Examples

\texttt{pagerename Page1 StateByYear}

Cross-references
See also \texttt{pagecreate} (p. 355).

\begin{tabular}{|l|l|}
\hline
\textbf{pagesave} & \textbf{Command} \\
\hline
\end{tabular}

Save the active page in the default workfile as an EViews workfile (.wf1 file), or as a foreign data source.

Syntax

Command: \texttt{pagesave(options) [path]\filename}
Command: \texttt{pagesave(options) [path]\datasource [@keep keep\_list] [@drop drop\_list] [@keepmap keepmap\_list] [@dropmap dropmap\_list]}
pagesave — 359

Command:

```
(pagesave (options) table_spec [@keep keep_list] [@drop drop_list]
[@keepmap keepmap_list] [@dropmap dropmap_list] [@smpl
smpl_spec])
```

saves the active page in the specified directory using filename. By default, the page is
saved as an EViews workfile, but options may be used to save all or part of the page in a
foreign file or data source.

Options

- `type = arg, t = arg` Optional type specification. Access database file
  (“access”), Aremos–TSD file (“a”, “aremso”,
  “tsd”), Binary file (“binary”), EViews database file
  (“e”, “evdb”), Excel file (“excel”), Gauss dataset
  file (“gauss”), GiveWin/PeGive file (“g”, “give”),
  HTML file/page (“html”), ODBC database
  (“odbc”), ODBC Dsn file (“dsn”), ODBC query file
  (“msquery”), MicroTSP workfile (“dos”,
  “microtsp”), MicroTSP Macintosh workfile
  (“mac”), Rats file (“r”, “rats”), Rats portable/Troll
  file (“tl”, “trl”), SAS program file (“sasprog”), SAS
  transport file (“sasxport”), SPSS file (“spss”), SPSS
  portable file (“spssport”), Stata file (“stata”), Text
  file (“text”), TSP portable file (“t”, “tsp”).

- `mode = create` Create new file only; error on attempt to overwrite.

- `maptype = arg` Write selected maps as: “n” (numeric), “c” (char-
  acter), “b” (both numeric and character).

- `mapval` Write mapped values for series with attached value
  labels.

The following options may be required when connecting to a remote server:

- `s = server_id, server = server_id` Server name.

- `u = user, username = user` Username.

- `p = pswd, password = pswd` Password.
Examples

```bash
pagesave new_wf
```
saves the EViews workfile NEW_WF.WF1 in the default directory.

```bash
pagesave "c:\documents and settings\my data\consump"
```
saves the workfile CONSUMP.WF1 in the specified path.

Cross-references
See also `wfsave` (p. 493).

### pageselect Command

Make the specified page in the default workfile the active page.

**Syntax**

Command: `pageselect pgname`

where `pgname` is the name of a page in the default workfile.

**Examples**

```bash
pageselect page2
```
change the active page to PAGE2.

Cross-references
See also `wfselect` (p. 495).

### pagemake Command

Create a panel structured workfile page using series, alphas, or links from the default workfile page (convert repeated series to repeated observations).

Series in the panel workfile may be created by stacking series, alphas, or links, whose names contain a pattern (series with names that differ only by a “stack identifier”), or by repeating a single series, alpha, or link, for each value in a set of identifiers.
Syntax

Command: `pagestack(options) stack_id_list [@ series_to_stack]`

Command: `pagestack(options) pool_name [@ series_to_stack]`

Command: `pagestack(options) series_name_pattern [@ series_to_stack]`

The resulting panel workfile will use the identifiers specified in one of the three forms of the command:

- `stack_id_list` includes a list of the ID values (e.g., “US UK JPN”).
- `pool_name` is the name of a pool object that contains the ID values.
- `series_name_pattern` contains an expression from which the ID values may be determined. The pattern should include the “?” character corresponding to the parts of the series names containing the stack identifiers. For example, if “CONS?” is the `series_name_pattern`, EViews will find all series with names beginning with “CONS” and will extract the IDs from the trailing parts of the observed names.

The `series_to_stack` list may contain two types of entries: stacked series (corresponding to sets of series, alphas, and links whose names contain the stack IDs) and simple series (other series, alphas, and links).

To stack a set of series whose names differ only by the stack IDs, you should enter an expression that includes the “?” character in place of the IDs. You may list the names of a single stacked series (e.g., “GDP?” or “CONS”), or you may use expressions containing the wildcard character “*” (e.g., “*?” and “?C?”) to specify multiple series. By default, the stacked series will be named using the base portion of the series name (i.e., if you specify “?CONS” the stacked series will be named “CONS”) and will contain the values of the individual series stacked one on top of another. If one of the series associated with a particular stack ID does not exist, the corresponding stacked values will be assigned the value NA.

Individual (simple) series may also be stacked. You may list the names of individual simple series (e.g., “POP INC”), or you can specify your series using expressions containing the wildcard character “*” (e.g., “*”, “*C”, and “F*”). A simple series will be stacked on top of itself, once for each ID value. If the target workfile page is in the same workfile, EViews will create a link in the new page, otherwise, the stacked series will contain repeated copies of the original values.

When evaluating wildcard expressions, stacked series take precedence over simple series. This means that simple series wildcards will be processed using the list of series not already included as a stacked series.

If the `series_to_copy` list is not specified, the expression “*? *”, is assumed.
Options

? = name_pattern, idreplace = name_pattern

Specifies the characters to use instead of the identifier, “?”, in naming the stacked series. By default, the pattern is blank, indicating, for example, that the stacked series corresponding to the pattern “GDP?” should be named “GDP” in the stacked workfile page. If pattern is set to “STK”, the stacked series will be named GDP-STK in the destination page.

interleave

Interleave the observations in the destination workfile (put all of the series values for the first source observation, followed by the values for the second observation, and so on). The default is to stack observations by identifier (stack series on top of each other).

wf = wf_name

Optional name for the new workfile. If not provided, EViews will create a new page in the default workfile.

page = page_name

Optional name for the page in the destination workfile. If not provided, EViews will use the next available name of the form “Untitled”.

Examples

Consider a workfile that contains the seven series: GDPUS, GDPUK, GDPJPN, CONSUS, UKCONS, JPNCONS, FRCONS, along with the ordinary series WORLDGDP.

pagestack @ *?

creates a new, panel structured workfile page containing the series GDP and CONS, containing three GDP? and three ?CONS series stacked on top of each other. Note that CONSFR and WORLDGDP will not be copied/stacked.

We may specify the stacked series list explicitly. For example,

pagestack(page=stackctry) gdp? @ gdp? ?cons

first determines the stack IDs from the names of series beginning with “GDP”, they stacks the GDP? and ?CONS series. Note that this latter example also names the new workfile page STACKCTRY.

If we have a pool object, we may use it to specify the IDs:
Here, the panel structured page STACKCTRY will be created in the workfile NEWWF.

Simple series may be specified by adding them to the stack list, either directly, or using wildcard expressions. Both commands

\[ \text{pagestack} \left( \text{wf=ne\text{w}wf, page=stackctry} \right) \text{ countrypool @ gdp? cons?} \]

stack the various GDP? and CONS series on top of each other, and stack the simple series GDPFR and WORLDGDP on top of themselves. In the first case, we create a new panel structured page in the same workfile containing the stacked series GDP and CONS and the link objects GDPFR and WORLDGDP. In the latter case, the new panel page in the workfile ALTWF will contain the stacked GDP and CONS, and series named GDPFR and WORLDGDP containing the repeated values of the series.

The following two commands are equivalent:

\[ \text{pagestack us uk jpn @ gdp? const? *} \]

Here, every series, alpha, and link in the source workfile is stacked and copied to the destination workfile, either by stacking different series containing the stack_id or by stacking simple series on top of themselves.

The "?=" option may be used to prevent name collision.

\[ \text{pagestack(?="sk") us uk jpn @ gdp? gdp} \]

stacks GDPUS, GDPUK and GDPJPN into a series called GDPSK and repeats the values of the simple series GDP in the destination series GDP.

Cross-references
For additional discussion, see “Stacking a Workfile” on page 230 in the User’s Guide. See also pageunstack (p. 366).

<table>
<thead>
<tr>
<th>pagestruct</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Assign a structure to the active workfile page.</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Syntax**

Command:  
\[ \text{pagestruct(options) [id_list]} \]

Command:  
\[ \text{pagestruct(options) *} \]
where \textit{id\_list} is an (optional) list of ID series. The “*” may be used as shorthand for the indices currently in place.

If an \textit{id\_list} is provided, EViews will attempt to auto determine the workfile structure. EViews auto-determination may be overridden through the use of options.

If you do not provide an \textit{id\_list}, the workfile will be restructured as a regular frequency workfile. In this case either the “none” or the “freq = ” and “start = ” options described below must be provided.

\textbf{Options}

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>Remove any existing workfile structure.</td>
</tr>
<tr>
<td>freq = \textit{arg}</td>
<td>Specifies a regular frequency; if not specified EViews will auto-determine the frequency.</td>
</tr>
<tr>
<td>start = \textit{arg}</td>
<td>Start date of the regular frequency structure; if not specified, defaults to “@FIRST”. Legal values for \textit{arg} are described below.</td>
</tr>
<tr>
<td>end = \textit{arg}</td>
<td>End date of the regular frequency structure; if not specified, defaults to “@LAST”. Legal values for \textit{arg} are described below.</td>
</tr>
<tr>
<td>regular, reg</td>
<td>When used with a date ID, this option informs EViews to insert observations (if necessary) to remove gaps in the date series so that it follows the calendar. The option has no effect unless a date index is specified.</td>
</tr>
</tbody>
</table>
balance = arg, bal = arg

Balance option (for panel data) describing how EViews should handle data that are unbalanced across index (cross-section) groups.

The arg should be formed using a combination of “s” (starts), “e” (ends), and “m” (middles), e.g., “balance = se”, “balance = m”.

If balancing starts (arg contains “s”), EViews will (if necessary) add observations to your workfile so that each cross-section begins at the same observation (the earliest date or observation observed).

If balancing ends (arg contains “e”), EViews will add any observations required so that each cross-section ends at the same point (the last date or observation observed).

If balancing middles (arg contains “m”) EViews will add observations to ensure that each cross-section has consecutive observations from the earliest date or observation for the cross-section to the last date or observation for the cross-section.

Note that “balance = m” is implied by the “regular” option.

create

Allow creation of a new series to serve as an additional ID series when duplicate ID values are found in any group. EViews will use this new series as the observation ID.

The default is to prompt in interactive mode and to fail in programs.

dropna

Specifies that observations which contain missing values (NAs or blank strings) in any ID series (including the date or observation ID) will be removed from the workfile. If the “dropna” option is not provided and NAs are found, EViews will prompt in interactive mode and fail in programs.

dropbad

Specifies that observations for which any of the date index series contain values that do not represent dates be removed from the workfile. If the “dropbad” option is not provided and bad dates are present, EViews will prompt in interactive mode and fail in programs.
Legal values for start and end dates include date literals (actual dates or periods), e.g. – “1950q1” and “2/10/1951”, or “@FIRST” or “@LAST” (first and last observed dates in the date index). Date literals must be used for the “start = ” option when restructuring to a regular frequency.

In addition offsets from these date specifications can be specified with a “+” or “−” followed by an integer: “@FIRST-5”, “@LAST + 2”, “1950m12 + 6”. Offsets are most often used when resizing the workfile to add or remove observations from the endpoints.

Examples

```
pagestruct state industry
```

structures the workfile using the IDs in the STATE and INDUSTRY series.

A date index series (or series used to form a date index) should be identified by the tagging the series using the keyword “@DATE” and surrounded by parentheses. For example:

```
pagestruct state @date(year)
pagestruct(regular) @date(year, month)
```

A “∗” may be used to indicate the indices defined in the current structure.

```
pagestruct(end=@last+5) *
```

When you omit the `id_list`, EViews will attempt to restructured the workfile to a regular frequency. In this case you must either provide the “freq = ” and “start = ” options to identify the regular frequency structure, or you must specify “none” to indicate that there is no structure:

```
pagestruct(freq=a, start=1950)
pagestruct(none)
```

Cross-references

For extensive discussion, see “Structuring a Workfile” beginning on page 187 in the User’s Guide.

| pageunstack | Command |

Unstack workfile page (convert repeated observations to repeated series).

Takes series objects (series, alphas, or links) in the default workfile page and breaks them into multiple series (or alphas), one for each distinct value found in a user supplied list of series objects, in a new workfile page. Typically used on a page with a panel structure.
Syntax

Command: \texttt{pageunstack} (options) stack\_id obs\_id [@ series\_to\_unstack]

where \texttt{stack\_id} is a single series containing the unstacking ID values used to create individual series, \texttt{obs\_id} is a series containing the observation IDs, and \texttt{series\_to\_unstack} is an optional list of series objects to be copied to the new workfile.

Options

- \texttt{namepat = name\_pattern} Specifies the pattern from which unstacked series names are constructed, where “**” indicates the original series name and “?” indicates the stack ID.
  
  By default the \texttt{name\_pattern} is “**?” indicating, for example, that if we have the IDs “US”, “UK”, “JPN”, the unstacked series corresponding to the series GDP should be named “GDPUS”, “GDPUK”, “GDPJPN” in the unstacked workfile page.

- \texttt{wf = wf\_name} Optional name for the new workfile. If not provided, EViews will create a new page in the default workfile.

- \texttt{page = page\_name} Optional name for the page in the destination workfile. If not provided, EViews will use the next available name of the form “Untitled”.

Examples

Consider a workfile that contains the series GDP and CONS which contain the values of Gross Domestic Product and consumption for three countries stacked on top of each other. Suppose further there is an alpha object called COUNTRY containing the observations “US”, “UK”, and “JPN” which identify which country each observation of GDP and CONS come from. Finally suppose there is a date series DATEID which identifies the date of each observation of GDP and CONS. The command

\texttt{pageunstack country dateid @ gdp cons}

would create a new workfile page using the dates and frequency of the values found in DATEID and containing the 6 series GDPUS, GDPUK, GDPJPN, CONSUS, CONSUK, and CONSJPN.

Typically the source workfile described above would be structured as a dated panel with the cross-section ID series COUNTRY and the date ID series DATEID and the command
would be equivalent to the one above.

The stack_id may be an ordinary series or an alpha series, that uniquely identify the groupings to unstuck by obs_id may be one or more ordinary series or alpha series, the combination of which uniquely identify each observation in the new workfile.

You may provide an explicit list of series to unstack following an "@" immediately after the obs_id. Wildcards may be used in this list. For example:

```
pagenestack country dateid @ g* c*
```

unstacks all series and alphas that have names that begin with “G” or “C”.

If no series_to_unstack list is provided, all series in the source workfile will be unstacked. Thus, the commands

```
pagenestack country dateid @ *
pagenestack country dateid
```

are equivalent.

By default, series are named in the destination workfile page by appending the stack_id values to the original series name. Letting "*" stand for the original series name and "?" for the stack_id, names are constructed as "*?". This default can be changed using the "namepat=" option. For example:

```
pagenestack (namepat="?_*") country dateid @ gdp cons
```

creates the series US_GDP, UK_GDP, JPN_GDP, etc.

**Cross-references**

For additional discussion and examples, see “Unstacking a Workfile” on page 223 of the User’s Guide. See also pagenestack (p. 360).

## Set parameter values.

Allows you to set the current values of coefficient vectors. The command may be used to provide starting values for the parameters in nonlinear least squares, nonlinear system estimation, and (optionally) ARMA estimation.

### Syntax

Command:  
```plaintext
command coef_name1 number1 [coef_name2 number2 coef_name3]
```
Simply list, in pairs, the names of the coefficient vector and its element number followed by the corresponding starting values. You can use param to change the starting value of all, or just a subset, of the parameters in your equation.

Examples

\[
\text{param } c(1) .2 \ c(2) .1 \ c(3) .5
\]

resets the first three values of the coefficient vector C.

\[
\text{coef(3) beta}
\]

\[
\text{param beta(2) .1 beta(3) .5}
\]

The first line declares a coefficient vector BETA of length 3 filled with zeros. The second line resets the second and third elements of BETA to 0.1 and 0.5, respectively.

Cross-references

See “Starting Values” on page 455 of the User’s Guide for a discussion of setting initial values in nonlinear estimation.

**pcomp**

Group View

Principal components analysis.

**Syntax**

Group View:  \text{group \_name.pcomp(options) [ser1 ser2 ...]}  

Enter the name of the group followed by a period, the keyword and optionally, a list of \( k \) names to store the first \( k \) principal components. Separate each name in the list with a space and do not list more names than the number of series in the group.

**Options**

- \text{cor (default)}  
  Use sample correlation matrix.

- \text{cov}  
  Use sample covariance matrix.

- \text{dof}  
  Degrees of freedom adjustment if cov option used.  
  Default is no adjustment (which divides by \( n \) rather than \( n - 1 \)).
Examples

\begin{verbatim}
group g1 x1 x2 x3 x4
freeze(tab1) g1.pcomp(cor, eigval=v1, eigvec=m1) pc1 pc2
\end{verbatim}

The first line creates a group named G1 with four series X1, X2, X3, X4. The second line stores the first two principal components in series named PC1 and PC2 using the sample correlation matrix. The output view is stored in a table named TAB1, the eigenvalues in a vector named V1, and the eigenvectors in a matrix named M1.

Cross-references

See “Principal Components” on page 360 of the User’s Guide for further discussion.

### Syntax

- **Command:**
  - \texttt{pie(options) arg1 arg2 arg3 ...}
- **Object View:**
  - \texttt{object\_name.pie(options)}
- **Graph Proc:**
  - \texttt{graph\_name.pie(options)}

To use \texttt{pie} as a command, simply list the name of one or more series or groups, or a matrix object to include in the pie chart. You may also change the existing graph type by using \texttt{pie} as a proc. Simply list the graph name, followed by a period, and the \texttt{pie} keyword.
Options

\[ o = \text{graph\_name} \]  Use appearance options from the specified graph.

\[ t = \text{graph\_name} \]  Use appearance options and copy text and shading from the specified graph.

\[ p \]  Print the pie chart.

Panel options

The following options apply when graphing panel structured data.

\[ \text{panel} = \text{arg} \]  (default taken from global settings)

Panel data display: “stack” (stack the cross-sections), “individual” or “1” (separate graph for each cross-section), “combine” or “c” (combine each cross-section in single graph; one time axis), “mean” (plot means across cross-sections), “mean1se” (plot mean and +/- 1 standard deviation summaries), “mean2sd” (plot mean and +/- 2 s.d. summaries), “mean3sd” (plot mean and +/- 3 s.d. summaries), “median” (plot median across cross-sections), “med25” (plot median and +/- .25 quantiles), “med10” (plot median and +/- .10 quantiles), “med05” (plot median +/- .05 quantiles), “med025” (plot median +/- .025 quantiles), “med005” (plot median +/- .005 quantiles), “med\text{-}m\text{xm}n” (plot median, max and min).

Examples

\[ \text{smpl 1990 1995} \]

\[ \text{pie cons inv gov} \]

shows six pie charts, each divided into CONS, INV, and GOV.

\[ \text{graph gr1.line cons inv gov} \]

\[ \text{gr1.pie} \]

creates a line graph GR1 and then changes the graph to a pie chart.

Cross-references


See also \texttt{graph} (p. 288) for graph declaration and other graph types.
plot

Command

Line graph.
Provided for backward compatibility. See line (p. 306).

pool

Object Declaration

Declare pool object.

Syntax
Command: pool name id1 [id2 id3 ...]
Follow the pool keyword with a name for the pool object. You may optionally provide the identifiers for the cross-section members of the pool object. Pool identifiers may be added or removed at any time using add (p. 179) and drop (p. 255).

Examples
pool zoo1 dog cat pig owl ant
Declares a pool object named ZOO1 with the listed cross-section identifiers.

Cross-references
See add (p. 179) and drop (p. 255). See also ls (p. 315) for details on estimation using a pool object.

predict

Equation View

Prediction table for binary and ordered dependent variable models.
The prediction table displays the actual and estimated frequencies of each value of the discrete dependent variable.

Syntax
Equation Proc: eq_name.predict(options)
For binary models, you may optionally specify how large the estimated probability must be to be considered as a success \((y = 1)\). Specify the cutoff level as an option in parentheses after the keyword `predict`.

**Options**

\[
\begin{array}{ll}
  n \text{ (default = .5)} & \text{Cutoff probability for success prediction in binary models (between 0 and 1).} \\
p & \text{Print the prediction table.}
\end{array}
\]

**Examples**

\[
equation eq1.binary(d=l) \text{ work c edu age race} \\
eq1.predict(0.7)
\]

Estimates a logit and displays the expectation-prediction table using a cutoff probability of 0.7.

**Cross-references**

See “Binary Dependent Variable Models” on page 591 of the *User’s Guide* for a discussion of binary models, and “Expectation-Prediction (Classification) Table” on page 599 of the *User’s Guide* for examples of prediction tables.

<table>
<thead>
<tr>
<th>print</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>print((options)) object1 [object2 object3 …]</td>
</tr>
<tr>
<td></td>
<td>print((options)) object_name.view_command</td>
</tr>
</tbody>
</table>

Sends views of objects to the default printer.

**Syntax**

- \( \text{Command:} \) \( \text{print(\(options\)) object1 [object2 object3 …]} \)
- \( \text{Command:} \) \( \text{print(\(options\)) object_name.view_command} \)

`print` should be followed by a list of object names or a view of an object to be printed. The list of names must be of the same object type. If you do not specify the view of the object, `print` will print the default view for each object.

**Options**

\[
\begin{array}{ll}
p & \text{Override the default output orientation (set by Print Setup) and print in portrait.} \\
l & \text{Override the default output orientation (set by Print Setup) and print in landscape.}
\end{array}
\]
Examples

```plaintext
print gdp log(gdp) d(gdp) @pch(gdp)
```

sends a table of GDP, log of GDP, first difference of GDP, and percentage change of GDP to the printer.

```plaintext
print graph1 graph2 graph3
```

prints three graphs on a single page.

To merge the three graphs, realign them in one row, and print in landscape:

```plaintext
graph mygra.merge graph1 graph2 graph3
mygra.align(3,1,1)
print(l) mygra
```

To estimate the equation EQ1 and send the output view to the printer.

```plaintext
print eq1.ls gdp c gdp(-1)
```

Cross-references

See “Print Setup” beginning on page 903 of the User’s Guide for a discussion of print options and the Print Setup dialog.

See output (p. 347) for print redirection.

<table>
<thead>
<tr>
<th>probit</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimation of binary dependent variable models with normal errors.</td>
<td></td>
</tr>
<tr>
<td>Equivalent to “binary(d=n)”.</td>
<td></td>
</tr>
<tr>
<td>See binary (p. 202).</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>program</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declare a program.</td>
<td></td>
</tr>
</tbody>
</table>

Syntax

```plaintext
Command:     program [path\]prog_name
```

Enter a name for the program after the `program` keyword. If you do not provide a name, EViews will open an untitled program window. Programs are text files, not objects.
Examples

    program runreg

opens a program window named RUNREG which is ready for program editing.

Cross-references

See Chapter 6, “EViews Programming”, on page 83 of the Command and Programming Reference for further details and examples of writing EViews programs.

See also open (p. 342).

<table>
<thead>
<tr>
<th>qqplot</th>
<th>Group View</th>
<th>Series View</th>
</tr>
</thead>
</table>

Quantile-quantile plots.

Plots the (empirical) quantiles of a series against the quantiles of a theoretical distribution or the quantiles of another series. You may specify the theoretical distribution and/or the method used to compute the empirical quantiles as options.

Syntax

    Object View:     object_name.qqplot(options)

Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Plot against the quantiles of a normal distribution.</td>
</tr>
<tr>
<td>u</td>
<td>Plot against the quantiles of a uniform distribution.</td>
</tr>
<tr>
<td>e</td>
<td>Plot against the quantiles of an exponential distribution.</td>
</tr>
<tr>
<td>l</td>
<td>Plot against the quantiles of a logistic distribution.</td>
</tr>
<tr>
<td>x</td>
<td>Plot against the quantiles of an extreme value distribution.</td>
</tr>
<tr>
<td>s=series_name</td>
<td>Plot against the (empirical) quantiles of the specified series.</td>
</tr>
<tr>
<td>q=arg (default =“r”)</td>
<td>Compute quantiles using the definition: “b” (Blom), “r” (Rankit-Cleveland), “o” (simple fraction), “t” (Tukey), “v” (van der Waerden).</td>
</tr>
<tr>
<td>p</td>
<td>Print the QQ-plot.</td>
</tr>
</tbody>
</table>
Examples

equation eq1.binary(d=1) work c edu age race
eq1.makeresid(o) res1
res1.qqplot(l)

estimates a logit, retrieves the residuals, and plots the quantiles of the residuals against those from the logistic distribution. If the error distribution is correctly specified, the QQ-plot should lie on a straight line.

Cross-references

See “Quantile-Quantile” on page 367 of the User’s Guide for a discussion of QQ-plots.

See also cdfplot (p. 214).

qstats

Multivariate residual autocorrelation Portmanteau tests.

Syntax

Var View: var_name.qstats(h,options)

You must specify the highest order of lag \( h \) to test for serial correlation. \( h \) must be larger than the VAR lag order.

Options

name = arg  Save \( Q \)-statistics in named matrix object. The matrix has two columns, where the first column is the unmodified and the second column is the modified \( Q \)-statistics.

p  Print the Portmanteau test results.

Examples

var var1.ls 1 6 lgdp lm1 lcpi
show var1.qstats(l2,name=q)

The first line declares and estimates a VAR. The second line displays the portmanteau tests for lags up to 12 and stores the \( Q \)-statistics in a matrix named Q.
Cross-references

See “Diagnostic Views” on page 694 of the User’s Guide for a discussion of the portmanteau tests and other VAR diagnostics.

See also arlm (p. 193) for a related multivariate residual serial correlation LM test.

<table>
<thead>
<tr>
<th>read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
</tr>
</tbody>
</table>

Import data from a foreign disk file into an existing EViews workfile.

May be used to import data into an existing workfile from a file on disk. Used as a command, it imports data into series in the workfile. When used as a procedure, read imports data directly into pool and matrix objects.

Syntax

Command: \[read\](options) \[\{path\\}file_name name1 \{name2 name3 ...\}\]
Command: \[read\](options) \[\{path\\}file_name n\]
Coef Proc: \[coef\_name.read\](options) \[\{path\\}file_name\]
Pool Proc: \[pool\_name.read\](options) \[\{path\\}file_name pool_ser1 \{pool_ser2 pool_ser3 ...\}\]
Matrix Proc: \[matrix\_name.read\](options) \[\{path\\}file_name\]

You must supply the name of the source file. If you do not include the optional path specification, EViews will look for the file in the default directory. The input specification follows the source file name. Path specifications may point to local or network drives. If the path specification contains a space, you may enclose the entire expression in double quotation marks.

In the command proc form of read, there are two ways to specify the input series. First, you may list the names of the series in the order they appear in the file. Second, if the data file contains a header line for the series names, you may specify the number \(n\) of the series in the file instead of a list of names; EViews will name the series as given in the header line. If you specify a number and the data file does not contain a header line, EViews will name the series as SER01, SER02, SER03, and so on.
For the pool proc form of `read`, you must provide a list of ordinary or pool series.

**Options**

**File type options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>t=dat, txt</code></td>
<td>ASCII (plain text) files.</td>
</tr>
<tr>
<td><code>t=wk1, wk3</code></td>
<td>Lotus spreadsheet files.</td>
</tr>
<tr>
<td><code>t=xls</code></td>
<td>Excel spreadsheet files.</td>
</tr>
</tbody>
</table>

If you do not specify the “t” option, EViews uses the file name extension to determine the file type. If you do specify the “t” option, then the file name extension will not be used to determine the file type.

**Options for ASCII text files**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>na=text</code></td>
<td>Specify text for NAs. Default is “NA”.</td>
</tr>
<tr>
<td><code>byper</code></td>
<td>Panel data organized by date/period. Default is data organized by cross-section (only for pool read).</td>
</tr>
<tr>
<td><code>bycross</code></td>
<td>Panel data organized by cross-section (only for pool read). (default)</td>
</tr>
<tr>
<td><code>t</code></td>
<td>Read by series (or transpose the data for matrix objects). Default is to read by observation with series in columns.</td>
</tr>
<tr>
<td><code>d=t</code></td>
<td>Treat tab as delimiter (note: you may specify multiple delimiter options). The default is “d=c” only.</td>
</tr>
<tr>
<td><code>d=c</code></td>
<td>Treat comma as delimiter.</td>
</tr>
<tr>
<td><code>d=s</code></td>
<td>Treat space as delimiter.</td>
</tr>
<tr>
<td><code>d=a</code></td>
<td>Treat alpha numeric characters as delimiter.</td>
</tr>
<tr>
<td><code>custom=symbol</code></td>
<td>Specify symbol/character to treat as delimiter.</td>
</tr>
<tr>
<td><code>mult</code></td>
<td>Treat multiple delimiters as one.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>Series names provided in file.</td>
</tr>
<tr>
<td><code>label=integer</code></td>
<td>Number of lines between the header line and the data. Must be used with the “name” option.</td>
</tr>
<tr>
<td><code>rect</code></td>
<td>Treat file layout as rectangular.</td>
</tr>
<tr>
<td><code>noret</code></td>
<td>Do not treat file layout as rectangular.</td>
</tr>
</tbody>
</table>
Options for spreadsheet (Lotus, Excel) files

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>skipcol</td>
<td>Number of columns to skip. Must be used with the “rect” option.</td>
</tr>
<tr>
<td>skiprow</td>
<td>Number of rows to skip. Must be used with the “rect” option.</td>
</tr>
<tr>
<td>comment</td>
<td>Specify character/symbol to treat as comment sign.</td>
</tr>
<tr>
<td>singlequote</td>
<td>Do not treat strings as NA; simply drop them.</td>
</tr>
<tr>
<td>dropstrings</td>
<td>Treat numbers in parentheses as negative numbers.</td>
</tr>
<tr>
<td>allowcomma</td>
<td>Allow commas in numbers (note that using commas as a delimiter takes precedence over this option).</td>
</tr>
<tr>
<td>currency</td>
<td>Specify symbol/character for currency data.</td>
</tr>
</tbody>
</table>

Example of options:

```
read(t=dat,na=.) a:\mydat.raw id lwage hrs
```

reads data from an ASCII file MYDAT.RAW in the A drive. The data file is listed by observation, NA is coded as a “.” (dot or period), and there are three series, which are to be named ID, LWAGE, HRS in this order from left to right.
read(a2, s=sheet3) cps88.xls 10

reads data from an Excel file CPS88 in the default directory. The data are organized by
observation, the upper left data cell is A2, and 10 series are read from a sheet named
SHEET3.

read(a2, s=sheet2) "\\network\dr 1\cps91.xls" 10

reads the Excel file CPS91 from the network drive specified in the path.

Cross-references

See “Frequency Conversion” on page 96 of the User’s Guide for a discussion and examples
of importing data from external files.

See also write (p. 498).

rename

Rename an object in the active workfile or database.

Syntax

Command: rename old_name new_name

After the rename keyword, first list the old name, followed by the new name. Note that
the names may include matching wildcard patterns.

Examples

rename temp_u u2

renames an object named TEMP_U as U2.

rename aa::temp_u aa::u2

renames TEMP_U to U2 in database AA.

Cross-references

with objects in EViews.
**representations**

Display text of specification for equation, pool, and var objects.

**Syntax**

Object View: object_name.representation(options)

**Options**

- `p` Print the representation text.

**Examples**

```
pooll1.representations
```

displays the specifications of the estimation object POOL1.

**Cross-references**

See Chapters 15, 24, and 27 of the User's Guide, for basic details on the relevant estimation objects.

See also `spec` (p. 441).

---

**resample**

Resample from observations in a series or group.

**Syntax**

Object Proc: object_name.resample(options) [output_spec]

You should follow the `resample` keyword and options with a list of names or wildcard expression identifying the series to hold the output. If a list is used to identify the targets, the number of target series must match the number of names implied by the keyword.
### Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outsmpl = &quot;&quot;</td>
<td>Sample to fill the new series. Either provide the sample range in double quotes or specify a named sample object. The default is the current workfile sample.</td>
</tr>
<tr>
<td>name = group_name</td>
<td>Name of group to hold created series</td>
</tr>
<tr>
<td>permute</td>
<td>Draw from rows without replacement. Default is to draw with replacement.</td>
</tr>
<tr>
<td>weight = series_name</td>
<td>Name of series to be used as weights. The weight series must be non-missing and non-negative in the current workfile sample. The default is equal weights.</td>
</tr>
<tr>
<td>block = integer</td>
<td>Block length for each draw. Must be a positive integer. The default block length is 1.</td>
</tr>
<tr>
<td>withna (default)</td>
<td>Draw from all rows in the current sample, including those with NAs.</td>
</tr>
<tr>
<td>dropna</td>
<td>Do not draw from rows that contain missing values in the current workfile sample.</td>
</tr>
<tr>
<td>fixna</td>
<td>Excludes NAs from draws but copies rows containing missing values to the output series.</td>
</tr>
</tbody>
</table>

- Since we append a suffix for the new name, no series in the group can be an auto-series. For example, a group containing series such as X(-1) or LOG(X) will error. You will have to generate new series, say by setting XLAG = X(-1) or LOGX = LOG(X). Then create a new group consisting of XLAG and LOGX and call the bootstrap procedure on this new group.

- If the name you provided to group the resampled series already exists and if it is a group object, the group object will be overwritten with the resample series. If it already exists but is not a group object, EViews will error.

- Block bootstrap (block length larger than 1) requires a continuous output sample. Therefore a block length larger than 1 cannot be used together with the “fixna” option and the “outsmpl” should not contain any gaps.

- The “fixna” option will have an effect only if there are missing values in the overlapping sample of the input sample (current workfile sample) and the output sample specified by “outsmpl”. 
If you specify “fixna”, we first copy any missing values in the overlapping sample to the output series. Then the input sample is adjusted to drop the rows containing missing values and the output sample is adjusted not to overwrite the copied values.

If you choose “dropna” and the block length is larger than 1, the input sample may shrink in order to ensure that there are no missing values in any of the drawn blocks.

If you choose “permute”, the block option will be reset to 1, the “dropna” and “fixna” options will be ignored (reset to the default “withna” option), and the “weight” option will be ignored (reset to default equal weights).

Examples

\[
g1 \times g1 \\
g1.\text{resample}
\]

creates new series X_B and Y_B by drawing with replacement from the rows of X and Y in the current workfile sample. If X_B or Y_B already exist in the workfile, they will be overwritten if they are series objects; otherwise EViews will error. Note that only values of X_B and Y_B in the output sample (in this case the current workfile sample) will be overwritten.

\[
g1.\text{resample(weight=wt,suffix=\_2) g2}
\]

will append “\_2” to the names for the new series and group them in a group object named G2. The rows in the sample will be drawn with probabilities proportional to the corresponding values in the series WT. WT must have non-missing non-negative values in the current workfile sample.

Cross-references


See also \@resample (p. 548) and \@permute (p. 547) for sampling from matrices.

reset

<table>
<thead>
<tr>
<th>Command</th>
<th>Equation View</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset(n, options)</td>
<td>eq_name.reset(n, options)</td>
</tr>
</tbody>
</table>

Ramsey’s regression specification error test.

Syntax

Command: \texttt{reset(n, options)}

Equation View: eq\textunderscore name.reset(n, options)
You must provide the number of powers of fitted terms \( n \) to include in the test regression.

**Options**

\[ p \quad \text{Print the test result.} \]

**Examples**

\[
\text{equation eql.ls lwage c edu race gender}
\text{eql.reset(2)}
\]

carries out the RESET test by including two terms, the square and the cube of the fitted values.

**Cross-references**


**residcor**

Residual correlation matrix.

Displays the correlations of the residuals from each equation in the system, sspace, or var object, or from each pool cross-section. The sspace object residuals are the standardized one-step ahead signal forecast errors.

**Syntax**

Object View: \( \text{object_name.residcor(options)} \)

**Options**

\[ p \quad \text{Print the correlation matrix.} \]

**Examples**

\[
\text{sys1.residcor}
\]

displays the residual correlation matrix of SYS1.

**Cross-references**

See Chapters 23, 24, 27, and 25 of the User’s Guide for a discussion of systems, VARs, pools, and sspace models.

See also residcov (p. 385) and makeresids (p. 328).
Residual covariance matrix.
Displays the covariances of the residuals from each equation in the system, sspace, or var object, or each pool cross-section. The sspace object residuals are the standardized one-step ahead forecast errors.

**Syntax**

Object View: `object_name.residcov(options)`

**Options**

- `p` Print the covariance matrix.

**Examples**

`var1.residcov`

displays the residual covariance matrix of VAR1.

**Cross-references**


See also `residcor (p. 384)` and `makeresids (p. 328)`.

Display residuals.
For equation and pool objects, `resids` allows you to display the actual, fitted values and residuals in either tabular or graphical form.

For sspace objects, `resids` allows you to display the actual-fitted-residual graph.

For system, var or pool objects, `resids` displays multiple graphs of the residuals. Each graph will contain the residuals for each equation in the VAR, or for each cross-section in the pool.

**Syntax**

Object View: `object_name.resids(options)`
Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>(default) Display graph(s) of residuals.</td>
</tr>
<tr>
<td>t</td>
<td>Display table(s) of residuals (not available for system, pool, sspace or var objects).</td>
</tr>
<tr>
<td>p</td>
<td>Print the table/graph.</td>
</tr>
</tbody>
</table>

Examples

\[
\text{equation eq1.ls m1 c inc tb3 ar(1)}
\]
\[
eq1\text{.resids}
\]
regresses M1 on a constant, INC, and TB3 correcting for first order serial correlation, and displays a table of actual, fitted, and residual series.

\[
eq1\text{.resids(g)}
\]
displays a graph of the actual, fitted, and residual series.

Cross-references

See also \text{makeresids} (p. 328).

<table>
<thead>
<tr>
<th>results</th>
</tr>
</thead>
</table>

Displays the results view of objects containing estimation output.

Syntax

Object View: \text{object\_name.results(options)}

Options

<table>
<thead>
<tr>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
</tr>
</tbody>
</table>

Examples

\[
\text{equation eq1.ls m1 c inc tb3 ar(1)}
\]
\[
eq1\text{.results(p)}
\]
estimates an equation using least squares, and displays and prints the results.

\[
\text{var mvar.ls 1 4 8 8 m1 gdp tb3 @trend(70.4)}
\]
\[
mvar\text{.results(p)}
\]
prints the estimation results from the estimated VAR.
Cross-references

See the various chapters of the User’s Guide for a description of the results view and data members for each of the objects.

### rls

**Equation View**

Recursive least squares regression.

The \texttt{rls} view of an equation displays the results of recursive least squares (rolling) regression. This view is only available for (non-panel) equations estimated by ordinary least squares without ARMA terms.

You can plot various statistics from \texttt{rls} by choosing an option.

**Syntax**

```
Equation View: eq_name.rls(options) c(1) c(2) …
```

**Options**

- \texttt{r} Plot the recursive residuals about the zero line with plus and minus two standard errors.
- \texttt{r,s} Plot the recursive residuals and save the residual series and their standard errors as series named R_RES and R_RESSE, respectively.
- \texttt{c} Plot the recursive coefficient estimates with two standard error bands.
- \texttt{c,s} Plot the listed recursive coefficients and save all coefficients and their standard errors as series named R_C1, R_C1SE, R_C2, R_C2SE, and so on.
- \texttt{o} Plot the \textit{p}-values of recursive one-step Chow forecast tests.
- \texttt{n} Plot the \textit{p}-values of recursive \textit{n}-step Chow forecast tests.
- \texttt{q} Plot the CUSUM (standardized cumulative recursive residual) and 5 percent critical lines.
- \texttt{v} Plot the CUSUMSQ (CUSUM of squares) statistic and 5 percent critical lines.
- \texttt{p} Print the view.
Examples

\begin{verbatim}
equation eq1.ls ml c tb3 gdp
eq1.rls(r,s)
eq1.rls(c) c(2) c(3)
\end{verbatim}

plots and saves the recursive residual series and their standard errors from EQ1 as R_RES and R_RESSE. The third line plots the recursive slope coefficients of EQ1.

\begin{verbatim}
equation eq2.ls ml c pdl(tb3,12,3) pdl(gdp,12,3)
eq2.rls(c) c(3)
eq2.rls(q)
\end{verbatim}

The second command plots the recursive coefficient estimates of PDL02, the linear term in the polynomial of TB3 coefficients. The third line plots the CUSUM test statistic and the 5% critical lines.

Cross-references

See “Recursive Least Squares” on page 557 of the User’s Guide.

\midrule
\textbf{rndint} & \textbf{Command} \\
\midrule
\begin{verbatim}
Generate uniform random integers.

The \texttt{rndint} command fills series, vector, and matrix objects with (pseudo) random integers drawn uniformly from zero to a user specified maximum. The \texttt{rndint} command ignores the current sample and fills the entire object with random integers.

Syntax

\texttt{Command: \texttt{rndint(object_name, n)}}

Type the name of the series, vector, or matrix object to fill followed by an integer value representing the maximum value \texttt{n} of the random integers. \texttt{n} should a positive integer.

Examples

\begin{verbatim}
series index
rndint(index,10)
\end{verbatim}

fills the entire series INDEX randomly with integers from 0 to 10. Note that, unlike \texttt{genr}, \texttt{rndint} ignores the current sample and fills the series for the entire workfile range.

\begin{verbatim}
sym(3) var3
rndint(var3,5)
\end{verbatim}
fills the entire symmetric matrix VAR3 with random integers ranging from 0 to 5.

Cross-references
See the list of available random number generators in Appendix F, “Operator and Function Reference”, beginning on page 573.

See also `nrnd (p. 523)`, `rnd (p. 525)` and `rndseed (p. 389)`.

<table>
<thead>
<tr>
<th><code>rndseed</code></th>
<th>Command</th>
</tr>
</thead>
</table>

Seed the random number generator.

Use `rndseed` when you wish to generate a repeatable sequence of random numbers or to select the generator to be used.

EViews 4.0 now supports three types of uniform random number generators. Note that many of the non-uniform random numbers are generated as transformations of the uniform random number.

Syntax
Command: `rndseed(options) integer`

Options
- `type = arg` Type of random number generator: Knuth’s (1997) lagged Fibonacci generator (“type = kn”), L’Ecuyer’s (1999) combined multiple recursive generator (“type = le”), or Matsumoto and Nishimura’s (1998) Mersenne Twister (“type = mt”).

Follow the `rndseed` keyword with the optional generator type, and then an integer for the seed.

- **Important backward compatibility note:** while the default generator type in EViews 4.0 remains the same as in EViews 3.0, setting the seed to the same number will no longer generate the same sequence. The reason for this discrepancy is due to the modification we introduced in the seeding of Knuth’s generator. After reseeding the generator, we now discard a few draws as burn-in. The reason for this “wasteful” step is to avoid the repeats we observed in a short run of draws immediately following a reseed of the Knuth generator.
When EViews starts up, the default generator type is set to the Knuth lagged Fibonacci generator. Unless changed using `rndseed`, Knuth’s generator will be used for subsequent pseudo-random number generation.

### Examples
```
rndseed 123456
genr t3=@qtdist(rnd,3)
rndseed 123456
genr t30=@qtdist(rnd,30)
```
generates random draws from a $t$-distribution with 3 and 30 degrees of freedom using the same seed.

### Cross-references
See the list of available random number generators in Appendix F, “Operator and Function Reference”, beginning on page 573.

See also `nrnd` (p. 523), `rnd` (p. 525) and `rndint` (p. 388).

---

**rowvector**

Object Declaration

Declare a rowvector object.

The `rowvector` command declares and optionally initializes a (row) vector object.

### Syntax
```
Command:          rowvector(n1) vector_name
Command:          rowvector vector_name = assignment
```

You may optionally specify the size (number of columns) of the row vector in parentheses after the `rowvector` keyword. If you do not specify the size, EViews creates a rowvector of size 1 unless the declaration is combined with an assignment.
By default, all elements of the vector are set to 0, unless an assignment statement is provided. EViews will automatically resize new rowvectors, if appropriate.

**Examples**

\[
\text{rowvector } rvec1 \\
\text{rowvector(20) coefvec=2} \\
\text{rowvector newcoef=coefvec}
\]

RVEC1 is a row vector of size one with element 0. COEFVEC is a row vector of size 20 with all elements equal to 2. NEWCOEF is also a row vector of size 20 with all elements equal to 2.

**Cross-references**

See “Rowvector” on page 156 for a complete description of the rowvector object.

See also coef (p. 223) and vector (p. 489).

### run

**Command**

Run a program.

The `run` command executes a program. The program may be located in memory or stored in a program file on disk.

**Syntax**

Command: `run(options) [path]\prog_name [%0 %1 ...]`

If the program has arguments, you should list them after the filename. EViews first checks to see if the specified program is in memory. If not, it looks for the program on disk in the current working directory, or in the specified path. EViews expects that program files will have a “.PRG” extension.

**Options**

- `integer (default = 1)` Set maximum errors allowed before halting the program.
- `c` Run program file without opening a window for display of the program file.
Examples

\[ \text{run}(q) \text{ simul } x \ xhat \]

quietly runs a program named SIMUL from the default directory using arguments X and XHAT.

Since `run` is a command, it may also be placed in a program file. You should note that if you put the `run` command in a program file and then execute the program, EViews will stop after executing the program referred to by the `run` command. For example, if you have a program containing

\[ \begin{align*}
\text{run simul} \\
\text{print } x
\end{align*} \]

the `print` statement will not be executed since execution will stop after executing the commands in SIMUL.PRG. If this behavior is not intended, you should consider using the `include` statement (p. 563).

Cross-references

See “Executing a Program” on page 84 for further details.

See also `include` (p. 563).

<table>
<thead>
<tr>
<th>sample</th>
<th>Object Declaration</th>
</tr>
</thead>
</table>

Declare a sample object.

The `sample` statement declares, and optionally defines, a sample object. If no sample statement is provided, the sample object will be set to the current workfile sample.

To reset the sample dates in a sample object, you must use the `set` procedure.

Syntax

Command:  \[ \text{sample } \text{smpl\_name} \text{ smpl\_range} [\text{if\_statement}] \]
Follow the `sample` keyword with a name for the sample object and a sample range, possibly with an if condition.

**Examples**

```eviews
sample ss
```
denotes a sample object named SS and sets it to the current workfile sample.

```eviews
sample s2 1974:1 1995:4
```
declares a sample object named S2 and sets it from 1974:1 to 1995:4.

```eviews
sample fe_bl @all if gender=1 and race=3
smpl fe_bl
```
The first line declares a sample FE_BL with observations such that GENDER = 1 and RACE = 3 in the current workfile range. The second line sets the current sample to FE_BL.

```eviews
sample sf @last-10 @last
```
declares a sample object named SF and sets it to the last 10 observations of the current workfile range.

```eviews
sample sl @first 1973:1
sl.set 1973:2 @last
```
The first line declares a sample object named S1 and sets it from the beginning of the workfile range to 1973:1. The second line resets S1 from 1973:2 to the end of the workfile range.

**Cross-references**

See “Samples” on page 81, and “Dates” on page 875 of the User’s Guide for a discussion of using dates and samples in EViews.

See also `set` (p. 407) and `smpl` (p. 436).

### save

<table>
<thead>
<tr>
<th><strong>Command</strong></th>
<th><strong>Graph Proc</strong></th>
<th><strong>Table Proc</strong></th>
</tr>
</thead>
</table>

Save graph to disk as a Windows metafile or PostScript file. Save table to disk as a CSV, tab-delimited ASCII text, RTF, or HTML file.

`save` may also be used as a command to save the current workfile to disk.

The latter usage is provided only for backward compatibility as it has been replaced with the equivalent `wfsave` (p. 493) command.
Syntax

Graph Proc: graph_name.save(options) [path\]file_name

Table Proc: graph_name.save(options) [path\]file_name

Follow the keyword with a name for the file. file_name may include the file type extension, or the file type may be specified using the “t=” option. The graph may be saved with an .EMF, .WMF, or .EPS extension.

If path is not specified, the file will be stored in the default directory, as set in the File Locations global options.

General Graph Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t=file_type</td>
<td>Specifies the file type, where file_type may be one of: “emf”, “meta”, or “metafile”, (Enhanced Windows metafile), “wmf” (ordinary Windows metafile), or “eps”, “ps” or “postscript” (Encapsulated PostScript). Files will be saved with the “.emf”, “.wmf”, and “.eps” extensions, respectively.</td>
</tr>
<tr>
<td>u=units</td>
<td>Specify units of measurement, where units is one of: “in” (inches), “cm” (centimeters), “pt” (points), “pica” (picas).</td>
</tr>
<tr>
<td>w=width</td>
<td>Set width of the graphic in the selected units.</td>
</tr>
<tr>
<td>h=height</td>
<td>Set height of the graphic in the selected units.</td>
</tr>
<tr>
<td>c / -c</td>
<td>[Save / Do not save] the graph in color.</td>
</tr>
</tbody>
</table>

Note that if only a width or a height option is specified, EViews will calculate the other dimension holding the aspect ratio of the graph constant. If both width and height are provided, the aspect ratio will no longer be locked. EViews will default to the current graph dimensions if size is unspecified.

All defaults are taken from the global graph export settings (Options/Graphics Defaults.../Exporting).
Postscript specific Graph Options

box / -box  
[Save / Do not save] the graph with a bounding box. The bounding box is an invisible rectangle placed around the graphic to indicate its boundaries. The default is taken from the global graph export settings.

land  
Save the graph in landscape orientation. The default uses portrait mode.

Table Options

\( t = \text{file\_type} \) \( (default = \text{"csv"}) \)  
Specifies the file type, where \text{file\_type} may be one of: “csv” (CSV - comma-separated), “rtf” (Rich-text format), “txt” (tab-delimited text), or “html” (HTML).
Files will be saved with the “.csv”, “.rtf”, “.txt”, and “.htm” extensions, respectively.

\( s = \text{arg} \)  
Scale size, where \text{arg} is from 0.05 to 2, representing the fraction of the original table size (only valid for HTML or RTF files).

\( r = \text{cell\_range} \)  
Range of table cells to be saved. See \text{setfillcolor} (p. 416) for the \text{cell\_range} syntax. If a range is not provided, the entire table will be saved.

\( n = \text{string} \)  
Replace all cells that contain NA values with the specified string. “NA” is the default.

\( f / -f \)  
[use full precision values/ do not use full precision] when saving values to the table (only applicable to numeric cells). By default, the values will be saved as they appear in the currently formatted table.

Examples

\( \text{graph1.save}(t=ps, -box, land) \text{ c:\data\MyGra1} \)
saves GRAPH1 as a PostScript file MYGRA1.EPS, without a bounding box and with landscape orientation.

\( \text{graph2.save}(u=pts, w=300, h=300) \text{ MyGra2} \)
saves GRAPH2 in the default directory as an Enhanced Windows metafile MYGRA2.EMF. The image will be scaled to \( 300 \times 300 \) points.
The command

```plaintext
tab1.save mytable
```

saves TAB1 to a CSV file named MYTABLE.CSV.

```plaintext
tab1.save(t=csv, n="NAN") mytable
```

saves TAB1 to a CSV (comma separated value) file named MYTABLE.CSV and writes all NA values as “NAN”.

```plaintext
tab1.save(r=B2:C10, t=html, s=.5) mytable
```

saves from data from the second row, second column to the tenth row, third column to a HTML file named MYTABLE.HTM at half of the original size.

```plaintext
tab1.save(f, n=".", r=B) mytable
```

saves the data in the second column in full precision to a CSV file names MYTABLE.CSV and writes all NA values as “.”.

**Cross-references**


To save a workfile, see the equivalent `wfsave` (p. 493) command.

---

<table>
<thead>
<tr>
<th><strong>scalar</strong></th>
<th><strong>Object Declaration</strong></th>
</tr>
</thead>
</table>

Declare a scalar object.

The `scalar` command declares a scalar object and optionally assigns a value.

**Syntax**

Command: `scalar scalar_name`

Command: `scalar scalar_name = assignment`

The `scalar` keyword should be followed by a valid name, and optionally, by an assignment. If there is no explicit assignment, the scalar will be assigned a value of zero.

**Examples**

```plaintext
scalar alpha
```

declares a scalar object named ALPHA with value zero.

```plaintext
equation eq1.ls res c res(-1 to -4) x1 x2
```

---
scalar lm=eq1.@regobs*eq1.@r2
show lm

runs a regression, saves the \( nR^2 \) as a scalar named LM, and displays its value in the status line at the bottom of the EViews window.

Cross-references

See “Scalar” on page 158 for a summary of the features of scalar objects in EViews.

<table>
<thead>
<tr>
<th>scale</th>
<th>Graph Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sets axis and data scaling characteristics for the graph.

By default, EViews optimally chooses the axes to fit the graph data.

Syntax

Graph Proc: graph_name.scale(axis_id) options_list

The axis_id parameter identifies which of the axes the proc modifies. If no option is specified, the proc will modify all of the axes. axis_id may take on one of the following values:

- left / l Left vertical axis.
- right / r Right vertical axis.
- bottom / b Bottom axis for XY and scatter graphs (scat (p. 399), xy (p. 510), xyline (p. 512), xypair (p. 515)).
- top / t Top axis for XY and scatter graphs (scat (p. 399), xy (p. 510), xyline (p. 512), xypair (p. 515)).
- all / a All axes.

Note: the syntax of scale has changed considerably from Version 4.1 of EViews (see, in particular axis (p. 197)). While not documented here, the previous options are still (for the most part) supported. However, we do not recommend using the old options as future support is not guaranteed.

Options

The options list may include any of the following options:
**Data scaling options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>Linear data scaling <em>(default)</em>.</td>
</tr>
<tr>
<td>linearzero</td>
<td>Linear data scaling (include zero when auto range selection is employed).</td>
</tr>
<tr>
<td>log</td>
<td>Logarithmic scaling <em>(does not apply to boxplots)</em>.</td>
</tr>
<tr>
<td>norm</td>
<td>Norm (standardize) the data prior to plotting <em>(does not apply to boxplots)</em>.</td>
</tr>
</tbody>
</table>

**Axes scaling options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>range(arg)</td>
<td>Specifies the endpoints for the scale: <em>arg</em> = “auto” <em>(automatic choice)</em>, <em>arg</em> = “minmax” <em>(use the maximum and minimum values of the data)</em>, <em>arg</em> = “n1,n2” <em>(set minimum to n1 and maximum to n2, e.g. “range(3,9)”)</em>.</td>
</tr>
<tr>
<td>overlap / -overlap</td>
<td>[Overlap / Do not overlap] scales on dual scale graphs.</td>
</tr>
<tr>
<td>invert / -invert</td>
<td>[Invert / do not invert] scale.</td>
</tr>
</tbody>
</table>

Note that the default settings are taken from the global defaults.

**Examples**

To set the right scale to logarithmic with manual range you can enter

```plaintext
graph1.scale(right) log range(10, 30)
```

Alternatively,

```plaintext
graph1.scale -invert range(minmax)
```

draws duplicate inverted scales on the left and right axes which are defined to match the data range.

**Cross-references**


See also *axis* (p. 197), *datelabel* (p. 237), *options* (p. 343) and *setelem* (p. 412).
Display scatterplot graph of object, or change existing graph object type to scatterplot (if possible).

By default, the first series or column of data will be located along the horizontal axis and the remaining data on the vertical axis. You may optionally choose to plot the data in pairs, where the first two series or columns are plotted against each other, the second two series or columns are plotted against each other, and so forth.

Syntax

Command: \texttt{scat(options) arg1 \{arg2 arg3 \ldots\}}

Object View: \texttt{group\_name.scat(options)}

Graph Proc: \texttt{graph\_name.scat(options)}

If used as a command, follow the keyword by a list of series and group objects, or by a matrix object. There must be at least two series or columns in the data to be graphed.

Scatterplots are simply XY plots (see \texttt{xy} (p. 510)) with symbols turned on, and lines turned off (see \texttt{setelem} (p. 412)).

Options

Template and printing options

\begin{itemize}
  \item \texttt{o = graph\_name} Use appearance options from the specified graph object.
  \item \texttt{t = graph\_name} Use appearance options and copy text and shading from the specified graph.
  \item \texttt{p} Print the scatter plot.
\end{itemize}

\textit{Note that the use of a template option will override the symbol setting.}

Scale options

\begin{itemize}
  \item \texttt{a (default)} Automatic single scale.
  \item \texttt{b} Plot series or columns in pairs (the first two against each other, the second two against each other, and so forth).
  \item \texttt{n} Normalized scale (zero mean and unit standard deviation). May not be used with the “s” option.
\end{itemize}
Panel options

The following options apply when graphing panel structured data.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>Dual scaling with no crossing.</td>
</tr>
<tr>
<td>x</td>
<td>Dual scaling with possible crossing.</td>
</tr>
<tr>
<td>m</td>
<td>Display XY plots in multiple graphs (will override the “s” option). Not for use with an existing graph object.</td>
</tr>
</tbody>
</table>

Examples

```
scat unemp inf want
```
produces an UNTITLED graph object containing a scatter plot with UNEMP on the horizontal and INF and WANT on the vertical axis.

```
group med age height weight
med.scat(t=scat2)
```
produces a scatter plot view of the group object MED using the graph object SCAT2 as a template.

```
group pairs age height weight length
pairs.scat(b)
```
produces a scatter plot view with AGE plotted against HEIGHT, and WEIGHT plotted against LENGTH.
If there is an existing graph GRAPH01, the expression

    graph01.scat(b)

changes its type to a scatterplot with data plotted in pairs, and the remaining XY graph settings at their default values.

Cross-references


See *xy* (p. 510), and *graph* (p. 288) for graph declaration and modification, and additional graph types. See also *xyline* (p. 512) for XY line graphs.

### scatmat

**Matrix of scatter plots.**

The *scatmat* view displays a matrix of scatter plots for all pairs of series in a group.

**Syntax**

```
Group View: group_name.scatmat(options)
```

**Options**

- **p** Print the scatter plot matrix.

**Examples**

```
    group g1 weight height age
    g1.scatmat
```

displays a $3 \times 3$ matrix of scatter plots for all pairs of the three series in group G1.

**Cross-references**


### scenario

**Manage the model scenarios.**

The scenario procedure is used to set the active and comparison scenarios for a model, to create new scenarios, to initialize one scenario with settings from another scenario, to delete scenarios, and to change the variable aliasing associated with a scenario.
Syntax

Model Proc: \texttt{model\_name.scenario(options) "name"}

performs scenario options on a scenario given by the specified name (entered in double quotes). By default the scenario procedure also sets the active scenario to the specified name.

Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Set the comparison scenario to the named scenario.</td>
</tr>
<tr>
<td>n</td>
<td>Create a new scenario with the specified name.</td>
</tr>
<tr>
<td>i = &quot;name&quot;</td>
<td>Copy the Excludes and Overrides from the named scenario.</td>
</tr>
<tr>
<td>d</td>
<td>Delete the named scenario.</td>
</tr>
<tr>
<td>a = string</td>
<td>Set the scenario alias string to be used when creating aliased variables (\texttt{string} is a 1 to 3 alphanumeric string to be used in creating aliased variables). If an underscore is not specified, one will be added to the beginning of the string. Examples: &quot;.5&quot;, &quot;.T&quot;, &quot;S2&quot;. The string &quot;A&quot; may not be used since it may conflict with add factor specifications.</td>
</tr>
</tbody>
</table>

Examples

The command string

\texttt{mod1.scenario "baseline"}

sets the active scenario to the baseline, while

\texttt{mod1.scenario(c) "actuals"}

sets the comparison scenario to the actuals (warning: this will overwrite any historical data in the solution period).

A newly created scenario will become the active scenario. Thus,

\texttt{mod1(n) "Peace Scenario"}

creates a scenario called "Peace Scenario" and makes it the active scenario. The scenario will automatically be assigned a unique numeric alias. To change the alias, simply use the "a = " option:
mod1(a=_ps) "Peace Scenario"

changes the alias for “Peace Scenario” to “_PS” and makes this scenario the active scenario.

The command:

    mod1.scenario(n, a=w, i="Peace Scenario", c) "War Scenario"

creates a scenario called “War Scenario”, initializes it with the Excludes and Overrides contained in “Peace Scenario”, associates it with the alias “_W”, and makes this scenario the comparison scenario.

    mod1.scenario(i="Scenario 1") "Scenario 2"

copies the Excludes and Overrides in “Scenario 1” to “Scenario 2” and makes “Scenario 2” the active scenario.

Compatibility Notes

For backward compatibility with EViews 4.0, the option “a” may be used to set the comparison scenario, but is method not guaranteed to be supported in the future.

In all of the arguments above the quotation marks around scenario name are currently optional. Support for the non-quoted names is provided for backward compatibility, but may be dropped in the future, thus

    mod1.scenario Scenario 1

is currently valid, but may not be in future versions of EViews.

Cross-references

Scenarios are described in detail beginning on page 765 of the User’s Guide. Chapter 26 of the User’s Guide documents EViews models in depth.

See also solve (p. 438) in the Command and Programming Reference.

Seasonal adjustment.

The seas command carries out seasonal adjustment using either the ratio to moving average, or the difference from moving average technique.

EViews also performs Census X11 and X12 seasonal adjustment. For details, see x11 (p. 502) and x12 (p. 504).
Syntax

Command: `seas(options) series_name name_adjust [name_fac]`
Series Proc: `series_name.seas(options) name_adjust [name_fac]`

To use `seas` as a command, list the name of the original series and the name to be given to the seasonally adjusted series. You may optionally include an additional name for the seasonal factors. `seas` will display the seasonal factors using the convention of the Census X11 program.

`seas` used as a series procedure applies seasonal adjustment to a series.

Options

- `m`: Multiplicative (ratio to moving average) method.
- `a`: Additive (difference from moving average) method.

Examples

- `seas(a) pass pass_adj pass_fac`
  seasonally adjusts the series PASS using the additive method and saves the adjusted series as PASS_ADJ and the seasonal factors as PASS_FAC.

- `sales.seas(m) adj_sales`
  seasonally adjusts the series SALES using the multiplicative method and saves the adjusted series as ADJ_SALES.

Cross-references

See “Seasonal Adjustment” on page 312 of the User’s Guide for a discussion of seasonal adjustment methods.

See also `seasplot` (p. 404), `x11` (p. 502) and `x12` (p. 504).

<table>
<thead>
<tr>
<th>seasplot</th>
<th>Series View</th>
</tr>
</thead>
</table>

Seasonal line graph.

`seasplot` displays a line graph view of a series ordered by season. Available only for quarterly and monthly frequencies.

Syntax

Series View: `series_name.seasplot(options)`
Options

m  Plot series split by season. Default is to plot series stacked by season.

Examples

freeze(gra_ip) ipnsa.seasplot

creates a graph object named GAR_IP that contains the stacked seasonal line graph view of the series IPNSA.

Cross-references

See “Spreadsheet and Graph Views” on page 285 of the User’s Guide for a brief discussion of seasonal line graphs.

See also seas (p. 403), x11 (p. 502) and x12 (p. 504).

<table>
<thead>
<tr>
<th>series</th>
<th>Object Declaration</th>
</tr>
</thead>
</table>

Declare a series object.

The series command creates and optionally initializes a series, or modifies an existing series.

Syntax

Command:  

series ser_name

Command:  

series ser_name = formula

The series command should be followed by either the name of a new series, or an explicit or implicit expression for generating a series. If you create a series and do not initialize it, the series will be filled with NAs. Rules for composing a formula are given in “Numeric Expressions” on page 111 of the User’s Guide.

Examples

series x

creates a series named X filled with NAs.

Once a series is declared, you do not need to include the series keyword prior to entering the formula. The following example generates a series named LOW that takes value 1 if either INC is less than or equal to 5000 or EDU is less than 13 and 0 otherwise.

series low
This example solves for the implicit relation and generates a series named Z which is the double log of Y so that \( Z = \log(\log(Y)) \).

\[
\text{series } \exp(\exp(z)) = y
\]

The command

\[
\text{series } z = \frac{x+y}{2}
\]

creates a series named Z which is the average of series X and Y.

\[
\text{series } \text{cwage} = \text{wage} \cdot (\text{hrs} > 5)
\]

generates a series named CWAGE which is equal to WAGE if HRS exceeds 5 and zero otherwise.

\[
\text{series } 10^z = y
\]

generates a series named Z which is the base 10 log of Y.

The commands

\[
\text{series } y_t = y
\]

\[
\text{smpl if } y < 0
\]

\[
y_t = \text{na}
\]

\[
\text{smpl } \text{@all}
\]

generates a series named Y_T which replaces negative values of Y with NAs.

\[
\text{series } z = \text{movav}(x(+2), 5)
\]

creates a series named Z which is the centered moving average of the series X with two leads and two lags. This works only for centered moving averages over an odd number of periods.

\[
\text{series } z = 0.5 \cdot x(6) + \text{movsum}(x(5), 11) + 0.5 \cdot x(-6))/12
\]

generates a series named Z which is the centered moving average of the series X over twelve periods.

\[
\text{genr } y = 2 + (5-2) \cdot \text{rnd}
\]

creates series named Y which is a random draw from a uniform distribution between 2 and 5.

\[
\text{series } y = 3 + \sqrt{5} \cdot \text{nrnd}
\]

generates a series named Y which is a random draw from a normal distribution with mean 3 and variance 5.
Cross-references

A full listing and description of series functions and expressions is provided in Appendix F, "Operator and Function Reference", on page 573.


<table>
<thead>
<tr>
<th>set</th>
<th>Sample Proc</th>
</tr>
</thead>
</table>

Set the sample in a sample object.

The `set` procedure resets the sample of an existing sample object.

Syntax

```
Sample Proc: sample_name.set sample_description
```

Follow the `set` command with a sample description. See `sample` for instructions on describing a sample.

Examples

```
sample s1 @first 1973
s1.set 1974 @last
```

The first line declares and defines a sample object named S1 from the beginning of the workfile range to 1973. The second line resets S1 from 1974 to the end of the workfile range.

Cross-references

See “Samples” on page 81 of the User’s Guide for a discussion of samples in EViews. See also `sample` (p. 392) and `smpl` (p. 436).

<table>
<thead>
<tr>
<th>setbpelem</th>
<th>Graph Proc</th>
</tr>
</thead>
</table>

Enable/disable individual boxplot elements.

Syntax

```
Graph Proc: graph_name.setbpelem element_list
```

The `element_list` may contain one or more of the following:
Examples

```bash
graph01.setbpelem -far width(n) ci(notch)
```

hides the far outliers, sets the box widths proportional to the number of observations, and enables notching of the confidence intervals.

Cross-references


See `setelem` (p. 412) to modify line and symbol attributes. See also `bplabel` (p. 210), `options` (p. 343), `axis` (p. 197), and `scale` (p. 397).

<table>
<thead>
<tr>
<th><code>setcell</code></th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert contents into cell of a table.</td>
<td></td>
</tr>
</tbody>
</table>

The `setcell` command puts a string or number into a cell of a table.
Syntax

Command: \texttt{setcell(table\_name, r, c, content[, "options"])}

Options

Provide the following information in parentheses in the following order: the name of the table object, the row number, \( r \), of the cell, the column number, \( c \), of the cell, a number or string to put in the cell, and optionally, a justification and/or numerical format code. A string of text must be enclosed in double quotes.

The justification options are:

- \( c \quad \text{Center the text/number in the cell.} \)
- \( r \quad \text{Right-justify the text/number in cell.} \)
- \( l \quad \text{Left-justify the text/number in cell.} \)

The numerical format code determines the format with which a number in a cell is displayed; cells containing strings will be unaffected. The format code can either be a positive integer, in which case it specifies the number of decimal places to be displayed after the decimal point, or a negative integer, in which case it specifies the total number of characters to be used to display the number. These two cases correspond to the Fixed decimal and Fixed character fields in the number format dialog.

Note that when using a negative format code, one character is always reserved at the start of a number to indicate its sign, and that if the number contains a decimal point, that will also be counted as a character. The remaining characters will be used to display digits. If the number is too large or too small to display in the available space, EViews will attempt to use scientific notation. If there is insufficient space for scientific notation (six characters or less), the cell will contain asterisks to indicate an error.

Examples

\texttt{setcell(tab1, 2, 1, "Subtotal"})

puts the string “Subtotal” in row 2, column 1 of the table object named TAB1.

\texttt{setcell(tab1, 1, 1, "Price and cost", "r")}

puts the a right-justify string “Price and cost” in row 1, column 1 of the table object named TAB1.

Cross-references

Chapter 4 describes table formatting using commands. See Chapter 14 of the User’s Guide for a discussion and examples of table formatting in EViews.
See also `setwidth` (p. 431).

<table>
<thead>
<tr>
<th>Command</th>
<th>setcolwidth</th>
</tr>
</thead>
</table>

Set width of a column of a table.

Provided for backward compatibility. See `setwidth` (p. 431) for the new method of setting the width of table and spreadsheet columns.

**Syntax**

Command: `setcolwidth(table_name, c, width)`

**Options**

To change the width of a column, provide the following information in parentheses in the following order: the name of the table, the column number `c`, and the number of characters `width` for the new width. EViews measures units in terms of the width of a numeric character. Because different characters have different widths, the actual number of characters that will fit may differ slightly from the number you specify. By default, each column is approximately 10 characters wide.

**Examples**

```
setcolwidth(mytab,2,20)
```

sets the second column of table `MYTAB` to fit approximately 20 characters.

**Cross-references**

Chapter 4, “Working with Tables”, on page 47 describes table formatting using commands. See also Chapter 14, “Graphs, Tables, and Text Objects”, on page 387 of the User’s Guide for a discussion and examples of table formatting in EViews.

See also `setwidth` (p. 431) and `setheight` (p. 423).

<table>
<thead>
<tr>
<th>Series Proc</th>
<th>setconvert</th>
</tr>
</thead>
</table>

Set frequency conversion method.

Determines the default frequency conversion method for a series when copied or linked to different frequency workfiles. You can override this default conversion method by specifying a frequency conversion method as an option in the specific command (using `copy` (p. 228), `fetch` (p. 264), or `linkto` (p. 310)).
If you do not set a conversion method and if you do not specify a conversion method as an option in the command, EViews will use the conversion method set in the global option.

Syntax

Series Proc: `ser_name.setconvert [up_method down_method]`

Follow the series name with a period, the keyword, and option letters to specify the frequency conversion method. If either the up-conversion or down-conversion method is omitted, EViews will set the corresponding method to **Use EViews default**.

Options

The following options control the frequency conversion method when copying series and group objects to a workfile, converting from **low** to **high** frequency:

<table>
<thead>
<tr>
<th>Low to high conversion methods</th>
<th>“r” (constant match average), “d” (constant match sum), “q” (quadratic match average), “t” (quadratic match sum), “i” (linear match last), “c” (cubic match last).</th>
</tr>
</thead>
</table>

The following options control the frequency conversion method when copying series and group objects to a workfile, converting from **high** to **low** frequency:

<table>
<thead>
<tr>
<th>High to low conversion methods</th>
<th><strong>High to low conversion methods removing NAs:</strong> “a” (average of the nonmissing observations), “s” (sum of the nonmissing observations), “f” (first nonmissing observation), “l” (last nonmissing observation), “x” (maximum nonmissing observation), “m” (minimum nonmissing observation).&lt;br&gt;<strong>High to low conversion methods propagating NAs:</strong> “an” or “na” (average, propagating missings), “sn” or “ns” (sum, propagating missings), “fn” or “nf” (first, propagating missings), “ln” or “nl” (last, propagating missings), “xn” or “nx” (maximum, propagating missings), “mn” or “nm” (minimum, propagating missings).</th>
</tr>
</thead>
</table>

Examples

```
unemp.setconvert a
```

sets the default down-conversion method of the series UNEMP to take the average of nonmissing observations, and resets the up-conversion method to use the global default.
ibm_hi.setconvert xn d

sets the default down-conversion method for IBM_HI to take the largest observation of the
higher frequency observations, propagating missing values, and the default up-conversion
method to constant, match sum.

consump.setconvert

resets both methods to the global default.

Cross-references

See “Frequency Conversion” on page 96 of the User’s Guide for a discussion of frequency
conversions and the treatment of missing values.

See also copy (p. 228), fetch (p. 264) and linkto (p. 310).

<table>
<thead>
<tr>
<th>setelem</th>
<th>Graph Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Set individual line, bar and legend options for each series in the graph.

Syntax

Graph Proc: graph_name.setelem(graph_elem) argument_list

where graph_elem is the identifier for the graph element whose options you wish to modify:

- **integer**
  Index for graph element (for non boxplot graphs). For example, if you specify provide the integer “2”, EViews will modify the second line in the graph.

- **box_elem**
  Boxplot element to be modified: box (“b”), median ("med"), mean ("mean"), near outliers ("near" or “no”), far outliers (“far” or “fo”), whiskers (“w”), staples (“s”). For boxplot graphs only.

The argument list for setelem may contain one or more of the following:

- **linecolor(args)**, **lcolor(args)**
  Sets the line and symbol color. The args value may set by using one of the color keywords (e.g., “blue”), or by using the RGB values, e.g., “@RGB(255, 255, 0)”). For a description of the available color keywords, see set-
  fillcolor (p. 416).
**setelem**

<table>
<thead>
<tr>
<th>linepattern(arg), lpat(arg)</th>
<th>Sets the line pattern to the type: <code>arg</code> can be an integer from 1–11 or one of the matching keywords.</th>
<th>(1) solid</th>
<th>(2) dash1</th>
<th>(3) dash2</th>
<th>(4) dash3</th>
<th>(5) dash4</th>
<th>(6) dash5</th>
<th>(7) dash6</th>
<th>(8) dash7</th>
<th>(9) dash8</th>
<th>(10) dash9</th>
<th>(11) none</th>
</tr>
</thead>
<tbody>
<tr>
<td>Note that the option interacts with the graph options for “color”, “lineauto”, “linesolid”, “linepat” (see options (p. 343), for details). You may need to set the graph option for “linepat” to enable the display of line patterns. See options (p. 343).</td>
<td>The “none” option turns off lines and uses only symbols.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>linewidth(n1), lwidth(n1)</td>
<td>Sets the line and symbol width: <code>n1</code> should be a number between “.25” and “5”, indicating the width in points.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>symbol(arg)</td>
<td>Sets the drawing symbol: <code>arg</code> can be an integer from 1–13, or one of the matching keywords. Selecting a symbol automatically turns on symbol use. The “none” option turns off symbol use.</td>
<td>(1) circle</td>
<td>(2) filledcircle</td>
<td>(3) transcircle</td>
<td>(4) star</td>
<td>(5) diagcross</td>
<td>(6) cross</td>
<td>(7) filledsquare</td>
<td>(8) square</td>
<td>(9) filledtriup</td>
<td>(10) triup</td>
<td>(11) filledtridown</td>
</tr>
<tr>
<td>fillcolor(arg), fcolor(arg)</td>
<td>Sets the fill color for symbols, bars, and pies. The <code>args</code> value may be set by using of the color keywords (e.g., “blue”), or by using the RGB values, e.g., “@RGB(255, 255, 0)”). For a description of the available color keywords, see <code>setfillcolor</code> (p. 416).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
fillgray(n1), gray(n1) Sets the gray scale for bars and pies: \( n1 \) should be an integer from 1–15 corresponding to one of the predefined gray scale settings (from lightest to darkest).

<table>
<thead>
<tr>
<th>( n1 )</th>
<th>Gray Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lightest</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Darkest</td>
</tr>
</tbody>
</table>

fillhatch(arg), hatch(arg) Sets the hatch characteristics for bars and pies: \( arg \) can be an integer from 1–7, or one of the matching keywords.

<table>
<thead>
<tr>
<th>( arg )</th>
<th>Hatch Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>none</td>
</tr>
<tr>
<td>2</td>
<td>diagcross</td>
</tr>
<tr>
<td>3</td>
<td>horizontal</td>
</tr>
<tr>
<td>4</td>
<td>fdiagonal</td>
</tr>
<tr>
<td>5</td>
<td>vertical</td>
</tr>
<tr>
<td>6</td>
<td>cross</td>
</tr>
<tr>
<td>7</td>
<td>bdiagonal</td>
</tr>
</tbody>
</table>

preset(n1) Sets line and fill characteristics to the specified EViews preset values, where \( n1 \) is an integer from 1–30. Simultaneously sets “linestyle”, “linepattern”, “linewidth”, “symbol”, “fillcolor”, “fillgray”, and “fillhatch” to the EViews predefined definitions for graph element \( n1 \).

When applied to boxplots, the line color of the specified element will be applied to the box, whiskers, and staples.
**Examples**

```plaintext
graph1.setelem(2) lcolor(blue) lwidth(2) symbol(circle)
```
sets the second line of GRAPH1 to be a blue line of width 2 with circle symbols.

```plaintext
graph1.setelem(1) lcolor(blue)
graph1.setelem(1) linecolor(0, 0, 255)
```
are equivalent methods of setting the linecolor to blue.

```plaintext
graph1.setelem(1) fillgray(6)
```
sets the gray-scale color for the first graph element.

The lines

```plaintext
graph1.setelem(1) scale(l)
graph1.setelem(2) scale(l)
graph1.setelem(3) scale(r)
```
create a dual scale graph where the first two series are scaled together and labeled on the left axis and the third series is scaled and labeled on the right axis.

```plaintext
graph1.setelem(2) legend("gross domestic product")
```
sets the legend for the second graph element.
Cross-references


See also axis (p. 197), datelabel (p. 237), scale (p. 397) and options (p. 343).

**setfillcolor** | Table Proc

Set the fill (background) color of the specified table cells.

**Syntax**

Table Proc: table_name.setfillcolor(cell_range) color_arg

where **cell_range** can take one of the following forms:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@all</td>
<td>Apply to all cells in the table.</td>
</tr>
<tr>
<td>cell</td>
<td>Cell identifier. You can identify cells using either the column letter and row number (e.g., “A1”), or by using “R” followed by the row number followed by “C” and the column number (e.g., “R1C2”). Apply to cell.</td>
</tr>
<tr>
<td>row[,] col</td>
<td>Row number, followed by column letter or number (e.g., “2,C”, or “2,3”), separated by “,”. Apply to cell.</td>
</tr>
<tr>
<td>row</td>
<td>Row number (e.g., “2”). Apply to all cells in the row.</td>
</tr>
<tr>
<td>col</td>
<td>Column letter (e.g., “B”). Apply to all cells in the column.</td>
</tr>
<tr>
<td>first_cell[,:]last_cell, first_cell[,]last_cell</td>
<td>Top left cell of the selection range (specified in “cell” format), followed by bottom right cell of the selection range (specified in “cell” format), separated by “;” or “,” (e.g., “A2:C10”, “A2,C10”, or “R2C1:R10C3”, “R2C1,R10C3”). Apply to all cells in the rectangular region defined by the first cell and last cell.</td>
</tr>
<tr>
<td>first_cell_row[,] first_cell_col[,] last_cell_row[,] last_cell_col</td>
<td>Top left cell of the selection range (specified in “row[,] col” format), followed by bottom right cell of the selection range (specified in “row[,] col” format), separated by a “,” (e.g., “2,A,10,C” or “2,1,10,3”). Apply to all cells in the rectangular region defined by the first cell and last cell.</td>
</tr>
</tbody>
</table>

The **color_arg** specifies the color to be applied to the text in the cells. The color may be specified using predefined color names, or by specifying the individual red-green-blue...
(RGB) components using the special “@RGB” function. The latter method is obviously more difficult, but allows you to design your own custom colors.

The predefined colors are given by the keywords (with their RGB equivalents):

<table>
<thead>
<tr>
<th>Color</th>
<th>RGB Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>blue</td>
<td>@rgb(0, 0, 255)</td>
</tr>
<tr>
<td>red</td>
<td>@rgb(255, 0, 0)</td>
</tr>
<tr>
<td>green</td>
<td>@rgb(0, 128, 0)</td>
</tr>
<tr>
<td>black</td>
<td>@rgb(0, 0, 0)</td>
</tr>
<tr>
<td>white</td>
<td>@rgb(255, 255, 255)</td>
</tr>
<tr>
<td>purple</td>
<td>@rgb(128, 0, 128)</td>
</tr>
<tr>
<td>orange</td>
<td>@rgb(255, 128, 0)</td>
</tr>
<tr>
<td>yellow</td>
<td>@rgb(255, 255, 0)</td>
</tr>
<tr>
<td>gray</td>
<td>@rgb(128, 128, 128)</td>
</tr>
<tr>
<td>ltgray</td>
<td>@rgb(192, 192, 192)</td>
</tr>
</tbody>
</table>

**Examples**

To set a purple background color for the cell in the second row and third column of TAB1, you may use and of the following

```plaintext
tab1.setfillcolor(C2) @rgb(128, 0, 128)
tab1.setfillcolor(2,C) @RGB(128, 0, 128)
tab1.setfillcolor(2,3) purple
```

You may also specify a yellow color for the background of an entire column, or an entire row,

```plaintext
tab1.setfillcolor(C) @RGB(255, 255, 0)
tab1.setfillcolor(2) yellow
```

or for the background of the cells in a rectangular region

```plaintext
tab1.setfillcolor(R2C3:R3C6) ltgray
tab1.setfillcolor(r2c3,r3c6) ltgray
tab1.setfillcolor(2,C,3,F) @rgb(192, 192, 192)
tab1.setfillcolor(2,3,3,6) @rgb(192, 192, 192)
```

**Cross-references**

For additional discussion of tables see Chapter 4, “Working with Tables”.
See `settextcolor` (p. 430) and `setfont` (p. 418) for details on changing text color and font, and `setlines` (p. 427) for drawing lines between around and through cells. See `setformat` (p. 419) for setting cell formats.

### setfont

**Set the font for text in the specified table cells**

**Syntax**

```
Table Proc: table_name.setfont(cell_range) font_args
```

where `cell_range` describes the table cells to be modified, and `font_args` is a set of arguments used to modify the existing font settings. See `setfillcolor` (p. 416) for the syntax for `cell_range`.

The `font_args` may include one or more of the following:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>face_name</code></td>
<td>A string that specifies the typeface name of the font, enclosed in double quotes.</td>
</tr>
<tr>
<td><code>integer[pt]</code></td>
<td>Integer font size, in points, followed by the “pt” identifier (e.g., “12pt”).</td>
</tr>
<tr>
<td><code>+b / -b</code></td>
<td>[Add / remove] boldface.</td>
</tr>
<tr>
<td><code>+i / -i</code></td>
<td>[Add / remove] italics.</td>
</tr>
<tr>
<td><code>+s / -s</code></td>
<td>[Add / remove] strikethrough.</td>
</tr>
<tr>
<td><code>+u / -u</code></td>
<td>[Add / remove] underscore.</td>
</tr>
</tbody>
</table>

**Examples**

The command

```
tabl.setfont(B3:D10) "Times New Roman" +i
```

sets the font to Times New Roman Italic for the cells defined by the rectangle from B3 (row 3, column 2) to D10 (row 10, column 4), and

```
tabl.setfont(3, B, 10, D) 8pt
```

changes all of text in the region to 8 point.

```
tabl.setfont(4, B) +b -i
```

removes the italic, and adds boldface to the B4 cell (row 4, column 2).

The commands
modify the fonts for the column B, and row 2, respectively. The first command changes the point size to 14, removes strikethrough and adds underscoring. The second changes the typeface to Batang, and adds underscoring.

Cross-references

For additional discussion of tables see Chapter 4, “Working with Tables”.

See settextcolor (p. 430) and setfillcolor (p. 416) for details on changing the text and cell background colors. See setformat (p. 419) for setting the cell formats.

|-----------|------------|-----------|------------|-------------|-------------|----------------|---------|------------|-------------|

Set the display format for cells in object spreadsheet and table views.

Syntax

Group Proc:    group_name.setformat(col_range) format_arg
Table Proc:    table_name.setformat(cell_range) format_arg
Object Proc:   object_name.setformat format_arg

where format_arg is a set of arguments used to specify format settings. If necessary, you should enclose the format_arg in double quotes.

The cell_range option is used to describe the columns to be updated in groups and the cells to be modified in tables. For all other objects, setformat operates on all of the cells in the object.

For groups, the col_range may take one of the following forms:

- @all: Apply to all series in the group.
- col: Column number or letter (e.g., “2”, “B”). Apply to the series corresponding to the column.
- first_col[:last_col]: Colon delimited range of columns (from low to high, e.g., “3:5”). Apply to all series corresponding to the column range.

For tables, the cell_range may take one of the following forms:
To format numeric values, you should use one of the following format specifications:

- **g[.precision]**: significant digits
- **f[.precision]**: fixed precision
- **y[width][.precision]**: fixed characters
- **c[.precision]**: fixed precision, comma separated thousands
- **e[.precision]**: scientific/float
- **p[.precision]**: percentage
- **r[.precision]**: fraction

If you wish to display negative numbers surrounded by parentheses (*i.e.*, display the number -37.2 as “(-37.2)”), then you should enclose the format string in “()” (*e.g.*, “(f.8)”).

To format numeric values using date and time formats, you may use a subset of the possible date format strings (see the appendix in the *User’s Guide* for a full description). The
possible format arguments, along with an example of the date number 730856.944793113 (January 7, 2002 10:40:30.125 p.m) formatted using the argument are:

<table>
<thead>
<tr>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>WF</td>
<td>(uses current EViews workfile date display format)</td>
</tr>
<tr>
<td>YYYY</td>
<td>“2002”</td>
</tr>
<tr>
<td>YYYY-Mon</td>
<td>“2002-Jan”</td>
</tr>
<tr>
<td>YYYYMon</td>
<td>“2002 Jan”</td>
</tr>
<tr>
<td>YYYY(M)MM</td>
<td>“2002(M)01”</td>
</tr>
<tr>
<td>YYYYMM</td>
<td>“2002:01”</td>
</tr>
<tr>
<td>YYYY(Q)Q</td>
<td>“2002(Q)1”</td>
</tr>
<tr>
<td>YYYYQ</td>
<td>“2002:Q”</td>
</tr>
<tr>
<td>YYYY[S]S</td>
<td>“2002[S]1” (semi-annual)</td>
</tr>
<tr>
<td>YYYY:S</td>
<td>“2002:1”</td>
</tr>
<tr>
<td>YYYY-MM-DD</td>
<td>“2002-01-07”</td>
</tr>
<tr>
<td>YYYY Mon dd</td>
<td>“2002 Jan 7”</td>
</tr>
<tr>
<td>YYYY Month dd</td>
<td>“2002 January 7”</td>
</tr>
<tr>
<td>YYYY-MM-DD HH:MI</td>
<td>“2002-01-07 22:40”</td>
</tr>
<tr>
<td>YYYY-MM-DD HH:MI:SS</td>
<td>“2002-01-07 22:40:30”</td>
</tr>
<tr>
<td>Mon-YYYY</td>
<td>“Jan-2002”</td>
</tr>
<tr>
<td>Mon dd YYYY</td>
<td>“Jan 7 2002”</td>
</tr>
<tr>
<td>Mon dd, YYYY</td>
<td>“Jan 7, 2002”</td>
</tr>
<tr>
<td>Month dd YYYY</td>
<td>“January 7 2002”</td>
</tr>
<tr>
<td>Month dd, YYYY</td>
<td>“January 7, 2002”</td>
</tr>
<tr>
<td>MM/DD/YYYY</td>
<td>“01/07/2002”</td>
</tr>
<tr>
<td>mm/DD/YYYY</td>
<td>“1/07/2002”</td>
</tr>
<tr>
<td>mm/DD/YYYY HH:MI</td>
<td>“1/07/2002 22:40”</td>
</tr>
<tr>
<td>mm/DD/YYYY HH:MI:SS</td>
<td>“1/07/2002 22:40:30”</td>
</tr>
<tr>
<td>mm/dd/YYYY</td>
<td>“1/7/2002”</td>
</tr>
<tr>
<td>mm/dd/YYYY HH:MI</td>
<td>“1/7/2002 22:40”</td>
</tr>
<tr>
<td>mm/dd/YYYY JJ:MI:SS</td>
<td>“1/7/2002 22:40:30”</td>
</tr>
</tbody>
</table>
Examples

To set the format for a series or a matrix object to fixed 5-digit precision, simply provide the format specification:

```plaintext
testdata.setformat f.5
```

For groups, provide the column identifier, and the format. The commands,

```plaintext
groupdata.setformat(2) (f.7)  
groupdata.setformat(c) e.5
```

set the formats for the second and third series in the group.

You may use any of the date formats given above

```plaintext
ser2.setformat YYYYMon  
group1(d).setformat "YYYY-MM-DD HH:MI:SS.SSS"
```

For tables, you must provide a valid cell specification:

```plaintext
tab1.setformat(B2) hh:MI:SS.SSS

tab1.setformat(2,B,10,D) (g.3)

tab1.setformat(R2C2:R4C4) "dd/MM/YY HH:MI:SS.SSS"
```

Cross-references

For additional discussion of tables see Chapter 4, “Working with Tables”.

See also `setfont` (p. 418), `settextcolor` (p. 430) and `setfillcolor` (p. 416) for details on changing the text font and text and cell background colors.
setheight

Set the row height of rows in a table.

Syntax

Table Proc: table_name.setheight(row_range) height_arg

where row_range is either a single row number (e.g., “5”), a colon delimit ed range of rows
(from low to high, e.g., “3:5”), or the “@ALL” keyword, and height_arg specifies the
height unit value, where height units are specified in character heights. The character
height is given by the font-specific sum of the units above and below the baseline and the
leading, where the font is given by the default font for the current table (the EViews table
default font at the time the table was created). height_arg values may be non-integer val-
ues with resolution up to 1/10 of a height unit.

Examples

    tab1.setheight(2) 1

sets the height of row 2 to match the table font character height, while

    tab1.setheight(2) 1.5

increases the row height to 1-1/2 character heights.

Similarly, the commands

    tab1.setheight(2:4) 1
    tab1.setheight(2:4) 1.5

sets the heights for rows 2 through 4.

Cross-references

For additional discussion of tables see Chapter 4, “Working with Tables”.  
For details on setting the column widths in a table, see setwidth (p. 431).
Set the display indentation for cells in object spreadsheet and table views.

Syntax

Group Proc: group_name.setindent(col_range) indent_arg
Table Proc: table_name.setindent(cell_range) indent_arg
Object Proc: object_name.setindent indent_arg

where indent_arg is an indent value specified in 1/5 of a width unit. The width unit is computed from representative characters in the default font for the current table (the EViews table default font at the time the table was created), and corresponds roughly to a single character. Indentation is only relevant for non-center justified cells.

The default value is taken from the Global Defaults at the time the view or table is created.

The col_range option is used to describe the columns to be updated in groups and the cell_range defines the cells to be modified in tables. For all other objects, setformat operates on all of the cells in the object. See setformat (p. 419) for the syntax for group col_range and table cell_range specifications.

Examples

To set the justification for a series or a matrix object:

ser1.setindent 2
matrix1.setindent top 4

For groups, you should provide the column identifier, and the format. The commands,

group1.setindent(2) 3
group1.setindent(c) 2

set the formats for the second and third series in the group, while

group2.setindent(@all) 3

sets formats for all of the series.

For tables, you must provide a valid cell specification:

tabl.setindent(@all) 2
tabl.setindent(2,B,10,D) 4
tabl.setindent(R2C2:R4C4) 2
Cross-references

For additional discussion of tables see Chapter 4, “Working with Tables”.

See setwidth (p. 431) and setjust (p. 425) for details on setting spreadsheet and table widths and justification.

|---------|------------|-----------|------------|-------------|-------------|----------------|----------|------------|-------------|

Set the display justification for cells in object spreadsheet and table views.

Syntax

- **Group Proc**: group_name.setjust(col_range) format_arg
- **Table Proc**: table_name.setjust(cell_range) format_arg
- **Object Proc**: object_name.setjust format_arg

where `format_arg` is a set of arguments used to specify format settings. You should enclose the `format_arg` in double quotes if it contains any spaces or delimiters.

The `col_range` option is used to describe the columns to be updated in groups and the `cell_range` defines the cells to be modified in tables. For all other objects, setformat operates on all of the cells in the object. See setformat (p. 419) for the syntax for group `col_range` and table `cell_range` specifications.

The `format_arg` may be formed using the following:

- `top / middle / bottom` Vertical justification setting.
- `auto / left / center / right` Horizontal justification setting.
  - “Auto” uses left justification for strings, and right for numbers.

You may enter one or both of the justification settings. The default settings are object specific: series and matrix object defaults are taken from the Global Defaults for spreadsheet views; table defaults are taken from the original view when created by freezing a view, or as “middle bottom” for newly created tables.

Examples

To set the justification for a series or a matrix object:
ser1.setjust middle
matrix1.setjust top left

For groups, you should provide the column identifier, and the format. The commands,
group1.setjust(2) bottom center
group1.setjust(c) center middle

set the formats for the second and third series in the group, while

group2.setjust(@all) right

sets all of the series formats.

For tables, you must provide a valid cell specification:
tab1.setjust(@all) top
tab1.setjust(2,B,10,D) left bottom
tab1.setjust(R2C2:R4C4) right top

Cross-references

For additional discussion of tables see Chapter 4, “Working with Tables”.

See setwidth (p. 431) and setindent (p. 424) for details on setting spreadsheet and table widths and indentation.

<table>
<thead>
<tr>
<th>Command</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>setline</td>
<td></td>
</tr>
</tbody>
</table>

Place a double horizontal line in a table.

Provided for backward compatibility. For a more general method of setting the line characteristics and borders for a set of table cells, see the table proc setlines (p. 427).

Syntax

Command: setline(table_name, r)

Options

Specify the name of the table and the row number r in which to place the horizontal line.

Examples

setline(tab3,8)

places a (double) horizontal line in the eighth row of the table object TAB3.
Cross-references

Chapter 4, “Working with Tables”, on page 47 describes table formatting using commands. See also Chapter 14, “Graphs, Tables, and Text Objects”, on page 387 of the User’s Guide for a discussion and examples of table formatting in EViews.

See setlines (p. 427).

<table>
<thead>
<tr>
<th>setlines</th>
<th>Table Proc</th>
</tr>
</thead>
</table>

Sets line characteristics and borders for a set of table cells.

Syntax

Table Proc: `table_name.setlines(cell_range) line_args`

where `cell_range` describes the table cells to be modified, and `line_args` is a set of arguments used to modify the existing line settings. See setfillcolor (p. 416) for the syntax for `cell_range`.

The `line_args` may contain one or more of the following:

- `+t / -t` Top border [on/off].
- `+b / -b` Bottom border [on/off].
- `+l / -l` Left border [on/off].
- `+r / -r` Right border [on/off].
- `+i / -i` Inner borders [on/off].
- `+o / -o` Outer borders [on/off].
- `+v / -v` Vertical inner borders [on/off].
- `+h / -h` Horizontal inner borders [on/off].
- `+a / -a` All borders [on/off].
- `+d / -d` Double middle lines [on/off].

Examples

The command

```
tabl.setlines(b2:d6) +o
```

draws borders around the outside of the rectangle defined by B2 and D6. Note that this is equivalent to a command
tabl.setlines(b2:d6) +a -h -v
which adds borders to all of the cells in the rectangle defined by B2 and D6, then removes the inner horizontal and vertical borders.

tabl.setlines(2,b) +o
puts a border around all four sides of the B2 cell.

tabl.setlines(2,b) -l -r +i
then removes both the left and the right border from the cell. Note that in this example, “+i” option has no effect; since the specification involves a single cell, there are no inner borders.

tabl.setlines(@all) -a
removes all borders from the table.

Cross-references
For additional discussion of tables see Chapter 4, “Working with Tables”.
See also settextcolor (p. 430) for details on setting cell colors. See setmerge (p. 428) for cell merging, and comment (p. 226) for adding comments to cells.

<table>
<thead>
<tr>
<th>setmerge</th>
<th>Table Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merges/unmerges one or more table cells.</td>
<td></td>
</tr>
</tbody>
</table>

Syntax

Table Proc: table_name.setmerge(cell_range)
where cell_range describes the table cells (in a single row) to be merged. The cell_range specifications are given by:
If the first specified column is less than the last specified column (left specified before right), the cells in the row will be merged left to right, otherwise, the cells will be merged from right to left. The contents of the merged cell will be taken from the first non-empty cell in the merged region. If merging from left to right, the leftmost cell contents will be used; if merging from right to left, the rightmost cell contents will be displayed.

If you specify a merge involving previously merged cells, EViews will unmerge all cells within the specified range.

**Examples**

```plaintext
tab1.setmerge(a2:d2)
tab1.setmerge(2,1,2,4)
```

merges the cells in row 2, columns 1 to 4, from left to right, while

```plaintext
tab2.setmerge(r2c5:r2c2)
```

merges the cells in row 2, columns 2 to 5, from right to left. We may then unmerge cells by issuing the command using any of the previously merged cells:

```plaintext
tab2.setmerge(r2c4:r2c4)
```

unmerges the previously merged cells.

Note that in all cases, the `setmerge` command must be specified using cells in a single row. The command

```plaintext
tab3.setmerge(r2c1:r3c5)
```

generates an error since the cell specification involves rows 2 and 3.
Cross-references

For additional discussion of tables see Chapter 4, “Working with Tables”.

See also *setlines* (p. 427).

<table>
<thead>
<tr>
<th>Command</th>
<th>Table Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>settextcolor</em></td>
<td></td>
</tr>
</tbody>
</table>

Changes the text color of the specified table cells.

**Syntax**

```
Table Proc:   table_name.settextcolor(cell_range) color_arg
```

where `cell_range` describes the table cells to be modified, and `color_arg` specifies the color to be applied to the text in the cells. See *setfillcolor* (p. 416) for the syntax for `cell_range` and `color_arg`.

**Examples**

To set an orange text color for the cell in the second row and sixth column of TAB1, you may use

```
TAB1.settextcolor(f2) @rgb(255, 128, 0)
TAB1.settextcolor(2,f) @RGB(255, 128, 0)
TAB1.settextcolor(2,6) orange
TAB1.settextcolor(r2c6) orange
```

You may also specify a blue color for the text in an entire column, or an entire row,

```
TAB1.settextcolor(C) @RGB(0, 0, 255)
TAB1.settextcolor(2) blue
```

or a green color for the text in cells in a rectangular region

```
TAB1.settextcolor(R2C3:R3C6) green
TAB1.settextcolor(r2c3,r3c6) green
TAB1.settextcolor(2,C,3,F) @rgb(0, 255, 0)
TAB1.settextcolor(2,3,3,6) @rgb(0, 255, 0)
```

**Cross-references**

For additional discussion of tables see Chapter 4, “Working with Tables”.

See *setfont* (p. 418) and *setfillcolor* (p. 416) for details on changing the text font and cell background color.
setwidth

Set the column width for selected columns in an object spreadsheet.

Syntax

Object Proc: object_name.setwidth width_arg
Object Proc: object_name.setwidth(col_range) width_arg

where col_range is either a single column number or letter (e.g., “5”, “E”), a colon delimited range of columns (from low to high, e.g., “3:5”, “C:E”), or the keyword “@ALL”, and width_arg specifies the width unit value. The width unit is computed from representative characters in the default font for the current table (the EViews table default font at the time the table was created), and corresponds roughly to a single character. width_arg values may be non-integer values with resolution up to 1/10 of a width unit.

Examples

  tab1.setwidth(2) 12
sets the width of column 2 to 12 width units.

The command

  tab1.setwidth(2:10) 20
sets the widths for columns 2 through 10 to 20 width units.

Cross-references

See setindent (p. 424) and setjust (p. 425) for details on setting spreadsheet and table indentation and justification. For details on setting the row heights in a table, see setheight (p. 423).

For additional discussion of tables see Chapter 4, "Working with Tables". See also Chapter 14 of the User’s Guide for a discussion and examples of table formatting in EViews.
Spreadsheet view of object.

The `sheet` view displays the spreadsheet view of the named object. For table objects, `sheet` simply displays the table.

**Syntax**

Object View: `object_name.sheet(options)`

Pool View: `pool_name.sheet(options) pool_ser1 [pool_ser2 pool_ser3 ...]`

The `sheet` view of pool objects displays the spreadsheet view of the series in the pool. Follow the word `sheet` by a list of series to display; you may use the cross section identifier “?” in the series name.

**Options**

- `p` Print the spreadsheet view.

**Examples**

```
  tab1.sheet  
```

displays the spreadsheet view of TAB1.

```
  pool1.sheet x? y? z?  
```

displays the pool spreadsheet view of the series `X?`, `Y?`, and `Z?`.

**Cross-references**

See Chapter 5 of the *User's Guide* for a discussion of the spreadsheet view of series and groups, and Chapter 27 of the *User's Guide* for a discussion of pools. See also the chapters for the other individual objects.

**show**

Display objects.

The `show` command displays series or other objects on your screen. A scalar object is displayed in the status line at the bottom of the EViews window.
Syntax
  Command:  show object_name.view_command
  Command:  show object1 [object2 object3 ...]

The command show should be followed by the name of an object, or an object name with an attached view.

For series and graph objects, show can operate on a list of names. The list of names must be of the same type. show creates and displays an untitled group or multiple graph object.

Examples
  genr x=nrnd
  show x.hist
  close x

generates a series X of random draws from a standard normal distribution, displays the histogram view of X, and closes the series window.

  show wage log(wage)

opens an untitled group window with the spreadsheet view of the two series.

  freeze(gra1) wage.hist
  genr lwage=log(wage)
  freeze(gra2) lwage.hist
  show gra1 gra2

opens an untitled graph object with two histograms.

Cross-references
See Chapter for a complete listing of the views of the various objects.
See also close (p. 222).

<table>
<thead>
<tr>
<th>signalgraphs</th>
<th>Sspace View</th>
</tr>
</thead>
</table>

Graph signal series.

Display graphs of a set of signal series computed using the Kalman filter.

Syntax
  Sspace View:  object_name.signalgraphs(options)
Options

\[ t = output\_type \]  
\[ \text{(default = “pred”)} \]


Examples

\[ \text{ss1.signalgraphs}(t=\text{smooth}) \]
\[ \text{ss1.signalgraphs}(t=\text{smoothse}) \]

displays a graph view containing the smoothed signal values, and then displays a graph view containing the root MSE of the smoothed states.

Cross-references


See also stategraphs (p. 446), makesignals (p. 330) and makestates (p. 331).

<table>
<thead>
<tr>
<th>smooth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
</tr>
</tbody>
</table>

Exponential smoothing.

\text{smooth} forecasts a series using one of a number of exponential smoothing techniques. By default, \text{smooth} estimates the damping parameters of the smoothing model to minimize the sum of squared forecast errors, but you may specify your own values for the damping parameters.

\text{smooth} automatically calculates in-sample forecast errors and puts them into the series \text{RESID}.

Syntax

Command: \[ \text{smooth}(method) \text{ series\_name smooth\_name \{freq\}} \]
Series Proc: \[ \text{series\_name.smooth}(method) \text{ smooth\_name \{freq\}} \]
You should follow the `smooth` keyword with the name of the series and a name for the smoothed series. You must specify the smoothing method in parentheses as an option. The optional `freq` may be used to override the default for the number of periods in the seasonal cycle. By default, this value is set to the workfile frequency (e.g., 4 for quarterly data). For undated data, the default is 5.

**Options**

*Smoothing method options*

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s[,x]</td>
<td>Single exponential smoothing for series with no trend. You may optionally specify a number ( x ) between zero and one for the mean parameter.</td>
</tr>
<tr>
<td>d[,x]</td>
<td>Double exponential smoothing for series with a trend. You may optionally specify a number ( x ) between zero and one for the mean parameter.</td>
</tr>
<tr>
<td>n[,x,y]</td>
<td>Holt-Winters without seasonal component. You may optionally specify numbers ( x ) and ( y ) between zero and one for the mean and trend parameters, respectively.</td>
</tr>
<tr>
<td>a[,x,y,z]</td>
<td>Holt-Winters with additive seasonal component. You may optionally specify numbers ( x ), ( y ), and ( z ), between zero and one for the mean, trend, and seasonal parameters, respectively.</td>
</tr>
<tr>
<td>m[,x,y,z]</td>
<td>Holt-Winters with multiplicative seasonal component. You may optionally specify numbers ( x ), ( y ), and ( z ), between zero and one for the mean, trend, and seasonal parameters, respectively.</td>
</tr>
</tbody>
</table>

*Other Options:*

- **p** Print a table of forecast statistics.

If you wish to set only some of the damping parameters and let EViews estimate the other parameters, enter the letter “e” where you wish the parameter to be estimated.

If the number of seasons is different from the frequency of the workfile (an unusual case that arises primarily if you are using an undated workfile for data that are not monthly or quarterly), you should enter the number of seasons after the smoothed series name. This optional input will have no effect on forecasts without seasonal components.
Examples

\[ \text{smooth}(s) \text{ sales sales}_f \]

smooths the SALES series by a single exponential smoothing method and saves the smoothed series as SALES_F. EViews estimates the damping (smoothing) parameter and displays it with other forecast statistics in the SALES series window.

\[ \text{smooth}(n,e,.3) \text{ tb3 tb3}_\text{hw} \]

smooths the TB3 series by a Holt-Winters no seasonal method and saves the smoothed series as TB3_HW. The mean damping parameter is estimated while the trend damping parameter is set to 0.3.

\[ \text{smpl} \ @\text{first} @\text{last-10} \]
\[ \text{smooth}(m,.1,.1,.1) \text{ order order}_\text{hw} \]
\[ \text{smpl} @\text{all} \]
\[ \text{graph} \text{ gra1.line order order}_\text{hw} \]
\[ \text{show} \text{ gra1} \]

smooths the ORDER series by a Holt-Winters multiplicative seasonal method leaving the last 10 observations. The damping parameters are all set to 0.1. The last three lines plot and display the actual and smoothed series over the full sample.

Cross-references


<table>
<thead>
<tr>
<th>smpl</th>
<th>Command</th>
</tr>
</thead>
</table>

Set sample range.

The smpl command sets the workfile sample to use for statistical operations and series assignment expressions.

Syntax

Command: \[ \text{smpl} \text{ smpl_spec} \]
Command: \[ \text{smpl} \text{ sample_name} \]

List the date or number of the first observation and the date or number of the last observation for the sample. Rules for specifying dates are given in “Dates” on page 875 of the User’s Guide. The sample spec may contain more than one pair of beginning and ending observations.
The `smpl` command also allows you to select observations on the basis of conditions specified in an `if` statement. This enables you to use logical operators to specify what observations to include in EViews' procedures. Put the `if` statement after the pairs of dates.

You can also use `smpl` to set the current observations to the contents of a named sample object; put the name of the sample object after the keyword.

**Special keywords for smpl**

The following “@-keywords” can be used in a `smpl` command:

- `@all` The whole workfile range.
- `@first` The first observation in the workfile.
- `@last` The last observation in the workfile.

**Examples**

```
smpl 1955:1 1972:12
```

sets the workfile sample from 1955:1 to 1972:12

```
smpl @first 1940 1946 1972 1975 @last
```

excludes observations (or years) 1941–1945 and 1973–1974 from the workfile sample.

```
smpl if union=1 and edu<=15
```

sets the sample to those observations where UNION takes the value 1 and EDU is less than or equal to 15.

```
sample half @first @first+@obs(x)/2
smpl half
smpl if x>0
smpl @all if x>0
```

The first line declares a sample object named HALF which includes the first half of the series X. The second line sets the sample to HALF and the third line sets the sample to those observations in HALF where X is positive. The last line sets the sample to those observations where X is positive over the full sample.

**Cross-references**

See “Samples” on page 81 of the User's Guide for a discussion of samples in EViews.

See also `set` (p. 407) and `sample` (p. 392).
solve  Command || Model Proc

Solve the model.

`solve` finds the solution to a simultaneous equation model for the set of observations specified in the current workfile sample.

**Syntax**

Command: `solve(options) model_name`

Model Proc: `model_name.solve(options)`

Note: When `solve` is used in a program (batch mode) models are always solved over the workfile sample. If the model contains a solution sample, it will be ignored in favor of the workfile sample.

You should follow the name of the model after the `solve` command or use `solve` as a procedure of a named model object. The default solution method is dynamic simulation. You may modify the solution method as an option.

`solve` first looks for the specified model in the current workfile. If it is not present, `solve` attempts to fetch a model file (.DBL) from the default directory or, if provided, the path specified with the model name.

**Options**

`solve` can take any of the options available in `solveopt` (p. 439).

**Examples**

```plaintext
solve mod1
```

solves the model MOD1 using the default solution method.

```plaintext
nonlin2.solve(m=500,e)
```

solves the model NONLIN2 with an extended search of up to 500 iterations.

**Cross-references**


See also `model` (p. 339), `msg` (p. 339) and `solveopt` (p. 439).
**solveopt**

Solve options for models.

`solveopt` sets options for model solution but does not solve the model. The same options can be set directly in a `solve` procedure.

### Syntax

```
Model Proc: model_name.solveopt(options)
```

### Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>d = arg</td>
<td>Model solution dynamics: “d” (dynamic solution), “s” (static solution), “f” (fitted values – single equation solution).</td>
</tr>
<tr>
<td>m = integer</td>
<td>Maximum number of iterations for solution (maximum 100,000).</td>
</tr>
<tr>
<td>c = number</td>
<td>Convergence criterion. Based upon the maximum change in any of the endogenous variables in the model. You may set a number between 1e-15 and 0.01.</td>
</tr>
<tr>
<td>r = integer</td>
<td>Number of stochastic repetitions (used with stochastic “s = ” options).</td>
</tr>
<tr>
<td>b = number</td>
<td>Size of stochastic confidence intervals (used with stochastic “s = ” options).</td>
</tr>
<tr>
<td>a = arg</td>
<td>Alternate scenario solution: “t” (true - solve both active and alternate scenario and collect deviations for stochastic), “f” (false - solve only the active scenario).</td>
</tr>
<tr>
<td>o = arg</td>
<td>Solution method: “g” (Gauss-Seidel), “e” (Gauss-Seidel with extended search/reduced step size), “n” (Newton), “m” (Newton with extended search/reduced step size).</td>
</tr>
</tbody>
</table>
Cross-references


See also model (p. 339), msg (p. 339) and solve (p. 438).

**sort**

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort(options) arg1 [arg2 arg3...]</td>
</tr>
</tbody>
</table>

Sort the workfile.

The `sort` command sorts all series in the workfile on the basis of the values of one or more of the series. For purposes of sorting, NAs are considered to be smaller than any other value. By default, EViews will sort the series in ascending order. You may use options to override the sort order.

EViews will first remove any workfile structures and then will sort the workfile using the specified settings.

**Syntax**

Command:   `sort(options) arg1 [arg2 arg3...]`
List the name of the series or groups by which you wish to sort the workfile. If you list two or more series, `sort` uses the values of the second series to resolve ties from the first series, and values of the third series to resolve ties from the second, and so on.

**Options**

- `d` sort in descending order.

**Examples**

```
sort(d) inc
```

sorts all series in the workfile in order of the INC series with the highest value of INC first. NAs in INC (if any) will be placed at the bottom.

```
sort gender race wage
```

sorts all series in the workfile in order of the values of GENDER from low to high, with ties resolved by ordering on the basis of RACE, with further ties resolved by ordering on the basis of WAGE.

**Cross-references**

See “Sorting a Workfile” on page 238 of the *User’s Guide*.

---

**spec**

Display the text specification view for logl, model, sspace, system objects.

**Syntax**

```
Object View: object_name.spec(options)
```

**Options**

- `p` Print the specification text.

**Examples**

```
modell.spec
```

displays the specification of the object MODEL1.

**Cross-references**

See also [append](p. 186), [merge](p. 336), [text](p. 469).
Display spike graph view of data, or change existing graph object type to spike.

The spike graph view of a series or group creates spike graphs for all specified series or matrix columns.

**Syntax**

Command: `spike arg1 [arg2 arg3 ...]`

Object View: `object_name.spike(options)`

Graph Proc: `graph_name.spike(options)`

**Options**

**Template and printing options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>o = <code>graph_name</code></td>
<td>Use appearance options from the specified graph.</td>
</tr>
<tr>
<td>t = <code>graph_name</code></td>
<td>Use appearance options and copy text and shading from the specified graph.</td>
</tr>
<tr>
<td>p</td>
<td>Print the spike graph.</td>
</tr>
</tbody>
</table>

**Scale options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a (default)</td>
<td>Automatic single scale.</td>
</tr>
<tr>
<td>d</td>
<td>Dual scaling with no crossing. The first series is scaled on the left and all other series are scaled on the right.</td>
</tr>
<tr>
<td>x</td>
<td>Dual scaling with possible crossing. See the “d” option.</td>
</tr>
<tr>
<td>n</td>
<td>Normalized scale (zero mean and unit standard deviation). May not be used with the “s” option.</td>
</tr>
<tr>
<td>s</td>
<td>Stacked spike graph. Each segment represents the cumulative total of the series listed (may not be used with the “l” option).</td>
</tr>
<tr>
<td>l</td>
<td>Spike graph for the first series listed and a line graph for all subsequent series (may not be used with the “s” option).</td>
</tr>
</tbody>
</table>
m  Plot spikes in multiple graphs (will override the “l” option). Not for use with an existing graph object.

Panel options

The following options apply when graphing panel structured data.

| panel = arg (default taken from global settings) | Panel data display: “stack” (stack the cross-sections), “individual” or “1” (separate graph for each cross-section), “combine” or “c” (combine each cross-section in single graph; one time axis), “mean” (plot means across cross-sections), “mean1se” (plot mean and +/- 1 standard deviation summaries), “mean2sd” (plot mean and +/- 2 s.d. summaries), “mean3sd” (plot mean and +/- 3 s.d. summaries), “median” (plot median across cross-sections), “med25” (plot median and +/- .25 quantiles), “med10” (plot median and +/- .10 quantiles), “med05” (plot median +/- .05 quantiles), “med025” (plot median +/- .025 quantiles), “med005” (plot median +/- .005 quantiles), “median” or “maxmin” (plot median, max and min). |

Examples

```
  group g1 gdp cons m1
  g1.spike(d)
```

plots line graphs of the three series in group G1 using dual scaling.

```
  matrix1.spike(t=mygra)
```

displays spike graphs of the columns of MATRIX1 using the graph object MYGRA as a template.

```
  graph1.spike(d)
```

changes GRAPH1 so that it contains spike graphs of each of the series in the original graph, using dual scaling.

Cross-references

See also graph (p. 288) for graph declaration and additional graph types.
## sspace

**Object Declaration**

Declare state space object.

**Syntax**

```verbatim
cmd: sspace sspace_name
```

Follow the `sspace` keyword with a name to be given the sspace object.

**Examples**

```
sspace stsp1
```
declares a sspace object named STSP1.

```
sspace tvp
tvp.append cs = c(1) + sv1*inc
tvp.append @state sv1 = sv1(-1) + [var=c(2)]
tvp.ml
```
declares a sspace object named TVP, specifies a time varying coefficient model, and estimates the model by maximum likelihood.

**Cross-references**

See Chapter 25 of the *User’s Guide* for a discussion of state space models. The `sspace` object is documented in greater detail beginning on page 161.

`append` (p. 186) may be used to add lines to an existing sspace object. See also `ml` (p. 338) for estimation of state space models.

## statby

**Series View**

Basic statistics by classification.

The `statby` view displays descriptive statistics for the elements of a series classified into categories by one or more series.

**Syntax**

```verbatim
Series View: series_name.statby(options) classifier_names
```

Follow the series name with a period, the `statby` keyword, and a name (or a list of names) for the series or group by which to classify. The options control which statistics to
display and in what form. By default, `statby` displays the means, standard deviations, and counts for the series.

Options

**Options to control statistics to be displayed**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sum</code></td>
<td>Display sums.</td>
</tr>
<tr>
<td><code>med</code></td>
<td>Display medians.</td>
</tr>
<tr>
<td><code>max</code></td>
<td>Display maxima.</td>
</tr>
<tr>
<td><code>min</code></td>
<td>Display minima.</td>
</tr>
<tr>
<td><code>quant</code></td>
<td>Display quantile with value given by the argument.</td>
</tr>
<tr>
<td><code>quant=</code></td>
<td>Specify quantile calculation method: <code>arg</code></td>
</tr>
<tr>
<td><code>q=</code></td>
<td>Compute quantiles using the specified definition:</td>
</tr>
<tr>
<td></td>
<td><code>arg</code> (default = .5)</td>
</tr>
<tr>
<td><code>skew</code></td>
<td>Display skewness.</td>
</tr>
<tr>
<td><code>kurt</code></td>
<td>Display kurtosis.</td>
</tr>
<tr>
<td><code>na</code></td>
<td>Display counts of NAs.</td>
</tr>
<tr>
<td><code>nomean</code></td>
<td>Do not display means.</td>
</tr>
<tr>
<td><code>nostd</code></td>
<td>Do not display standard deviations.</td>
</tr>
<tr>
<td><code>nocount</code></td>
<td>Do not display counts.</td>
</tr>
</tbody>
</table>

**Options to control layout**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>l</code></td>
<td>Display in list mode (for more than one classifier).</td>
</tr>
<tr>
<td><code>nor</code></td>
<td>Do not display row margin statistics.</td>
</tr>
<tr>
<td><code>noc</code></td>
<td>Do not display column margin statistics.</td>
</tr>
<tr>
<td><code>nom</code></td>
<td>Do not display table margin statistics (unconditional tables); for more than two classifier series.</td>
</tr>
<tr>
<td><code>nos</code></td>
<td>Do not display sub-margin totals in list mode; only used with “l” option and more than two classifier series.</td>
</tr>
<tr>
<td><code>sp</code></td>
<td>Display sparse labels; only with list mode option, “l”.</td>
</tr>
</tbody>
</table>
Options to control binning

- `dropna (default)`, `keepna` [Drop/Keep] NA as a category.
- `v = integer (default = 100)` Bin categories if classification series take on more than the specified number of distinct values.
- `nov` Do not bin based on the number of values of the classification series.
- `a = number (default = 2)` Bin categories if average cell count is less than the specified number.
- `noa` Do not bin based on the average cell count.
- `b = integer (default = 5)` Set maximum number of binned categories.

Other options

- `p` Print the descriptive statistics table.

Examples

```plaintext
wage.statby(max,min) sex race
```
displays the mean, standard deviation, max, and min of the series WAGE by (possibly binned) values of SEX and RACE.

Cross-references

See “By-Group Statistics” on page 579 for functions to compute by-group statistics. See also “Stats by Classification” on page 288, and “Descriptive Statistics” on page 355 of the User’s Guide.

See also `hist` (p. 294), `boxplotby` (p. 205) and `linkto` (p. 310).

<table>
<thead>
<tr>
<th><code>stategraphs</code></th>
<th>Sspace View</th>
</tr>
</thead>
</table>

Display graphs of a set of state series computed using the Kalman filter.

Syntax

```plaintext
Sspace View: sspace_name.stategraph(options)
```
Options

\[ t = \text{output\_type} \]

`(default = \text{“pred”})`


Other options

- \text{p} // Print the view.

Examples

\[ \text{ssl.stategraphs}(t = \text{filt}) \]

displays a graph view containing the filtered state values.

Cross-references


See also \text{signalgraphs (p. 433)}, \text{makesignals (p. 330)} and \text{makestates (p. 331)}.

<table>
<thead>
<tr>
<th>statefinal</th>
<th>Sspace View</th>
</tr>
</thead>
</table>

Display final state values.

Show the one-step ahead state predictions or the state prediction covariance matrix at the final values \((T + 1\mid T)\) where \(T\) is the last observation in the estimation sample. By default, EViews shows the state predictions.

Syntax

\[ \text{Sspace View: sspace\_name.statefinal(options)} \]
Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Display the state prediction covariance matrix.</td>
</tr>
<tr>
<td>p</td>
<td>Print the view.</td>
</tr>
</tbody>
</table>

Examples

```c
ss1.statefinal(c)
```

displays a view containing the final state covariances (the one-step ahead covariances for the first out-of-(estimation) sample period.

Cross-references


See also `stateinit` (p. 448).

---

### stateinit

Display initial state values.

Show the state initial values or the state covariance initial values used to initialize the Kalman Filter. By default, EViews shows the state values.

Syntax

```c
Sspace View: sspace_name.stateinit(options)
```

Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Display the covariance matrix.</td>
</tr>
<tr>
<td>p</td>
<td>Print the view.</td>
</tr>
</tbody>
</table>

Examples

```c
ss1.stateinit
```

displays a view containing the initial state values (the one-step ahead predictions for the first period).

Cross-references


See also `statefinal` (p. 447).
stats Command || Coef View | Group View | Matrix View | Rowvector View | Sym View | Valmap View | Vector View

Descriptive statistics.

Computes and displays a table of means, medians, maximum and minimum values, standard deviations, and other descriptive statistics of one or more series or a group of series, or displays summary statistics for the definitions of a valmap.

When used as a command, stats creates an untitled group containing all of the specified series and opens a statistics view of the group. By default, if more than one series is given, the statistics are calculated for the common sample.

When used for coef or matrix objects, stats computes the statistics for each column of data.

Syntax

Command: \texttt{stats(options) ser1 [ser2 ser3 \ldots]}
Object View: \texttt{object\_name.stats(options)}

Options

\begin{itemize}
  \item \texttt{i} \quad \text{Individual sample for each series (for group only), by default uses a common sample.}
  \item \texttt{p} \quad \text{Print the stats table.}
\end{itemize}

Examples

\begin{itemize}
  \item \texttt{stats height weight age}
  \quad opens an untitled group window displaying the histogram and descriptive statistics for the common sample of the three series.
  \item \texttt{group group1 wage hrs edu}
  \quad \texttt{group1.stats(i)}
  \quad displays the descriptive statistics view of \texttt{GROUP1} for the individual samples.
  \item \texttt{map1.stats}
  \quad displays the summary descriptive view of the definitions in the valmap \texttt{MAP1}.\end{itemize}
Cross-references


See also boxplot (p. 204) and hist (p. 294).

statusline

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>statusline</td>
</tr>
</tbody>
</table>

Send text to the status line.

Displays a message in the status line at the bottom of the EViews main window. The message may include text, control variables, and string variables.

Syntax

```plaintext
Command:  statusline message_string
```

Examples

```plaintext
statusline Iteration Number: !t
```

Displays the message “Iteration Number: !t” in the status line replacing “!t” with the current value of the control variable in the program.

Cross-references

See Chapter 6, “EViews Programming”, on page 83 for a discussion and examples of programs, control variables and string variables.

stom

<table>
<thead>
<tr>
<th>Group Proc</th>
<th>Series Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td>stom</td>
<td></td>
</tr>
</tbody>
</table>

Convert a series or group to a vector or matrix.

Fills a vector or matrix with the data from a series or group.

Syntax

```plaintext
Series Proc:  stom(series_name, vector[, sample])
Group Proc:   stom(group_name, matrix[, sample])
```

Include the series or group name in parentheses followed by a comma and the vector or matrix name. By default, the series values in the current workfile sample are used to fill the vector or matrix; you may optionally provide an alternative sample.

There are two important features of stom that you should keep in mind:
• If any of the series contain NAs, those observations will be dropped from the vector/matrix (for alternative behavior, see `stomna (p. 451)`).
• If the vector or matrix already exists in the workfile, EViews automatically resizes the vector/matrix to fit the series/group.

Examples

```plaintext
series lwage=log(wage)
stom(lwage,vec1)
```

converts the series LWAGE into a vector named VEC1 using the current workfile sample. Any NAs in LWAGE will be dropped from VEC1.

```plaintext
group rhs x1 x2 x3
sample s1 1951 1990
stom(rhs,x,s1)
```

converts a group of three series named X1, X2 and X3 to a matrix named X using sample S1. The matrix X will have 40 rows and 3 columns (provided there are no NAs).

Cross-references

See also `stomna (p. 451)` and `mtos (p. 544)`.

---

### stomna

Convert a series or group to a vector or matrix without dropping NAs.

Fills a vector or matrix with the data from a series or group without dropping observations with missing values.

Works in identical fashion to `stom`, above, but does not drop observations containing NAs.

**Syntax**

**Series Proc:**

```
stomna(series, vector[, sample])
```

**Group Proc:**

```
stomna(group, matrix[, sample])
```

Include the series or group name in parentheses followed by a comma and the vector or matrix name. By default, the series values in the current workfile sample are used to fill the vector or matrix; you may optionally provide an alternative sample.
Examples

series lwage = log(wage)
stomna(lwage, vec1)

converts the series LWAGE into a vector named VEC1 using the current workfile sample. Any NAs in LWAGE will be placed in VEC1.

group rhs x1 x2 x3
sample s1 1951 1990
stom(rhs, x, s1)

converts a group of three series RHS to a matrix named X using sample S1. The matrix X will always have 40 rows and 3 columns.

Cross-references


See also stom (p. 450) and mtos (p. 544).

| store | Command || Pool Proc |
|-------|----------|------------|

Store objects in databases and databank files.

Stores one or more objects in the current workfile in EViews databases or individual databank files on disk. The objects are stored under the name that appears in the workfile.

When used as a pool proc, EViews will first expand the list of series using the pool operator, and then perform the fetch.

Syntax

Command: store(options) object_list
Pool Proc: pool_name.store(options) pool_ser1 [pool_ser2 pool_ser3 ...]

Follow the store command keyword with a list of the object names (each separated by a space) that you wish to store. The default is to store the objects in the default database. (This behavior is a change from EViews Version 2 and earlier where the default was to store objects in individual databank files).

You can precede the object name with a database name and the double colon “::” to indicate a specific database. You can also specify the database name as an option in parentheses, in which case all objects without an explicit database name will be stored in the specified database.
When used as a command, you may use wild card characters “?” (to match any single character) or “*” (to match zero or more characters) in the object name list. All objects with names matching the pattern will be stored. You may not use “?” as a wildcard character if `store` is being used as a pool proc since this conflicts with the pool identifier.

You can optionally choose to store the listed objects in individual databank files. To store in files other than the default path, you should include a path designation before the object name.

Options

| `d = db_name` | Store to the specified database. |
| `i`           | Store to individual databank files. |
| `1 / 2`       | Store series in [single / double] precision to save space. |
| `o`           | Overwrite object in database (default is to merge data, where possible). |
| `g = arg`     | Group store from workfile to database: “s” (copy group definition and series as separate objects), “t” (copy group definition and series as one object), “d” (copy series only as separate objects), “l” (copy group definition only). |

If you do not specify the precision option (1 or 2), the global option setting will be used. See “Database Registry / Database Storage Defaults” on page 900 of the User’s Guide.

Examples

```
store ml gdp unemp
```

stores the three objects M1, GDP, UNEMP in the default database.

```
store(d=us1) ml gdp macro::unemp
```

stores M1 and GDP in the US1 database and UNEMP in the MACRO database.

```
store usdat::gdp macro::gdp
```

stores the same object GDP in two different databases USDAT and MACRO.

```
store(1) cons*
```

stores all objects with names starting with CONS in the default database. The “1” option uses single precision to save space.
store(i) ml c:\data\unemp

stores M1 and UNEMP in individual databank files.

Cross-references


See also fetch (p. 264) and copy (p. 228).

<table>
<thead>
<tr>
<th>structure</th>
<th>Sspace View</th>
</tr>
</thead>
</table>

Display summary of sspace specification.

Show view which summarizes the system transition matrices or the covariance structure of the state space specification. EViews can display either the formulae or the values of the system transition matrices or covariance. By default, EViews will display the formulae for the system matrices.

Syntax

Sspace View: sspace_name.\texttt{structure}(options) [argument]

If you choose to display the values for a time-varying system using the “v” option, you should use the optional [argument] to specify a single date at which to evaluate the matrices. If none is provided, EViews will use the first date in the current sample.

Options

- \texttt{v} Display the values of the system transition or covariance matrices.
- \texttt{c} Display the system covariance matrix.
- \texttt{p} Print the view.

Examples

\texttt{ss1.structure}

displays a system transition matrices.

\texttt{ss1.structure 1993:4}

displays the transition matrices evaluated at 1993:4.
Cross-references


<table>
<thead>
<tr>
<th>SUR</th>
<th>System Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Estimate a system object using seemingly unrelated regression (SUR).

Note that the EViews procedure is more general than textbook versions of SUR since the system of equations may contain cross-equation restrictions on parameters.

Syntax

System Method: system_name.sur(options)

Options

i Iterate on the weighting matrix and coefficient vector simultaneously.

s Iterate on the weighting matrix and coefficient vector sequentially.

o (default) Iterate only on the coefficient vector with one step of the weighting matrix.

c One step iteration on the coefficient vector after one step of the weighting matrix.

m = integer Maximum number of iterations.

c = number Set convergence criterion.

l = number Set maximum number of iterations on the first-stage iteration to get one-step weighting matrix.

showopts / -showopts [Do / do not] display the starting coefficient values and estimation options in the estimation output.

deriv = keyword Set derivative methods. The argument keyword should be a one- or two-letter string. The first letter should either be “f” or “a” corresponding to fast or accurate numeric derivatives (if used). The second letter should be either “n” (always use numeric) or “a” (use analytic if possible). If omitted, EViews will use the global defaults.

p Print estimation results.
Examples

sys1.sur(i)
estimates SYS1 by SUR, iterating simultaneously on the weighting matrix and coefficient vector.

nlsys.sur(d,m=500)
estimates NLSYS by SUR with up to 500 iterations. The “d” option displays the starting values.

Cross-references

<table>
<thead>
<tr>
<th>svar</th>
<th>Var Proc</th>
</tr>
</thead>
</table>

Estimate factorization matrix for structural innovations.

Syntax

Var Proc: var_name.svar(options)
The var object must previously have been estimated in unrestricted form.

You must specify the identifying restrictions either in text form by the append proc or by a pattern matrix option. See “Specifying the Identifying Restrictions” on page 703 of the User's Guide for details on specifying restrictions.

Options
You must specify one of the following restriction type:

- **rtype = text**  
  Text form restrictions. The restrictions must be specified by the append command to use this option.

- **rtype = patsr**  
  Short-run pattern restrictions. You must provide the names of the patterned matrices by the “namea =” and “nameb =” options as described below.

- **rtype = patlr**  
  Long-run pattern restrictions. You must provide the name of the patterned matrix by the “namelr =” option as described below.
Other Options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namea = arg</td>
<td>Names of the pattern matrices for A and B matrices. Must be used with “rtype = patsr”.</td>
</tr>
<tr>
<td>nameb = arg</td>
<td>Name of the pattern matrix for long-run impulse responses. Must be used with “rtype = patlr”.</td>
</tr>
<tr>
<td>fsign</td>
<td>Do not apply the sign normalization rule. Default is to apply the sign normalization rule whenever applicable. See “Sign Indeterminacy” on page 708 of the User’s Guide for a discussion of the sign normalization rule.</td>
</tr>
<tr>
<td>f0 = arg</td>
<td>Starting values for the free parameters: “f0 = scalar” (specify fixed value for starting values), “f0 = s” (user specified starting values are taken from the C coefficient vector), “f0 = u” (draw starting values for free parameters from a uniform distribution on [0,1]), “f0 = n” (draw starting values for free parameters from standard normal).</td>
</tr>
<tr>
<td>maxiter = integer</td>
<td>Maximum number of iterations. Default is taken from global option setting.</td>
</tr>
<tr>
<td>conv = number</td>
<td>Convergence criterion. Default is taken from global option setting.</td>
</tr>
<tr>
<td>trace = integer</td>
<td>Trace iterations process every integer iterations (displays an untitled text object containing summary information).</td>
</tr>
<tr>
<td>nostop</td>
<td>Suppress “Near Singular Matrix” error message even if Hessian is singular at final parameter estimates.</td>
</tr>
</tbody>
</table>

Examples

```
var var1.ls 1 4 ml gdp cpi
matrix(3,3) pata
'fill matrix in row major order
pata.fill(by=r) 1,0,0, na,1,0, na,na,1
matrix(3,3) patb
pata.fill(by=r) na,0,0, 0,na,0, 0,0,na
var1.svar(rtype=patsr,namea=pata,nameb=patb)
```
The first line declares and estimates a VAR with three variables. Then we create the short-run pattern matrices and estimate the factorization matrix.

\[
\begin{align*}
\text{var var1.ls 1 8 dy u @} \\
\text{var1.append(svar) @lr1(@u1)=0} \\
\text{freeze(out1) var1.svar(rtype=text)}
\end{align*}
\]

The first line declares and estimates a VAR with two variables without a constant. The next two lines specify a long-run restriction in text form and stores the estimation output in a table object named OUT1.

Cross-references
See “Structural (Identified) VARs” on page 703 of the User’s Guide for a discussion of structural VARs.

<table>
<thead>
<tr>
<th>sym</th>
<th>Object Declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Declare a symmetric matrix object.</td>
</tr>
<tr>
<td></td>
<td>The \texttt{sym} command declares and optionally initializes a matrix object.</td>
</tr>
</tbody>
</table>

**Syntax**

Command: \texttt{sym(n) sym\_name}

Command: \texttt{sym(n) sym\_name=assignment}

\texttt{sym} takes an optional argument \texttt{n} specifying the row and column dimension of the matrix and is followed by the name you wish to give the matrix.

You may also include an assignment in the \texttt{sym} command. The symmetric matrix will be resized, if necessary. Once declared, symmetric matrices may be resized by repeating the \texttt{sym} command for a given matrix name.

You may use \texttt{sym} instead of \texttt{matrix} for working with symmetric matrices.

**Examples**

\texttt{sym mom}

declares a symmetric matrix named MOM with one zero element.

\texttt{sym y=@inner(x)}

declares a symmetric matrix Y and assigns to it the inner product of the matrix X.
Cross-references

See “Matrix Language” on page 23 of the Command and Programming Reference for a discussion of matrix objects in EViews.

See also matrix (p. 335).

system

Object Declaration

Declare system of equations.

The system command declares an object as a system for estimation by system methods such as three-stage least squares.

Syntax

Command: system system_name

Follow the system keyword by a name for the system. If you do not provide a name, EViews will open an untitled system object (if in interactive mode).

Examples

system mysys

creates a system named MYSYS.

Cross-references

Chapter 23 of the User’s Guide provides a full discussion of system objects.

See append (p. 186) for adding specification lines to an existing system. The system object is described in greater detail in beginning on page 166.

table

Object Declaration

Declare a table object.

The table command declares and optionally sizes a table object.

Syntax

Command: table(rows,cols) table_name

The table command takes two optional arguments specifying the row and column dimension of the table, and is followed by the name you wish to give the matrix. If no sizing information is provided, the table will be a single dimension.
You may also include an assignment in the `sym` command. The symmetric matrix will be resized, if necessary. Once declared, symmetric matrices may be resized by repeating the `sym` command for a given matrix name.

**Examples**

```
table onelement
```
declares a one element table

```
table(10,5) outtable
```
creates a table `OUTTABLE` with 10 rows and 5 columns.

**Cross-references**

Chapter 4 describes table formatting using commands. See Chapter 14 of the User’s Guide for a discussion and examples of table formatting in EViews.

See also `freeze` (p. 275).

<table>
<thead>
<tr>
<th>template</th>
<th>Graph Proc</th>
</tr>
</thead>
</table>

**Applies templates to graph objects.**

If you apply template to a multiple graph object, the template options will be applied to each graph in the multiple graph. If the template graph is a multiple graph, the options of the first graph will be used.

**Syntax**

```
Graph Proc: graph_name.template(options) graph_name
```

Follow the name of the graph you want to apply the template options with a period, the keyword `template`, and the name of the graph object to use as a template.

**Options**

```
t            Copy any text labels and shading in the template graph in addition to its option settings.
```

**Examples**

```
gra_cs.template gra_gdp
```
applies the option settings in the graph object `GRA_GDP` to the graph `GRA_CS`. Text and shadings in `GRA_GDP` will not be applied to `GRA_CS`.
g1.template(t) mygraph1

applies the option settings of MYGRAPH1 and all text and shadings in the graph to the graph G1. Text and shading objects include those added with the `addtext` (p. 183) or `draw` (p. 252) commands. Note that use the “t” option overwrites any existing text and shading objects in the target graph.

If you are using a boxplot as a template for another graph type, or vice versa, note that the graph types and boxplot specific attributes will not be changed. In addition, when the “t” option is used, vertical lines or shaded areas will not be copied between the graphs, since the horizontal scales differ.

**Cross-references**

See “Graph Templates” on page 394 of the *User’s Guide* for additional discussion.

| testadd | Command || Equation View |
|---------|-----------|----------------|

Test whether to add regressors to an estimated equation.

Tests the hypothesis that the listed variables were incorrectly omitted from an estimated equation (only available for equations estimated by list). The test displays some combination of Wald and LR test statistics, as well as the auxiliary regression.

**Syntax**

Command: `testadd arg1 [arg2 arg3 ...]`

Equation View: `eq_name.testadd arg1 [arg2 arg3 ...]`

List the names of the series or groups of series to test for omission after the keyword. The command form applies the test to the default equation.

**Options**

- `p
  Print output from the test.`

**Examples**

```
ls sales c adver lsales ar(1)
testadd gdp gdp(-1)
```

tests whether GDP and GDP(-1) belong in the specification for SALES. The commands

```
equation oldeq.ls sales c adver lsales ar(1)
oldeq.testadd gdp gdp(-1)
```

perform the same test using a named equation object.
Cross-references


See also `testdrop` (p. 464) and `wald` (p. 490).

testbtw Group View

Test equality of the mean, median or variance between (among) series in a group.

Syntax

Group View: `group_name.testbtw(options)`

Specify the type of test as an option.

Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean (default)</td>
<td>Test equality of mean.</td>
</tr>
<tr>
<td>med</td>
<td>Test equality of median.</td>
</tr>
<tr>
<td>var</td>
<td>Test equality of variance.</td>
</tr>
<tr>
<td>c</td>
<td>Use common sample.</td>
</tr>
<tr>
<td>i (default)</td>
<td>Use individual sample.</td>
</tr>
<tr>
<td>p</td>
<td>Print the test results.</td>
</tr>
</tbody>
</table>

Examples

```
group g1 wage_m wage_f
g1.testbtw
```

tests the equality of means between the two series WAGE_M and WAGE_F.

Cross-references

See “Tests of Equality” on page 355 of the User's Guide for further discussion of these tests.

See also `testby` (p. 462), `teststat` (p. 468).

testby Series View

Test equality of the mean, median, or variance of a series across categories classified by a list of series or a group.
Syntax

Series View:  series_name.testby(options) arg1 [arg2 arg2 …]

Follow the testby keyword by a list of the names of the series or groups to use as classifiers. Specify the type of test as an option.

Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean (default)</td>
<td>Test equality of mean.</td>
</tr>
<tr>
<td>med</td>
<td>Test equality of median.</td>
</tr>
<tr>
<td>var</td>
<td>Test equality of variance.</td>
</tr>
<tr>
<td>dropna (default), keepna</td>
<td>[Drop /Keep] NAs as a classification category.</td>
</tr>
<tr>
<td>v = integer (default = 100)</td>
<td>Bin categories if classification series take more than the specified number of distinct values.</td>
</tr>
<tr>
<td>nov</td>
<td>Do not bin based on the number of values of the classification series.</td>
</tr>
<tr>
<td>a = number (default = 2)</td>
<td>Bin categories if average cell count is less than the specified number.</td>
</tr>
<tr>
<td>noa</td>
<td>Do not bin on the basis of average cell count.</td>
</tr>
<tr>
<td>b = integer (default = 5)</td>
<td>Set maximum number of binned categories.</td>
</tr>
<tr>
<td>p</td>
<td>Print the test results.</td>
</tr>
</tbody>
</table>

Examples

wage.testby(med) race

Tests equality of medians of WAGE across groups classified by RACE.

Cross-references


See also testbtw (p. 462), teststat (p. 468).
Test whether to drop regressors from a regression.

Tests the hypothesis that the listed variables were incorrectly included in the estimated equation (only available for equations estimated by list). The test displays some combination of $F$ and LR test statistics, as well as the test regression.

Syntax

Command: `testdrop arg1 [arg2 arg3 ...]

Equation View: `eq_name.testdrop arg1 [arg2 arg3 ...]

List the names of the series or groups of series to test for omission after the keyword. The command form applies the test to the default equation.

Options

`p` Print output from the test.

Examples

\begin{verbatim}
ls sales c adver lsales ar(1)
testdrop adver
\end{verbatim}

tests whether ADVER should be excluded from the specification for SALES. The commands

\begin{verbatim}
equation oldq.ls sales c adver lsales ar(1)
oldq.testdrop adver
\end{verbatim}

perform the same test using a named equation object.

Cross-references

See “Coefficient Tests” on page 538 of the User’s Guide for further discussion of testing coefficients.

See also `testadd` (p. 461) and `wald` (p. 490).
Perform exogeneity (Granger causality) tests on a VAR object.

Syntax

Var View: var_name.testexog(options)

Options

- name = arg: Save the Wald test statistics in named matrix object. See below for a description of the statistics stored in the matrix.
- p: Print output from the test.

The name= option stores a \( (k + 1) \times k \) matrix, where \( k \) is the number of endogenous variables in the VAR. For the first \( k \) rows, the \( i \)-th row, \( j \)-th column entry is the Wald statistic for the joint significance of lags of the \( i \)-th endogenous variable in the \( j \)-th equation. (Note that the entries in the main diagonal are not reported in the table view). The degrees of freedom of the Wald statistics for the first \( k \) rows is the number of lags you included in the VAR.

The \( j \)-th column of the last row contains the Wald statistic for the joint significance of all lagged endogenous variables (excluding lags of the dependent variable) in the \( j \)-th equation. The degrees of freedom of the Wald statistics in the last row is \( (k - 1) \times k \) times the number of lags you included in the VAR.

Examples

```
var var1.ls 1 6 lgdp lml lcpi
freeze(tab1) var1.testexog(name=exog)
```

The first line declares and estimates a VAR. The second line stores the exclusion test results in a named table TAB1 and stores the Wald statistics in a matrix named EXOG.

Cross-references

See “Diagnostic Views” on page 694 of the User’s Guide for a discussion other VAR diagnostics.

See also testlags (p. 467).
testfit

Syntax

Equation View: binary_equation.testfit(options)

Options

- **h**
  - Group by the predicted values of the estimated equation.

- **s = series name**
  - Group by the specified series.

- **integer**
  - Specify the number of quantile groups in which to classify observations.
  - *(default = 10)*

- **u**
  - Unbalanced grouping. Default is to randomize ties to balance the number of observations in each group.

- **v**
  - Group according to the values of the reference series.

- **l = integer**
  - Limit the number of values to use for grouping.
  - *(default = 100)*
  - Should be used with the “v” option.

- **p**
  - Print the result of the test.

Examples

```plaintext
  equation eq1.binary work c age edu
  eq1.testfit(h,5,u)
```

estimates a probit specification, and tests goodness-of-fit by comparing five unbalanced groups of actual data to those estimated by the model.

Cross-references

testlags — Perform lag exclusion (Wald) tests on a VAR.

Syntax

Var View: var_name.testlags(options)

Options

- name = arg
  - Save the Wald test statistics in named matrix object. See below for a description of the statistics contained in the stored matrix.
- p
  - Print the result of the test.

The “name =” option stores an $m \times (k + 1)$ matrix, where $m$ is the number of lagged terms and $k$ is the number of endogenous variables in the VAR. For the first $k$ columns, the $i$-th row, $j$-th column entry is the Wald statistic for the joint significance of all $i$-th lagged endogenous variables in the $j$-th equation. These Wald statistics have a $\chi^2$ distribution with $k$ degrees of freedom under the exclusion null.

The $i$-th row of the last column contains the system Wald statistic for testing the joint significance of all $i$-th lagged endogenous variables in the VAR system. The system Wald statistics has a chi-square distribution with $k^2$ degrees of freedom under the exclusion null.

Examples

```
var var1.ls 1 6 lgdp lml lcpi
freeze(tab1) var1.testlags(name=lags)
```

The first line declares and estimates a VAR. The second line stores the lag exclusion test results in a table named TAB1 and stores the Wald statistics in a matrix named LAGS.

Cross-references

See “Diagnostic Views” on page 694 of the User’s Guide for a discussion other VAR diagnostics.

See also laglen (p. 303) and testexog (p. 465).
teststat  Series View

Test simple hypotheses of whether the mean, median, or variance of a series is equal to a specified value.

Syntax

Series View:  series_name.teststat(options)

Specify the type of test and the value under the null hypothesis as an option.

Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean = number</td>
<td>Test the null hypothesis that the mean equals the specified number.</td>
</tr>
<tr>
<td>med = number</td>
<td>Test the null hypothesis that the median equals the specified number.</td>
</tr>
<tr>
<td>var = number</td>
<td>Test the null hypothesis that the variance equals the specified number. The number must be positive.</td>
</tr>
<tr>
<td>std = number</td>
<td>Test equality of mean conditional on the specified standard deviation. The standard deviation must be positive.</td>
</tr>
<tr>
<td>p</td>
<td>Print the test results.</td>
</tr>
</tbody>
</table>

Examples

```
smpl if race=1
lwage.teststat(var=4)
```

tests the null hypothesis that the variance of LWAGE is equal to 4 for the subsample with RACE = 1.

Cross-references


See also testbtw (p. 462), testby (p. 462).
text

Declare a text object when used as a command, or display text representation of the model specification or text object.

Syntax

Command: \text{text} \ object\_name

Model View: model\_name.text(options)

Text View: text\_name.text(options)

Follow the keyword with a name of the text object. When used as a model view, text is equivalent to \text{spec} (p. 441).

Options

p \hspace{1cm} \text{Print the model text specification.}

Examples

\text{text} \ notes1

declares a text object named NOTES1.

Cross-references


See also \text{spec} (p. 441).

tic

Reset the timer.

Syntax

Command: tic

Examples

The sequence of commands

\begin{verbatim}
tic
[\text{some commands}]
\end{verbatim}
toc
resets the timer, executes commands, and then displays the elapsed time in the statusline. Alternatively,

\[
\text{tic} \\
[\text{some commands}] \\
!\text{elapsed} = @\text{toc}
\]
resets the time, executes commands, and saves the elapsed time in the control variable !ELAPSED.

Cross-references
See also toc (p. 470) and @toc (p. 570).

<table>
<thead>
<tr>
<th>toca</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Display elapsed time (since timer reset) in seconds.</td>
</tr>
</tbody>
</table>

Syntax

Command: toca

Examples

The sequence of commands

\[
\text{tic} \\
[\text{some commands}] \\
\text{toc}
\]
resets the timer, executes commands, and then displays the elapsed time in the statusline, while the set of commands

\[
\text{tic} \\
[\text{some commands}] \\
!\text{elapsed} = @\text{toc} \\
[\text{more commands}] \\
\text{toc}
\]
resets the time, executes commands, saves the elapsed time in the control variable !ELAPSED, executes additional commands, then displays the total elapsed time in the statusline.
Cross-references
See also tic (p. 469) and @toc (p. 570).

### trace

<table>
<thead>
<tr>
<th>Model View</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display trace view of a model showing iteration history for selected solved variables.</td>
</tr>
<tr>
<td>Syntax</td>
</tr>
<tr>
<td>Model View: model_name.trace(options)</td>
</tr>
<tr>
<td>Options</td>
</tr>
<tr>
<td>p</td>
</tr>
</tbody>
</table>

Cross-references
See also msg (p. 339), solve (p. 438) and solveopt (p. 439).

### tramoseats

<table>
<thead>
<tr>
<th>Series Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run the external seasonal adjustment program Tramo/Seats using the data in the series.</td>
</tr>
<tr>
<td>Syntax</td>
</tr>
<tr>
<td>Series Proc: series_name.tramoseats(options) [base_name]</td>
</tr>
</tbody>
</table>

tractor seats is available for annual, semi-annual, quarterly, and monthly series. The procedure requires at least \( n \) observations and can adjust up to 600 observations where

\[
n = \begin{cases} 
36 & \text{for monthly data} \\
\max\{12, 4s\} & \text{for other seasonal data}
\end{cases}
\]

(B.2)

Enter the name of the original series followed by a dot, the keyword, and optionally provide a base name (no more than 20 characters long) to name the saved series. The default base name is the original series name. The saved series will have postfixes appended to the base name.
Options

`runtype = arg`  Tramo/Seats Run Specification: “ts” (run Tramo followed by Seats; the “opt = ” options are passed to Tramo, and Seats is run with the input file returned from Tramo), “t” (run only Tramo), “s” (run only Seats).

`(default = “ts”)`

`save = arg`  Specify series to save in workfile: you must use one or more from the following key word list: “hat” (forecasts of original series), “lin” (linearized series from Tramo), “pol” (interpolated series from Tramo), “sa” (seasonally adjusted series from Seats), “trd” (final trend component from Seats), “ir” (final irregular component from Seats), “sf” (final seasonal factor from Seats), “cyc” (final cyclical component from Seats).

To save more than one series, separate the list of key words with a space. Do not use commas within the list of save series.

The special key word “save = *” will save all series in the key word list. The five key words “sa”, “trd”, “ir”, “sf”, “cyc” will be ignored if “runtype = t”.

`opt = arg`  A space delimited list of input namelist. Do not use commas within the list. The syntax for the input namelist is explained in the pdf documentation file. See also “Notes” below.

`reg = arg`  A space delimited list for one line of reg namelist. Do not use commas within the list. This option must be used in pairs, either with another “reg=” option or “regname=” option. The reg namelist is available only for Tramo and its syntax is explained in the pdf documentation file. See also “Notes” below.

`regname = arg`  Name of a series or group in the current workfile that contains the exogenous regressors specified in the previous “reg=” option. See “Notes” below.

`p`  Print the results of the Tramo/Seats procedure.
Notes

The command line interface to Tramo/Seats does very little error checking of the command syntax. EViews simply passes on the options you provide “as is” to Tramo/Seats. If the syntax contains an error, you will most likely see the EViews error message “output file not found”. If you see this error message, check the input files produced by EViews for possible syntax errors as described in “Trouble Shooting” on page 324 of the User’s Guide.

Additionally, here are some of the more commonly encountered syntax errors.

- To replicate the dialog options from the command line, use the following input options in the “opt = ” list. See the pdf documentation file for a description of each option.
  1. data frequency: “mq = ”.
  2. forecast horizon: “npred = ” for Tramo and “fh = ” for Seats.
  3. transformation: “lam = ”.
  4. ARIMA order search: “inic = ” and “idif = ”.
  5. Easter adjustment: “ieast = ”.
  6. trading day adjustment: “itrad = ”.
  7. outlier detection: “iatip = ” and “aio = ”.

- The command option input string list must be space delimited. Do not use commas. Options containing an equals sign should not contain any spaces around the equals; the space will be interpreted as a delimiter by Tramo/Seats.

- If you set “rtype = ts”, you are responsible for supplying either “seats = 1” or “seats = 2” in the “opt = ” option list. EViews will issue the error message “Seats.itr not found” if the option is omitted. Note that the dialog option Run Seats after Tramo sets “seats = 2”.

- Each “reg = ” or “regname = ” option is passed to the input file as a separate line in the order that they appear in the option argument list. Note that these options must come in pairs. A “reg = ” option must be followed by another “reg = ” option that specifies the outlier or intervention series or by a “regname = ” option that provides the name for an exogenous series or group in the current workfile. See the sample programs in the EXAMPLE FILES directory.

- If you specify exogenous regressors with the “reg = ” option, you must set the appropriate “ireg = ” option (for the total number of exogenous series) in the “opt = ” list.

- To use the “regname = ” option, the preceding “reg = ” list must contain the “user = -1” option and the appropriate “ilong = ” option. Do not use “user = 1” since EViews will always write data in a separate external file. The “ilong = ” option must be at
least the number of observations in the current workfile sample plus the number of forecasts. The exogenous series should not contain any missing values in this range. Note that Tramo may increase the forecast horizon, in which case the exogenous series is extended by appending zeros at the end.

Examples

freeze(tab1) show x.tramoseats(runtype=t, opt="lam=-1 iatip=1 aio=2 va=3.3 noadmiss=1 seats=2", save=*) x
replicates the example file EXAMPLE.1 in Tramo. The output file from Tramo is stored in a text object named tab1. This command returns three series named X_HAT, X_LIN, X_POL.

show x.TramoSeats(runtype=t, opt="NPRED=36 LAM=1 IREG=3 INTERP=2 IMEAN=0 P=1 Q=0 D=0", reg="ISEQ=1 DELTA=1.0", reg="61 1", reg="ISEQ=8 DELTAS=1.0", reg="138 5 150 5 162 5 174 5 186 5 198 5 210 5 222 5", reg="ISEQ=8 DELTAS=1.0", reg="143 7 155 7 167 7 179 7 191 7 203 7 215 7 227 7") x
replicates the example file EXAMPLE.2 in Tramo. This command produces an input file that looks as follows:

$INPUT NPRED=36 LAM=1 IREG=3 INTERP=2 IMEAN=0 P=1 Q=0 D=0, $
$REG ISEQ=1 DELTA=1.0$
61 1
$REG ISEQ=8 DELTAS=1.0$
138 5 150 5 162 5 174 5 186 5 198 5 210 5 222 5
$REG ISEQ=8 DELTAS=1.0$
143 7 155 7 167 7 179 7 191 7 203 7 215 7 227 7

Further examples that replicate many of the example files provided by Tramo/Seats can be found in the EXAMPLE FILES directory. You will also find example files that compare seasonal adjustments from Census X12 and Tramo/Seats.

Cross-references

See also the Tramo/Seats documentation that accompanied your EViews distribution.

See also seas (p. 403) and x12 (p. 504).

tsls

<table>
<thead>
<tr>
<th>Command</th>
<th>Equation Method</th>
<th>Pool Method</th>
<th>System Method</th>
</tr>
</thead>
</table>

Two-stage least squares.
Carries out estimation for equations, pools, or systems using two-stage least squares.
**Syntax**

**Command:**

\[
\text{tsls(options) } y \ x1 \ [x2 \ x3 \ ...] \ @ \ z1 \ [z2 \ z3 \ ...]
\]

\[
\text{tsls(options) specification } @ \ z1 \ [z2 \ z3 \ ...]
\]

**Equation Method:**

\[
\text{eq\_name.tls(options) } y \ x1 \ [x2 \ x3 \ ...] \ @ \ z1 \ [z2 \ z3 \ ...]
\]

\[
\text{eq\_name.tls(options) specification } @ \ z1 \ [z2 \ z3 \ ...]
\]

**Pool Method:**

\[
\text{pool\_name.tls(options) } y \ x1 \ [x2 \ x3 \ ...] \ [\text{xreg} \ w1 \ w2 \ ...] \ [\text{per-reg} \ w3 \ w4 \ ...] \ [\text{inst} \ z1 \ z2 \ ...] \ [\text{cxinst} \ z3 \ z4 \ ...] \ [\text{perinst} \ z5 \ z6 \ ...]
\]

**System Method:**

\[
\text{system\_name.tls(options)}
\]

To use `tsls` as a command or equation method, list the dependent variable first, followed by the regressors, then any AR or MA error specifications, then an “@”-sign, and finally, a list of exogenous instruments. There must be at least as many instrumental variables as there are independent variables. All exogenous variables included in the regressor list should also be included in the instrument list. A constant is included in the list of instrumental variables even if not explicitly specified. You can estimate nonlinear equations or equations specified with formulas; always list the instrumental variables after an “@”-sign.

**Options**

**General options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>m = integer</code></td>
<td>Set maximum number of iterations.</td>
</tr>
<tr>
<td><code>c = number</code></td>
<td>Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.</td>
</tr>
<tr>
<td><code>deriv = keyword</code></td>
<td>Set derivative methods. The argument <code>keyword</code> should be a one- or two-letter string. The first letter should either be “f” or “a” corresponding to fast or accurate numeric derivatives (if used). The second letter should be either “n” (always use numeric) or “a” (use analytic if possible). If omitted, EViews will use the global defaults.</td>
</tr>
<tr>
<td><code>showopts / -showopts</code></td>
<td>[Do / do not] display the starting coefficient values and estimation options in the estimation output.</td>
</tr>
<tr>
<td><code>p</code></td>
<td>Print estimation results.</td>
</tr>
</tbody>
</table>
### Additional Options for Non-Panel Equation estimation

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>w = series_name</strong></td>
<td>Weighted TSLS. Each observation will be weighted by multiplying by the specified series.</td>
</tr>
<tr>
<td><strong>h</strong></td>
<td>White’s heteroskedasticity consistent standard errors.</td>
</tr>
<tr>
<td><strong>n</strong></td>
<td>Newey-West heteroskedasticity and autocorrelation consistent (HAC) standard errors.</td>
</tr>
<tr>
<td><strong>s</strong></td>
<td>Use the current coefficient values in “C” as starting values for equations with AR or MA terms (see also <code>param</code> (p. 368)).</td>
</tr>
<tr>
<td><strong>s = number</strong></td>
<td>Determine starting values for equations specified by list with AR or MA terms. Specify a number between zero and one representing the fraction of preliminary least squares estimates computed without AR or MA terms. Note that out of range values are set to “s = 1”. Specifying “s = 0” initializes coefficients to zero. By default EViews uses “s = 1”.</td>
</tr>
<tr>
<td><strong>z</strong></td>
<td>Turn off backcasting in ARMA models.</td>
</tr>
</tbody>
</table>

### Additional Options for Pool and Panel Equation estimation

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cx = arg</strong></td>
<td>Cross-section effects. For fixed effects estimation, use “cx = f”; for random effects estimation, use “cx = r”.</td>
</tr>
<tr>
<td><strong>per = arg</strong></td>
<td>Period effects. For fixed effects estimation, use “cx = f”; for random effects estimation, use “cx = r”.</td>
</tr>
<tr>
<td><strong>wgt = arg</strong></td>
<td>GLS weighting: (default) none, cross-section system weights (“wgt = cxsur”), period system weights (“wgt = persur”), cross-section diagonal weights (“wgt = cxdiag”), period diagonal weights (“wgt = perdiag”).</td>
</tr>
</tbody>
</table>
### \( \text{cov} = \text{arg} \)
Coefficient covariance method: (default) ordinary, White cross-section system robust ("cov = cxwhite"), White period system robust ("cov = perwhite"), White heteroskedasticity robust ("cov = stackedwhite"), Cross-section system robust/PCSE ("cov = cxsur"), Period system robust/PCSE ("cov = persur"), Cross-section heteroskedasticity robust/PCSE ("cov = cxdiag"), Period heteroskedasticity robust ("cov = perdiag").

### keepwghts
Keep full set of GLS weights used in estimation with object, if applicable (by default, only small memory weights are saved).

### rancalc = arg (default = "sa")
Random component method: Swamy-Arora ("rancalc = sa"), Wansbeek-Kapteyn ("rancalc = wk"), Wallace-Hussain ("rancalc = wh").

### nodf
Do not perform degree-of-freedom corrections in computing coefficient covariance matrix. The default is to use degree-of-freedom corrections.

### coef = arg
Specify the name of the coefficient vector (if specified by list); the default is to use the "C" coefficient vector.

### iter = arg (default = "onec")
Iteration control for GLS specifications: perform one weight iteration, then iterate coefficients to convergence ("iter = onec"), iterate weights and coefficients simultaneously to convergence ("iter = sim"), iterate weights and coefficients sequentially to convergence ("iter = seq"), perform one weight iteration, then one coefficient step ("iter = oneb").

Note that random effects models currently do not permit weight iteration to convergence.

### s
Use the current coefficient values in “C” as starting values for equations with AR or MA terms (see also \( \text{param} \) (p. 368)).
### Additional options for systems

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| `s = number` | Determine starting values for equations specified by list with AR terms. Specify a number between zero and one representing the fraction of preliminary least squares estimates computed without AR terms. Note that out of range values are set to “s = 1”. Specifying “s = 0” initializes coefficients to zero. By default EViews uses “s = 1”.
| `i` | Iterate on the weighting matrix and coefficient vector simultaneously. |
| `s` | Iterate on the weighting matrix and coefficient vector sequentially. |
| `o (default)` | Iterate only on the coefficient vector with one step of the weighting matrix. |
| `c` | One step iteration of the coefficient vector after one step of the weighting matrix. |
| `l = number` | Set maximum number of iterations on the first-stage iteration to get one-step weighting matrix. |

### Examples

```
eq1.tsls y_d c cpi inc ar(1) @ lw(-1 to -3)
estimates EQ1 using TSLS regression of Y_D on a constant, CPI, INC with AR(1) using a constant, LW(-1), LW(-2), LW(-3) as instruments.
param c(1) .1 c(2) .1
eq1.tsls(s,m=500) y_d=c(1)+inc^C(2) @ cpi
```
estimates a nonlinear TSLS model using a constant and CPI as instruments. The first line sets the starting values for the nonlinear iteration algorithm.

```
sys1.tsls
```
estimates the system object using TSLS.

### Cross-references

See “Two-stage Least Squares” on page 441 and page 667 of the User’s Guide for details on two-stage least squares estimation in single equations and systems.

See also `ls` (p. 315). For estimation of weighted TSLS in systems, see `wtsls` (p. 501).
unlink — 479

<table>
<thead>
<tr>
<th>Command</th>
<th>Model Proc</th>
</tr>
</thead>
<tbody>
<tr>
<td>unlink</td>
<td></td>
</tr>
</tbody>
</table>

Break links in series objects or models.

Syntax

Command:  
unlink  

Model Proc:  
object.unlink  

When used as a command, unlink converts link objects to ordinary series or alphas. Follow the keyword with a list of names of links to be converted to ordinary series (values). The list of links may include wildcard characters.

Used as a model proc, unlink breaks equation links in the model. Follow the name of the model object by a period, the keyword, and a specification for the variables to unlink.

The  spec  may contain either a list of the endogenous variables to be unlinked, or the keyword “@ALL”, instructing EViews to unlink all equations in the model.

Note: if a link is to another model or a system object, then more than one endogenous variable may be associated with the link. If the  spec  contains any of the endogenous variables in a linked model or system, EViews will break the link for all of the variables found in the link.

Examples

The command

```plaintext
unlink gdp income
```

converts the link series GDP and INCOME to ordinary series.

```plaintext
unlink *
```

breaks all links in the current workfile page.

Used as a model proc, the expressions

```plaintext
mod1.unlink @all
mod2.unlink z1 z2
```

unlink all of equations in MOD1, and all of the variables associated with the links for Z1 and Z2 in MOD2.
Cross-references

See Chapter 8, “Series Links”, on page 159 for a detailed description of link objects. See also \texttt{link (p. 309)} and \texttt{linkto (p. 310)}.

See Chapter 26, “Models”, on page 749 of the \textit{User’s Guide} for a discussion of specifying and solving models in EViews. See also \texttt{append (p. 186)}, \texttt{merge (p. 336)} and \texttt{solve (p. 438)}.

\begin{center}
\begin{tabular}{|l|l|}
\hline
\textbf{update} & \textbf{Model Proc} \\
\hline
\end{tabular}
\end{center}

Update model specification.
Rec compiles the model and updates all links.

\textbf{Syntax}

\begin{verbatim}
Model Proc: model.update
\end{verbatim}

\begin{verbatim}
Follow the name of the model object by a period and the keyword update. Examples
mod1.update
\end{verbatim}

recompiles and updates all of the links in MOD1.

Cross-references

See Chapter 26, “Models”, on page 749 of the \textit{User’s Guide} for a discussion of specifying and solving models in EViews. See also \texttt{append (p. 186)}, \texttt{merge (p. 336)} and \texttt{solve (p. 438)}.

\begin{center}
\begin{tabular}{|l|l|}
\hline
\textbf{updatecoefs} & \textbf{Equation Proc | Logl Proc | Pool Proc | Sspace Proc | System Proc} \\
\hline
\end{tabular}
\end{center}

Update coefficient object values from estimation object.
Copies coefficients from the estimation object into the appropriate coefficient vector or vectors.

\textbf{Syntax}

\begin{verbatim}
Object Proc: object.updatecoef
\end{verbatim}

\begin{verbatim}
Follow the name of the estimation object by a period and the keyword updatecoef.
\end{verbatim}

\textbf{Examples}

\begin{verbatim}
equation eql.ls y c x1 x2 x3
\end{verbatim}
equation eq2.ls z c z1 z2 z3
eq1.updatecoef
places the coefficients from EQ1 in the default coefficient vector C.
coef(3) a
equation eq3.ls y=a(1)+z1^c(1)+log(z2+a(2))+exp(c(4)+z3/a(3))
equation eq2.ls z c z1 z2 z3
eq3.updatecoef
updates the coefficient vector A and the default vector C so that both contain the coefficients from EQ3.

Cross-references
See also coef (p. 223).

<table>
<thead>
<tr>
<th>Command</th>
<th>Group View</th>
<th>Pool View</th>
<th>Series View</th>
</tr>
</thead>
<tbody>
<tr>
<td>uroot</td>
<td>Command</td>
<td>Group View</td>
<td>Pool View</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Series View</td>
</tr>
</tbody>
</table>

Carries out unit root tests on a single series, pool series, group of series, or panel structured series.

If used on a series in a panel structured workfile, or with a pool series or group of series, the procedure will perform panel unit root testing.

The non-panel unit root tests include Augmented Dickey-Fuller (ADF), GLS detrended Dickey-Fuller (DFGLS), Phillips-Perron (PP), Kwiatkowski, et. al. (KPSS), Elliot, Rothenberg, and Stock (ERS) Point Optimal, or Ng and Perron (NP) tests for a unit root in the series (or its first or second difference).

The panel unit root tests include Levin, Lin and Chu (LLC), Breitung, Im, Pesaran, and Shin (IPS), Fisher - ADF, Fisher - PP, and Hadri tests on levels, or first or second differences.

Syntax
Command: uroot(options) series_name
Group View: group_name.uroot(options)
Pool View: pool_name.uroot(options) pool_series
Series View: series_name.uroot(options)

Tests on a single series may be carried out in two ways: as a command, or as a procedure off of the series. Accordingly, you may enter the keyword followed by the individual series name, or you may enter the series name followed by a period and the keyword.
For other cases, you should enter the object name followed by a period, and the keyword, and, in the case of pool testing, the name of a pool “?” series.

**Options**

**Basic Specification Options**

You should specify the exogenous variables and order of dependent variable differencing in the test equation using the following options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>const</code></td>
<td>Include a constant in the test equation.</td>
</tr>
<tr>
<td><code>trend</code></td>
<td>Include a constant and a linear time trend in the test equation.</td>
</tr>
<tr>
<td><code>none</code></td>
<td>Do not include a constant or time trend (only available for the ADF and PP tests).</td>
</tr>
<tr>
<td><code>dif=integer</code></td>
<td>Order of differencing of the series prior to running the test. Valid values are {0, 1, 2}.</td>
</tr>
</tbody>
</table>

For ordinary (non-panel) unit root tests, you should specify the test type using one of the following keywords:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>adf</code></td>
<td>Augmented Dickey-Fuller.</td>
</tr>
<tr>
<td><code>dfgls</code></td>
<td>GLS detrended Dickey-Fuller (Elliot, Rothenberg, and Stock).</td>
</tr>
<tr>
<td><code>pp</code></td>
<td>Phillips-Perron.</td>
</tr>
<tr>
<td><code>ers</code></td>
<td>Elliot, Rothenberg, and Stock (Point Optimal).</td>
</tr>
<tr>
<td><code>np</code></td>
<td>Ng and Perron.</td>
</tr>
</tbody>
</table>

For panel testing, you may use one of the following keywords to specify the test:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sum</code></td>
<td>Summary of all of the panel unit root tests.</td>
</tr>
<tr>
<td><code>llc</code></td>
<td>Levin, Lin, and Chu.</td>
</tr>
<tr>
<td><code>breit</code></td>
<td>Breitung.</td>
</tr>
<tr>
<td><code>ips</code></td>
<td>Im, Pesaran, and Shin.</td>
</tr>
<tr>
<td><code>adf</code></td>
<td>Fisher - ADF.</td>
</tr>
<tr>
<td><code>pp</code></td>
<td>Fisher - PP.</td>
</tr>
<tr>
<td><code>hadri</code></td>
<td>Hadri.</td>
</tr>
</tbody>
</table>
For backward compatibility, the shortened forms for these latter options, “c”, “t”, and “n”, are presently supported. However for future compatibility we recommend that you use the longer forms.

*Options for ordinary (non-panel) unit root tests*

In addition, the following panel specific options are available:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Applicable Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>band = arg, b=arg (default = “nw”)</td>
<td>Method of selecting the bandwidth: “nw” (Newey-West automatic variable bandwidth selection), “a” (Andrews automatic selection), “number” (user specified bandwidth).</td>
<td>PP, KPSS, ERS, NP</td>
</tr>
<tr>
<td>lag=arg (default = “a”)</td>
<td>Method of selecting lag length (number of first difference terms) to be included in the regression: “a” (automatic information criterion based selection), or “integer” (user-specified lag length).</td>
<td>ADF, DFGLS, ERS, PP, KPSS, ERS, NP tests</td>
</tr>
</tbody>
</table>
The following panel specific options are available:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| `maxlag = integer` | Maximum lag length to consider when performing automatic lag length selection: $\text{default} = \text{int}(\frac{12T}{100})^{0.25}$

Applicable to ADF and DFGLS tests, and for other tests when using AR spectral density estimators (where “hac =” is one of {ar, ardt, argls}). |
| `balance` | Use balanced (across cross-sections or series) data when performing test.                                                                 |

Applicable to “Summary”, LLC, Fisher-PP, and Hadri testing. |
| `band = arg, b = arg (default = “nw”)` | Method of selecting the bandwidth: “nw” (Newey-West automatic variable bandwidth selection), “a” (Andrews automatic selection), `number` (user-specified common bandwidth), `vector_name` (user-specified individual bandwidth).

Applicable to “Summary”, LLC, Fisher-PP, and Hadri testing. |
**lag = arg**

Method of selecting lag length (number of first difference terms) to be included in the regression: "a" (automatic information criterion based selection), integer (user-specified common lag length), vector_name (user-specific individual lag length).

If the “balance” option is used,

\[
\text{default} = \begin{cases} 
1 & \text{if } (T_{\text{min}} \leq 60) \\
2 & \text{if } (60 < T_{\text{min}} \leq 100) \\
4 & \text{if } (T_{\text{min}} > 100)
\end{cases}
\]

where \(T_{\text{min}}\) is the length of the shortest cross-section or series, otherwise \(\text{default} = “a”\). Applicable to “Summary”, LLC, Breitung, IPS, and Fisher-ADF testing.

**info = arg**


Applicable to “Summary”, LLC, Breitung, IPS, and Fisher-ADF testing.

**maxlag = arg**

Maximum lag length to consider when performing automatic lag length selection, where arg is an integer (common maximum lag length) or a vector_name (individual maximum lag length)

\[
\text{default} = \text{int}(\min(12, T_i / 3) \cdot (T_i / 100)^{0.25})
\]

where \(T_i\) is the length of the cross-section or series.

Other options

**p**

Print output from the test.

**Examples**

The command
gnp.uroot(adf,const,lag=3,save=mout)

performs an ADF test on the series GDP with the test equation including a constant term and three lagged first-difference terms. Intermediate results are stored in the matrix MOUT.

ip.uroot(dfngls,trend,info=sic)

runs the DFGLS unit root test on the series IP with a constant and a trend. The number of lagged difference terms is selected automatically using the Schwarz criterion.

unemp.uroot(kpss,const,hac=pr,b=2.3)

runs the KPSS test on the series UNEMP The null hypothesis is that the series is stationary around a constant mean. The frequency zero spectrum is estimated using kernel methods (with a Parzen kernel), and a bandwidth of 2.3.

sp500.uroot(np,hac=ardt,info=maic)

runs the NP test on the series SP500. The frequency zero spectrum is estimated using the OLS AR spectral estimator with the lag length automatically selected using the modified AIC.

Cross-references

See “Unit Root Tests” on page 486 of the User’s Guide for discussion of standard unit root tests performed on a single series, and “Panel Unit Root Tests” on page 498 of the User’s Guide for discussion of unit roots tests performed on panel structured workfiles, groups of series, or pooled data.

<table>
<thead>
<tr>
<th>usage</th>
<th>Valmap View</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Find series and alphas which use the valmap.

Display list of series and alpha objects which use the valmap.

Syntax

Valmap View: valmap_name.stats(options)

Options

p Print the usage table.

Examples

map1.usage
displays a list of series and alphas which use the valmap MAP1.
Cross-references

For additional details, see “Value Maps” on page 145 of the User’s Guide. See also “Val-map” (p. 171).

See also map (p. 334).

### valmap

**Object Declaration**

Declare a value map object.

**Syntax**

Command:  `valmap valmap_name`

Follow the `valmap` keyword with a name for the object.

**Examples**

The commands

```
valmap mymap
mymap.append 3 three
mymap.append 99 "not in universe"
```

declare the valmap MYMAP and add two lines mapping the values 3 and 99 to the strings “three” and “not in universe”.

### var

**Object Declaration**

Declare a var (Vector Autoregression) object.

**Syntax**

Command:  `var var_name`

Command:  `var var_name.ls lag_pairs endog_list [@ exog_list]`

Command:  `var var_name.ec(trend, n) lag_pairs endog_list [@ exog_list]`

Declare the var as a name, or a name followed by an estimation method.
The `ls` method estimates an unrestricted VAR using equation-by-equation OLS. You must specify the order of the VAR (using one or more pairs of lag intervals), and then provide a list of series or groups to be used as endogenous variables. You may include exogenous variables such as trends and seasonal dummies in the VAR by including an "@-sign" followed by a list of series or groups.

See `ec` (p. 256) for the error correction specification of a VAR.

**Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p</code></td>
<td>Print the estimation result if the estimation procedure is specified.</td>
</tr>
</tbody>
</table>

**Examples**

```
var mvar.ls 1 4 8 8 m1 gdp tb3 @ @trend
```

declares and estimates an unrestricted VAR named MVAR with three endogenous variables (M1, GDP, TB3), five lagged terms (lags 1 through 4, and 8), and a linear trend.

```
var jvar.ec(c,2) 1 4 m1 gdp tb3
```

declares and estimates an error correction model named JVAR with three endogenous variables (M1, GDP, TB3), four lagged terms (lags 1 through 4), two cointegrating relations. The "c" option assumes a linear trend in data but only a constant in the cointegrating relations.

**Cross-references**


See `ec` (p. 256) for error correction models in VARs.

<table>
<thead>
<tr>
<th>vars</th>
<th>Model View</th>
</tr>
</thead>
</table>

**View of model organized by variable.**

Display the model in variable form with identification of endogenous, exogenous, and identity variables, as well as dependency tracking.

**Syntax**

```
Model View:    model_name.vars
```
Cross-references


See also block (p. 203), text (p. 469), and eqs (p. 260) for alternative representations of the model.

<table>
<thead>
<tr>
<th>vector</th>
<th>Object Declaration</th>
</tr>
</thead>
</table>

Declare a vector object.

The vector command declares and optionally initializes a (column) vector object.

Syntax

- Command: `vector(size) vector_name`
- Command: `vector(size) vector_name=assignment`

The keyword should be followed by the name you wish to give the vector. You may also provide an optional argument specifying the size of the vector. Once declared, vectors may be resized by repeating the command with a new size.

You may combine vector declaration and assignment. If there is no assignment statement, the vector will initially be filled with zeros.

Examples

```eviews
vector vec1
vector(20) vec2=nrnd
rowvector(10) row3=3
vector vec3=row3
```

VEC1 is declared as a column vector of size one with element 0. VEC2 is declared as a column vector of size 20 containing a simulated random draw from the standard normal distribution. Although declared as a column vector, VEC3 is reassigned as a row vector of size 10 with all elements equal to 3.

Cross-references


See also coef (p. 223) and rowvector (p. 390).
Wald coefficient restriction test.

The `wald` view carries out a Wald test of coefficient restrictions for an estimation object.

**Syntax**

Object View: `object_name.wald restrictions`

Enter the object name, followed by a period, and the keyword. This should be followed by a list of the coefficient restrictions. Joint (multiple) coefficient restrictions should be separated by commas.

**Options**

`p` Print the test results.

**Examples**

```
eq1.wald c(2)=0, c(3)=0
```

tests the null hypothesis that the second and third coefficients in equation EQ1 are jointly zero.

```
sys1.wald c(2)=c(3)*c(4)
```

tests the non-linear restriction that the second coefficient is equal to the product of the third and fourth coefficients in SYS1.

```
pool panel us uk jpn
panel.ls cons? inc? @ ar(1)
panel.wald c(3)=c(4)=c(5)
```

declares a pool object with three cross section members (US, UK, JPN), estimates a pooled OLS regression with separate AR(1) coefficients, and tests the null hypothesis that all AR(1) coefficients are equal.

**Cross-references**

See “Wald Test (Coefficient Restrictions)” on page 541 of the User’s Guide for a discussion of Wald tests.

See also `testdrop` (p. 464), `testadd` (p. 461).
Create a new workfile. The workfile becomes the active workfile.

**Syntax**

Command:  
```
wfcreate(options) frequency start_date end_date
[num_cross_sections]
```

Command:  
```
wfcreate(options) u num_observations
```

The first form of the command is used to create a new regular frequency workfile with the specified frequency, start and end date. If you include the optional `num_cross_sections`, EViews will create a balanced panel page using integer identifiers for each of the cross-sections. Note that more complex panel structures may be created using `pagestruct` (p. 363).

The second form of the command is used to create an unstructured workfile with the specified number of observations.

**Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wf=wf_name</code></td>
<td>Optional name for the new workfile.</td>
</tr>
<tr>
<td><code>page=page_name</code></td>
<td>Optional name for the page in the new workfile page. If not provided, EViews will use a name given by the page structure (e.g., “Undated”, “Annual”, “Daily5”).</td>
</tr>
</tbody>
</table>

**Examples**

```
wfcreate(wf="annual", page="myproject") a 1950 2005
wfcreate(wf="unstruct") u 1000
```

creates two workfiles. The first is a workfile named ANNUAL containing a single page named MYPROJECT containing annual data from 1950 to 2005; the second is a workfile named UNSTRUCT containing a single page named UNDATED with 1000 unstructured observations.

```
wfcreate(wf="griliches_grunfeld") a 1935 1954 10
```

creates a workfile named GRILICHES_GRUNFELD with an ANNUAL page containing 10 cross-section annual panel for the years 1935 to 1954.

**Cross-references**

See also `pagecreate` (p. 355) and `pagedelete` (p. 356).
Open a workfile. Reads in a previously saved workfile, or reads the contents of a foreign data source into a workfile.

The opened workfile becomes the default workfile; existing workfiles in memory remain on the desktop but become inactive.

**Syntax**

**Command:**

```
wfopen [path]\workfile_name
```

**Command:**

```
wfopen( options ) [path]\datasource [ @keep keep_list] [ @drop drop_list ] [ @keepmap keepmap_list] [ @dropmap dropmap_list] [ @selectif condition]
```

**Command:**

```
wfopen( options ) table_spec [ @keep keep_list] [ @drop drop_list ] [ @keepmap keepmap_list] [ @dropmap dropmap_list] [ @selectif condition]
```

**Options**


- **wf = wf_name** Optional name for the new workfile.

- **page = page_name** Optional name for the page in the new workfile.

The following options may be required when connecting to a remote server:
Examples

\texttt{wfopen c:\data\macro}

loads a previously saved EViews workfile MACRO.WF1 from the DATA directory in the C drive.

\texttt{wfopen c:\tsp\nipa.wf}

loads a MicroTSP workfile NIPA.WF. If you do not use the workfile type option, you should add the extension ".WF" to the workfile name when loading a DOS MicroTSP workfile. An alternative method uses the type "d" option:

\texttt{wfopen(d) nipa}

Cross-references


See also \texttt{pageload}, \texttt{read}, \texttt{fetch}, \texttt{wfsave}, and \texttt{pagesave}.

<table>
<thead>
<tr>
<th>wfsave</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Save the default workfile as an EViews workfile (.wf1 file), or optionally as a foreign file or data source.</td>
</tr>
</tbody>
</table>

Syntax

Command: \texttt{wfsave(options) [path\]filename}

Command: \texttt{wfsave(options) [path\]datasource [@keep keep_list] [@drop drop_list] [@keepmap keepmap_list] [@dropmap dropmap_list] [@smpl smpl_spec]}

Command: \texttt{wfsave(options) table_spec [@keep keep_list] [@drop drop_list] [@keepmap keepmap_list] [@dropmap dropmap_list] [@smpl smpl_spec]}

$s = server\_id$, Server name.
$server = server\_id$
$u = user$, Username.
$username = user$
$p = pswd$, Password.
$password = pswd$
sml_spec]
saves the active workfile in the specified directory using filename. By default, the workfile is saved as an EViews workfile, but options may be used to save all or part of the active page in a foreign file or data source.

Options

Workfile Save Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Save using single precision.</td>
</tr>
<tr>
<td>2</td>
<td>Save using double precision.</td>
</tr>
<tr>
<td>c</td>
<td>Save compressed workfile (not compatible with versions of EViews prior to 5.0).</td>
</tr>
</tbody>
</table>

The default workfile save settings use the Global Options.

Foreign Source Save Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode=create</td>
<td>Create new file only; error on attempt to overwrite.</td>
</tr>
<tr>
<td>map=arg</td>
<td>Write selected maps as: “n” (numeric), “c” (character), “b” (both numeric and character).</td>
</tr>
<tr>
<td>mapval</td>
<td>Write mapped values for series with attached value labels.</td>
</tr>
</tbody>
</table>

The following options may be required when connecting to a remote server:
Cross-references

See also pagesave (p. 358), wfopen (p. 492), and pageload (p. 357).

<table>
<thead>
<tr>
<th>wfselect</th>
<th>Command</th>
</tr>
</thead>
</table>

Make the selected workfile page the active workfile page.

Syntax

Command:

```
wfselect wfname[\pgname]
```

where `wfname` is the name of a workfile that has been loaded into memory. You may optionally provide the name of a page in the new default workfile that you wish to be made active.

Examples

```
wfselect myproject
wfselect myproject\page2
```

change the default workfile to MYPROJECT. The first command uses the default active page, while the second changes the page to PAGE2.

Examples

Cross-references

See also pageselect (p. 360).

<table>
<thead>
<tr>
<th>white</th>
<th>Equation View</th>
<th>Var View</th>
</tr>
</thead>
</table>

Performs White’s test for heteroskedasticity of residuals.

Carries out White’s test for heteroskedasticity of the residuals of the specified equation or Var. By default, the test is computed without the cross-product terms (using only the terms
involving the original variables and squares of the original variables). You may elect to compute the original form of the White test including the cross-products.

White’s test is not available for equations estimated by binary, ordered, censored, or count. For a Var object, the command computes the multivariate version of the test.

**Syntax**

**Equation View:**  
`eq_name.white(options)`  

**Var View:**  
`var_name.white(options)`  

**Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>c</strong></td>
<td>Include all possible nonredundant cross-product terms in the test regression.</td>
</tr>
<tr>
<td><strong>p</strong></td>
<td>Print the test results.</td>
</tr>
</tbody>
</table>

**Options for Var View**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name=arg</code></td>
<td>Save test statistics in named matrix object. See below for a description of the statistics stored in the matrix.</td>
</tr>
</tbody>
</table>

For var views, the “name= ” option stores a $(r + 1) \times 5$ matrix, where $r$ is the number of unique residual cross-product terms. For a VAR with $k$ endogenous variables, $r = k(k + 1) / 2$. The first $r$ rows contain statistics for each individual test equation, where the first column is the regression R-squared, the second column is the $F$-statistic, the third column is the $p$-value of $F$-statistic, the 4th column is the $T \times R^2$ chi-square statistic, and the fifth column is the $p$-value of the chi-square statistic. The numerator and denominator degrees of freedom of the $F$-statistic are stored in the $(r + 1)$-st row, third and fourth columns, while the chi-square degrees of freedom is stored in the $(r + 1)$-st row, fifth column.

In the $(r + 1)$-st row and first column contains the joint (system) LM chi-square statistic and the second column contains the degrees of freedom of this $\chi^2$ statistic.

**Examples**

`eq1.white(c)`

Carries out the White test of heteroskedasticity including all possible cross-product terms.

**Cross-references**

See “White’s Heteroskedasticity Test” on page 552 for a discussion of White’s test. For the multivariate version of this test, see page 698 of the User’s Guide.
Estimates a system of equations using weighted least squares.

To perform weighted least squares in single equation estimation, see `ls` (p. 315).

**Syntax**

```
System Method: system_name.wls(options)
```

**Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>i</code></td>
<td>Iterate simultaneously over the weighting matrix and coefficient vector.</td>
</tr>
<tr>
<td><code>s</code></td>
<td>Iterate sequentially over the computation of the weighting matrix and the estimation of the coefficient vector.</td>
</tr>
<tr>
<td><code>o</code> (default)</td>
<td>Iterate the estimate of the coefficient vector to convergence following one-iteration of the weighting matrix.</td>
</tr>
<tr>
<td><code>c</code></td>
<td>One step (iteration) of the coefficient vector estimates following one iteration of the weighting matrix.</td>
</tr>
<tr>
<td><code>m = integer</code></td>
<td>Maximum number of iterations.</td>
</tr>
<tr>
<td><code>c = number</code></td>
<td>Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.</td>
</tr>
<tr>
<td><code>l = number</code></td>
<td>Set maximum number of iterations on the first-stage coefficient estimation to get one-step weighting matrix.</td>
</tr>
</tbody>
</table>
Examples

\texttt{sys1.wls}

estimates the system of equations in SYS1 by weighted least squares.

Cross-references


See also the available options for weighted least squares in \texttt{ls} (p. 315).

\begin{tabular}{|l|l|}
\hline
\textbf{workfile} & \textbf{Command} \\
\hline
\end{tabular}

Create or change workfiles.

No longer supported; provided for backward compatibility. This command has been replaced by \texttt{wcreate} (p. 491) and \texttt{pageselect} (p. 360).

\begin{tabular}{|l|l|}
\hline
\hline
\end{tabular}

Write EViews data to a text (ASCII), Excel, or Lotus file on disk.

Creates a foreign format disk file containing EViews data. May be used to export EViews data to another program.

Syntax

\begin{itemize}
\item Command: \texttt{write(options) [path\filename arg1 arg2 arg3 \ldots]}
\item Pool Proc: \texttt{pool\_name.write(options) [path\filename] pool\_series1}
\end{itemize}
Follow the keyword by a name for the output file and list the series to be written. The optional path name may be on the local machine, or may point to a network drive. If the path name contains spaces, enclose the entire expression in double quotation marks. To write matrix objects, simply provide a filename; the entire matrix will be exported.

Note that EViews cannot, at present, write into an existing file. The file that you select will, if necessary, be replaced.

**Options**

Options are specified in parentheses after the keyword and are used to specify the format of the output file.

**File type**

- **t=dat, txt** ASCII (plain text) files.
- **t wk1, wk3** Lotus spreadsheet files.
- **t xls** Excel spreadsheet files.

If you omit the “t=” option, EViews will determine the type based on the file extension. Unrecognized extensions will be treated as ASCII files. For Lotus and Excel spreadsheet files specified without the “t=” option, EViews will automatically append the appropriate extension if it is not otherwise specified.

**ASCII text files**

- **na = string** Specify text string for NAs. Default is “NA”.
- **names (default) / nonames** [Write / Do not write] series names.
- **dates (default) / nodates** [Write / Do not write] dates/obs and (pool) cross-section identifiers (only for pool write).
- **bycross (default) / byper** Stack pool data by [cross-section / date or period] (only for pool write).
- **id** Write cross-section identifier (only for pool write).
**Spreadsheet (Lotus, Excel) files**

- **d = arg** Specify delimiter (*default* is tab): “s” (space), “c” (comma).
- **t** Write by series (or transpose the data for matrix objects). Default is to read by obs with series in columns.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>letter_number</strong></td>
<td>Coordinate of the upper-left cell containing data.</td>
</tr>
<tr>
<td><strong>names</strong></td>
<td>Series names. [Write / Do not write]</td>
</tr>
<tr>
<td><strong>dates</strong></td>
<td>Dates/obs and (pool) cross-section identifiers (only for pool write)</td>
</tr>
<tr>
<td><strong>bycross</strong></td>
<td>Stack pool data by [cross-section / date or period] (only for pool write).</td>
</tr>
<tr>
<td><strong>id</strong></td>
<td>Cross-section identifier (only for pool write).</td>
</tr>
<tr>
<td><strong>dates = arg</strong></td>
<td>Excel format for writing date: “first” (convert to the first day of the corresponding observation if necessary), “last” (convert to the last day of the corresponding observation).</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
write(t=txt, na=., d=c, dates) a: \ dat1.csv hat1 hat_se1
```

Writes the two series HAT1 and HAT_SEL into an ASCII file named DAT1.CSV on the A drive. The data file is listed by observations, NAs are coded as “.” (dot), each series is separated by a comma, and the date/observation numbers are written together with the series names.

```plaintext
write(t=txt, na=., d=c, dates) dat1.csv hat1 hat_se1
```

writes the same file in the default directory.

```plaintext
mypool.write(t=xls, per) "\\ network\ drive a\ growth" gdp? edu?
```

writes an Excel file GROWTH.XLS in the specified directory. The data are organized by observations and are listed by period/time.
Cross-references
See “Exporting to a Spreadsheet or Text File” on page 95 of the User’s Guide for a discussion. See also read (p. 377).

WtSls

Perform weighted two-stage least squares estimation of a system of equations.
To perform weighted two-stage least squares in a single equation, see ls (p. 315).

Syntax
System Method: system_name.wtSls(options)

Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Iterate simultaneously over the weighting matrix and coefficient vector.</td>
</tr>
<tr>
<td>s</td>
<td>Iterate sequentially over the computation of the weighting matrix and the estimation of the coefficient vector.</td>
</tr>
<tr>
<td>o</td>
<td>Iterate the coefficient vector to convergence following one-iteration of the weighting matrix. (default)</td>
</tr>
<tr>
<td>c</td>
<td>One step (iteration) of the coefficient vector following one iteration of the weighting matrix.</td>
</tr>
<tr>
<td>m = integer</td>
<td>Maximum number of iterations.</td>
</tr>
<tr>
<td>c = number</td>
<td>Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.</td>
</tr>
<tr>
<td>l = number</td>
<td>Set maximum number of iterations on the first-stage iteration to get the one-step weighting matrix.</td>
</tr>
</tbody>
</table>
Examples

sys1.wtsls

estimates the system of equations in SYS1 by weighted two-stage least squares.

Cross-references

See “Weighted Two-Stage Least Squares” on page 667 of the User’s Guide for further discussion.

See also tsls (p. 474) for both unweighted and weighted single equation 2SLS.

Seasonally adjust series using the Census X11.2 method.

Syntax

Series Proc: series_name.x11(options) adj_name [fac_name]

The X11 procedure carries out Census X11.2 seasonal adjustment. Enter the name of the original series followed by a period, the keyword, and then provide a name for the seasonally adjusted series. You may optionally list a second series name for the seasonal factors. The seasonal adjustment method is specified as an option in parentheses after the x11 keyword.

The X11 procedure is available only for quarterly and monthly series. The procedure requires at least four full years of data, and can adjust up to 20 years of monthly data and 30 years of quarterly data.
Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>Multiplicative seasonals.</td>
</tr>
<tr>
<td>a</td>
<td>Additive seasonals.</td>
</tr>
<tr>
<td>s</td>
<td>Use sliding spans.</td>
</tr>
<tr>
<td>h</td>
<td>Adjustment for all holidays (only for monthly data specified with the m option).</td>
</tr>
<tr>
<td>i</td>
<td>Adjustment for holidays if significant (only for monthly data specified with the “m” option).</td>
</tr>
<tr>
<td>t</td>
<td>Adjustment for all trading days (only for monthly data).</td>
</tr>
<tr>
<td>q</td>
<td>Adjustment for trading days if significant (only for monthly data).</td>
</tr>
<tr>
<td>p</td>
<td>Print the X11 results.</td>
</tr>
</tbody>
</table>

Examples

```
sales.x11(m,h) salesx11 salesfac
```

Seasonally adjusts the SALES series and saves the adjusted series as SALESX11 and the seasonal factors as SALESFAC. The adjustment assumes multiplicative seasonals and makes adjustment for all holidays.

Cross-references


Note that the X11 routines are separate programs provided by the Census and are installed in the EViews directory in the files X11Q2.EXE and X11SS.EXE. The documentation for these programs can also be found in your EViews directory as text files X11DOC1.TXT through X11DOC3.TXT.

See also seas (p. 403), seasplot (p. 404), tramoseats (p. 471), and x12 (p. 504).
Seasonally adjust series using the Census X12 method.

\texttt{x12} is available only for quarterly and monthly series. The procedure requires at least 3 full years of data and can adjust up to 600 observations (50 years of monthly data or 150 years of quarterly data).

**Syntax**

\begin{verbatim}
Series Proc: series_name.x12(options) base_name
\end{verbatim}

Enter the name of the original series followed by a dot, the keyword, and a base name (no more than the maximum length of a series name minus 4) for the saved series. If you do not provide a base name, the original series name will be used as a base name. See the description in “save=” option below for the naming convention used to save series.

**Options**

*Commonly Used Options*

- **mode = arg**
  
  (default = “m”)

  Seasonal adjustment method: “m” (multiplicative adjustment; \textit{Series must take only non-negative values}), “a” (additive adjustment), “p” (pseudo-additive adjustment), “l” (log-additive seasonal adjustment; \textit{Series must take only positive values}).

- **filter = arg**
  
  (default = “msr”)

  Seasonal filter: “msr” (automatic, moving seasonality ratio), “x11” (X11 default), “stable” (stable), “s3x1” (3x1 moving average), “s3x3” (3x3 moving average), “s3x5” (3x5 moving average), “s3x9” (3x9 moving average), “s3x15” (3x15 moving average seasonal filter; \textit{Series must have at least 20 years of data}).
save = “arg” Optionally saved series keyword enclosed in quotes. List the extension (given in Table 6-8, p.71 of the X12-ARIMA Reference Manual) for the series you want to save. The created series will use names of the form `basename`, followed by a series keyword specific suffix. Commonly used options and suffixes are: “d10” (final seasonal factors, saved with suffix “_sf”), “d11” (final seasonally adjusted series using “_sa”), “d12” (final trend-cycle component using “_tc”), “d13” (final irregular component using “_ir”). All other options are named using the option symbol. For example `save = “d16”` will store a series named `basename_D16`. To save more than two series, separate the list with a space. For example, `save = “d10 d12”` saves the seasonal factors and the trend-cycle series.

tf = arg Transformation for regARIMA: “logit” (Logit transformation), “auto” (automatically choose between no transformation and log transformation), number (Box-Cox power transformation using specified parameter; use “tf = 0” for log transformation).

sspan Sliding spans stability analysis. Cannot be used along with the “h” option.

history Historical record of seasonal adjustment revisions. Cannot be used along with the “sspan” option.

check Check residuals of regARIMA.

outlier Outlier analysis of regARIMA.

x11reg = arg Regressors to model the irregular component in seasonal adjustment. Regressors must be chosen from the predefined list in Table 6-14, p. 88 of the X12-ARIMA Reference Manual. To specify more than one regressor, separate by a space within the double quotes.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>reg = arg_list</code></td>
<td>Regressors for the regARIMA model. Regressors must be chosen from the predefined list in Table 6-17, pp. 100-101 of the X12-ARIMA Reference Manual. To specify more than one regressor, separate by a space within the double quotes.</td>
</tr>
<tr>
<td><code>arima = arg</code></td>
<td>ARIMA spec of the regARIMA model. Must follow the X12 ARIMA specification syntax. Cannot be used together with the “amdl = ” option.</td>
</tr>
<tr>
<td><code>amdl = f</code></td>
<td>Automatically choose the ARIMA spec. Use forecasts from the chosen model in seasonal adjustment. Cannot be used together with the arima = option and must be used together with the mfile = option.</td>
</tr>
<tr>
<td><code>amdl = b</code></td>
<td>Automatically choose the ARIMA spec. Use forecasts and backcasts from the chosen model in seasonal adjustment. Cannot be used together with the “arima = ” option and must be used together with the “mfile = ” option.</td>
</tr>
<tr>
<td><code>best</code></td>
<td>Sets the method option of the auto model spec to best (default is first). Also sets the identify option of the auto model spec to all (default is first). Must be used together with the “amdl = ” option.</td>
</tr>
<tr>
<td><code>modelsmpl = arg</code></td>
<td>Sets the subsample for fitting the ARIMA model. Either specify a sample object name or a sample range. The model sample must be a subsample of the current workfile sample and should not contain any breaks.</td>
</tr>
<tr>
<td><code>mfile = arg</code></td>
<td>Specifies the file name (include the extension, if any) that contains a list of ARIMA specifications to choose from. Must be used together with the “amdl = ” option. The default is the X12A.MDL file provided by the Census.</td>
</tr>
<tr>
<td><code>outsmp</code></td>
<td>Use out-of-sample forecasts for automatic model selection. Default is in-sample forecasts. Must be used together with the “amdl = ” option.</td>
</tr>
</tbody>
</table>
plotspectra  Save graph of spectra for differenced seasonally adjusted series and outlier modified irregular series. The saved graph will be named GR_seriesname_SP.

p  Print X12 procedure results.

**Other Options**

hma = integer  Specifies the Henderson moving average to estimate the final trend-cycle. The X12 default is automatically selected based on the data. To override this default, specify an odd integer between 1 and 101.

sigl = arg  Specifies the lower sigma limit used to downweight extreme irregulars in the seasonal adjustment. The default is 1.5 and you can specify any positive real number.

sigh = arg  Specifies the upper sigma limit used to downweight extreme irregulars in the seasonal adjustment. The default is 2.5 and you can specify any positive real number less than the lower sigma limit.

ea  Nonparametric Easter holiday adjustment (x11easter). *Cannot be used together with the “easter[w]” regressor in the “reg = ” or “x11reg = ” options.*

f  Appends forecasts up to one year to some optionally saved series. Forecasts are appended only to the following series specified in the “save = ” option: “b1” (original series, adjusted for prior effects), “d10” (final seasonal factors), “d16” (combined seasonal and trading day factors).

flead = integer  Specifies the number of periods to forecast (to be used in the seasonal adjustment procedure). The default is one year and you can specify an integer up to 60.
User provided spec file

EViews provides most of the basic options available in the X12 program. For users who need to access the full set of options, we have provided an option to pass your own X12 specification file from EViews. The advantage of using this option (as opposed to running the X12 program in DOS) is that EViews will automatically handle the data in the input and output series.

To provide your own specification file, specify the path/name of your file using the “sfile=” option in the x12 proc. Your specification file should follow the format of an X12 specification file as described in the X12-ARIMA Reference Manual, except

- the specification file should not have a series spec nor a composite spec.
- the x11 spec must have a save option for D11 (the final seasonally adjusted series) and any other extensions you want to store. EViews will always look for D11 and will error if not found.
- to read back data for a save option other than D11, you must use the “save=” option in the x12 proc. For example, to get back the final trend-cycle series (D12) into EViews, you must have a “save=” option for D12 (and D11) in the x11 spec of your specification file and a “save=d12” option in the EViews x12 proc.

Note that when you use an “sfile=” option, EViews will ignore any other options in the x12 proc, except for the “save=” option.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fback = integer</td>
<td>Specifies the number of periods to backcast (to be used in the seasonal adjustment procedure). The default is 0 and you can specify an integer up to 60. No backcasts are produced for series more than 15 years long.</td>
</tr>
<tr>
<td>aicx11</td>
<td>Test (based on AIC) whether to retain the regressors specified in “x11reg = “. Must be used together with the “x11reg = “ option.</td>
</tr>
<tr>
<td>aicreg</td>
<td>Test (based on AIC) whether to retain the regressors specified in “reg = “. Must be used together with the “reg = “ option.</td>
</tr>
<tr>
<td>sfile = arg</td>
<td>Path/name (including extension, if any) of user provided specification file. The file must follow a specific format; see the discussion below.</td>
</tr>
</tbody>
</table>
**Difference between the dialog and command line**

The options corresponding to the **Trading Day/Holiday** and **Outliers** tab in the X12 dialog should be specified by listing the appropriate regressors in the “x11reg=” and “reg=” options.

**Examples**

The command

```plaintext
sales.x12(mode=m,save="d10 d12") salesx12
```

seasonally adjusts the SALES series in multiplicative mode. The seasonal factors (d10) are stored as SALESX12_SF and the trend-cycles series is stored as SALESX12_TC.

```plaintext
sales.x12(tf=0,arima="(0 0 1)" ,reg="const td")
```

specifies a regARIMA model with a constant, trading day effect variables, and MA(1) using a log transformation. This command does not store any series.

```plaintext
freeze(x12out) sales.x12(tf=auto, amdl=f, mfile="c:\eviews\mymdl.txt")
```

stores the output from X12 in a text object named X12OUT. The options specify an automatic transformation and an automatic model selection from the file MYMDL.TXT.

```plaintext
revenue.x12(tf=auto,sfile="c:\eviews\spec1.txt",save="d12 d13")
```

adjusts the series REVENUE using the options given in the SPEC1.TXT file. Note the following: (1) the "tf=auto" option will be ignored (you should specify this in your specification file) and (2) EViews will save two series REVENUE_TC and REVENUE_IR which will be filled with NAs unless you provided the save option for D12 and D13 in your specification file.

```plaintext
freeze(x12out) sales.x12(tf=auto, amdl=f, mfile="c:\eviews\mymdl.txt")
```

stores the output from X12 in a text object named X12OUT. The options specify an automatic transformation and an automatic model selection from the file MYMDL.TXT. The seasonally adjusted series is stored as SALES_SA by default.

```plaintext
revenue.x12(tf=auto,sfile="c:\eviews\spec1.txt",save="d12 d13")
```

adjusts the series REVENUE using the options given in the SPEC1.TXT file. Note the following: (1) the "tf=auto" option will be ignored (you should specify this in your specification file) and (2) EViews will error if you did not specify a save option for D11, D12, and D13 in your specification file.
Cross-references

See “Census X12” on page 312 of the User’s Guide for a discussion of the Census X12 program. The documentation for X12, X12-ARIMA Reference Manual, can be found in the DOCS subdirectory of your EViews directory in the PDF files FINALPT1.PDF and FINALPT2.PDF.

See also seas (p. 403) and x11 (p. 502).

| xy | Command || Graph Command | Group View | Matrix View | Sym View |

Display XY graph of object, or change existing graph object type to XY (if possible).

By default, the first series or column of data will be located along the horizontal axis and the remaining data on the vertical axis. You may optionally choose to plot the data in pairs, where the first two series or columns are plotted against each other, the second two series or columns are plotted against each other, and so forth.

Syntax

Command: \texttt{xy(options) arg1 \{arg2 arg3 ...\}}
Object View: \texttt{group\_name.xy(options)}
Graph Proc: \texttt{graph\_name.xy(options)}

If used as a command, follow the keyword by a list of series and group objects, or by a matrix object. There must be at least two series or columns in the data to be graphed.

If changing the type of a graph, the default behavior is to use the existing settings for lines and symbols in the graph.

See scat (p. 399) and xyline (p. 512) if you wish to create an XY graph with specific line/symbol settings, or use setelem (p. 412) to change the settings after the graph is created.

Options

Options may be specified in parentheses after the keyword.

Template and printing options

- \texttt{o=graph\_name} Use appearance options from the specified graph.
- \texttt{t=graph\_name} Use appearance options and copy text and shading from the specified graph.
- \texttt{p} Print the XY line graph.
Note that use of a template will override the existing line and symbol settings.

**Scale options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a (default)</td>
<td>Automatic single scale.</td>
</tr>
<tr>
<td>b</td>
<td>Plot series or columns in pairs (the first two against each other, the second two against each other, and so forth).</td>
</tr>
<tr>
<td>n</td>
<td>Normalized scale (zero mean and unit standard deviation). May not be used with the “s” option.</td>
</tr>
<tr>
<td>d</td>
<td>Dual scaling with no crossing.</td>
</tr>
<tr>
<td>x</td>
<td>Dual scaling with possible crossing.</td>
</tr>
<tr>
<td>m</td>
<td>Display XY plots in multiple graphs (will override the “s” option). Not for use with an existing graph object.</td>
</tr>
</tbody>
</table>

**Panel options**

The following options apply when graphing panel structured data.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>panel = arg (default taken from global settings)</td>
<td>Panel data display: “stack” (stack the cross-sections), “individual” or “1” (separate graph for each cross-section), “combine” or “c” (combine each cross-section in single graph; one time axis), “mean” (plot means across cross-sections), “mean1se” (plot mean and +/- 1 standard deviation summaries), “mean2sd” (plot mean and +/- 2 s.d. summaries), “mean3sd” (plot mean and +/- 3 s.d. summaries), “median” (plot median across cross-sections), “med25” (plot median and +/- .25 quartiles), “med10” (plot median and +/- .10 quartiles), “med05” (plot median +/- .05 quartiles), “med025” (plot median +/- .025 quartiles), “med005” (plot median +/- .005 quartiles), “med-mxmn” (plot median, max and min).</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
group gl inf unemp gdp inv
g1.xy(o=graf)
```
plots INF on the horizontal axis and UNEMP, GDP and INV on the vertical axis, using the graph object GRA1 as a template.

\[ g1.xy(b) \]
\[ g1.xy(b, m) \]

plots INF against UNEMP and GDP against INV in first in a single graph, and then in multiple graphs.

If there is an existing graph GRAPH01, the expression

\[ \text{graph01.xy}(m, n) \]

changes its type to an XY plot displayed in multiple graphs with normalized scales, with the remaining XY graph settings at their default values, and using the existing GRAPH01 settings for line and symbol display.

Cross-references


See graph (p. 288) for graph declaration and additional graph types and options (p. 343) for graph options. scat (p. 399) and xyline (p. 512) are specialized forms of XY graphs.

| xyline | Command || Graph Command | Group View | Matrix View | Sym View |
|--------|----------|----------------------|------------|-------------|----------|

Display XY line graph, or change existing graph object type to XY line (if possible).

By default, the first series or column of data will be located along the horizontal axis and the remaining data on the vertical axis. You may optionally choose to plot the data in pairs, where the first two series or columns are plotted against each other, the second two series or columns are plotted against each other, and so forth.

Syntax

Command: \text{xyline}(options) \text{arg1 [arg2 arg3 ...]}

Object View: \text{group_name}.xyline(options)

Graph Proc: \text{graph_name}.xyline(options)

If used as a command, follow the keyword by a list of series and group objects, or by a matrix object. There must be at least two series or columns in the data to be graphed.

XY line graphs are simply XY plots (see \text{xy} (p. 510)) with lines turned on, and symbols turned off (see setelem (p. 412)).
Options

Options may be specified in parentheses after the keyword.

Template and printing options

- **o = graph_name** Use appearance options from the specified graph.
- **t = graph_name** Use appearance options and copy text and shading from the specified graph.
- **p** Print the XY graph.

**Note that use of the template option will override the lines setting.**

Scale options

- **a (default)** Automatic single scale.
- **b** Plot series or columns in pairs (the first two against each other, the second two against each other, and so forth).
- **n** Normalized scale (zero mean and unit standard deviation). May not be used with the “s” option.
- **d** Dual scaling with no crossing.
- **x** Dual scaling with possible crossing.
- **m** Display XY plots in multiple graphs (will override the “s” option). Not for use with an existing graph object.

Panel options

The following options apply when graphing panel structured data.
Examples

\begin{verbatim}
  group g1 inf unemp gdp inv
  g1.xyline(o=gra1)
\end{verbatim}

plots, in a single graph, INF (on the vertical axis) against UNEMP and GDP against INV, using the graph object GRA1 as a template.

\begin{verbatim}
  g1.xyline
  g1.xyline(m)
\end{verbatim}

plots INF against UNEMP and GDP against INV in first in a single graph, and then in multiple graphs.

If there is an existing graph GRAPH01, the expression

\begin{verbatim}
  graph01.xyline(d)
\end{verbatim}

changes its type to an XY line plot with dual scales and no crossing, with the remaining XY graph settings at their default values.

Cross-references


See xy (p. 510) and graph (p. 288) for graph declaration and additional graph types and options (p. 343) for graph options. See scat (p. 399) for XY scatterplots.
**xypair**

Display XY pairs graph, or change existing graph object type to XY pairs (if possible).

Plots the data in pairs, where the first two series or columns are plotted against each other, the second two series or columns are plotted against each other, and so forth.

**Syntax**

Command: `xypair(options) arg1 [arg2 arg3 ...]`

Object View: `group_name.xypair(options)`

Graph Proc: `graph_name.xypair(options)`

If used as a command, follow the keyword by a list of series and group objects, or by a matrix object. There must be at least two series or columns in the data to be graphed.

If changing the type of a graph, the default behavior is to use the existing settings for lines and symbols in the graph.

This graph type is equivalent to using `xy` (p. 510) with the “b” option indicating that the data should be graphed in pairs.

**Options**

Options may be specified in parentheses after the keyword.

*Template and printing options*

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>o = <code>graph_name</code></td>
<td>Use appearance options from the specified graph.</td>
</tr>
<tr>
<td>t = <code>graph_name</code></td>
<td>Use appearance options and copy text and shading from the specified graph.</td>
</tr>
<tr>
<td>p</td>
<td>Print the XY graph.</td>
</tr>
</tbody>
</table>

*Note that the use of a template option will override the pairs setting.*

*Scale options*

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a (default)</td>
<td>Automatic single scale.</td>
</tr>
<tr>
<td>b</td>
<td>Plot series or columns in pairs (the first two against each other, the second two against each other, and so forth).</td>
</tr>
<tr>
<td>n</td>
<td>Normalized scale (zero mean and unit standard deviation). May not be used with the “s” option.</td>
</tr>
</tbody>
</table>
Panel options

The following options apply when graphing panel structured data.

- **d** Dual scaling with no crossing.
- **x** Dual scaling with possible crossing.
- **m** Display XY plots in multiple graphs (will override the “s” option). Not for use with an existing graph object.

**Examples**

```plaintext
    group g1 inf unemp gdp inv
g1.xypair(o=gra1)
```

Plots, in a single graph, INF (on the vertical axis) against UNEMP and GDP against INV, using the graph object GRA1 as a template.

```plaintext
    g1.xypair
    g1.xypair(m)
```

Plots INF against UNEMP and GDP against INV in first in a single graph, and then in multiple graphs.

If there is an existing graph GRAPH01, the expression

```plaintext
    panel=arg
```

(configured from global settings) can be used to display panel data. The display options include: "stack" (stack the cross-sections), "individual" or "1" (separate graph for each cross-section), "combine" or "c" (combine each cross-section in single graph; one time axis), "mean" (plot means across cross-sections), "mean1se" (plot mean and +/- 1 standard deviation summaries), "mean2sd" (plot mean and +/- 2 s.d. summaries), "mean3sd" (plot mean and +/- 3 s.d. summaries), "median" (plot median across cross-sections), "med25" (plot median and +/- .25 quantiles), "med10" (plot median and +/- .10 quantiles), "med05" (plot median +/- .05 quantiles), "med025" (plot median +/- .025 quantiles), "med005" (plot median +/- .005 quantiles), "median-mxmn" (plot median, max and min).
\texttt{graph01.xypair(m, n)}

changes its type to an XY pair plot displayed in multiple graphs with normalized scales, with the remaining XY graph settings at their default values, and using the existing GRAPH01 settings for line and symbol display.

\textbf{Cross-references}


See \texttt{graph} (p. 288) for graph declaration and additional graph types and \texttt{options} (p. 343) for graph options. See \texttt{xy} (p. 510) for the creation of general XY graphs, and \texttt{scat} (p. 399) for scatterplots.
Appendix C. Special Expression Reference

This Appendix provides an alphabetical listing of special expressions that may be used in series assignment and generation, or as terms in estimation specifications.

<table>
<thead>
<tr>
<th>@expand</th>
<th>Equation Expression</th>
</tr>
</thead>
</table>

Automatic dummy variables.

The @expand expression may be added in estimation to indicate the use of one or more automatically created dummy variables.

Syntax

Expression: \[ @expand\{ser1[, ser2, ser3, ...][, drop_spec]\} \]

creates a set of dummy variables that span the unique values of the input series ser1, ser2, etc. The optional drop_spec may be used to drop one or more of the dummy variables. drop_spec may contain the keyword “@DROPFIRST” (indicating that you wish to drop the first category), “@DROPLAST” (to drop the last category), or a description of an explicit category, using the syntax:

\[ @DROP\{val1[, val2, val3,...]\} \]

where each argument corresponds to a category in @EXPAND. You may use the wild card “*” to indicate all values of a corresponding category.

Example

For example consider the following two variables:

- SEX is a numeric series which takes the values 1 and 0.
- REGION is an alpha series which takes the values “North”, “South”, “East”, and “West”.

The command

\[ eq.ls income @expand\{sex\} age \]

regresses INCOME on two dummy variables, one for “SEX = 0” and one for “SEX = 1” as well as the simple regressor AGE.

The @EXPAND statement in
eq.ls income @expand(sex, region) age

creates 8 dummy variables corresponding to

sex = 0, region = "North"
sex = 0, region = "South"
sex = 0, region = "East"
sex = 0, region = "West"
sex = 1, region = "North"
sex = 1, region = "South"
sex = 1, region = "East"
sex = 1, region = "West"

The expression

@expand(sex, region, @dropfirst)

creates the set of dummies defined above, but no dummy is created for “SEX=0, REGION="North"”. In

@expand(sex, region, @droplast)

no dummy is created for “SEX=1, REGION="WEST"”.

The expression

@expand(sex, region, @drop(0,"West"), @drop(1,"North"))

create a set of dummy variables from SEX and REGION pairs, but no dummy is created for “SEX=0, REGION="West"” and “SEX=1, REGION="North"”

@expand(sex, region, @drop(1,*))

specifies that dummy variables for all values of REGION where “SEX=1” should be dropped.

Cross-references

See “Automatic Categorical Dummy Variables” on page 434 of the User’s Guide for further discussion.
Autoregressive error specification.

The AR specification can appear in an `ls` (p. 315) or `tsls` (p. 474) specification to indicate an autoregressive component. `ar(1)` indicates the first order component, `ar(2)` indicates the second order component, and so on.

**Examples**

The command

```
ls m1 c tb3 tb3(-1) ar(1) ar(4)
```

regresses M1 on a constant, TB3, and TB3 lagged once with a first order and fourth order autoregressive component. The command

```
tsls sale c adv ar(1) ar(2) ar(3) ar(4) @ c gdp
```

performs two-stage least squares of SALE on a constant and ADV with up to fourth order autoregressive components using a constant and GDP as instruments.

**Cross-references**


See also `sar` (p. 525), `ma` (p. 521), and `sma` (p. 526).

Moving average error specification.

The `ma` specification may be added in an `ls` (p. 315) or `tsls` (p. 474) specification to indicate a moving average error component. `ma(1)` indicates the first order component, `ma(2)` indicates the second order component, and so on.

**Examples**

```
ls(z) m1 c tb3 tb3(-1) ma(1) ma(2)
```

regresses M1 on a constant, TB3, and TB3 lagged once with first order and second order moving average error components. The “z” option turns off backcasting in estimation.
Cross-references


See also \texttt{sma} (p. 526), \texttt{ar} (p. 521), and \texttt{sar} (p. 525).

<table>
<thead>
<tr>
<th>\textbf{na}</th>
<th>\textbf{Series Expression}</th>
</tr>
</thead>
</table>

Not available code. “NA” is used to represent missing observations.

{

\textbf{Examples}

\begin{verbatim}
smpl if y >= 0
series z = y
smpl if y < 0
z = na
\end{verbatim}

generates a series Z containing the contents of Y, but with all negative values of Y set to “NA”.

NA values will also be generated by mathematical operations that are undefined:

\begin{verbatim}
series y = nrnd
y = log(y)
\end{verbatim}

will replace all positive value of Y with \( \log(Y) \) and all negative values with “NA”.

\begin{verbatim}
series test = (yt <> na)
\end{verbatim}

creates the series TEST which takes the value one for nonmissing observations of the series YT. A zero value of TEST indicates missing values of the series YT.

Note that the behavior of missing values has changed since EViews Version 2. Previously, NA values were coded as 1e-37. This implied that in EViews 2 you could use the expression

\begin{verbatim}
series z = (y>=0)*x + (y<0)*na
\end{verbatim}

to return the value of Y for non-negative values of Y and “NA” for negative values of Y. This expression will now generate the value “NA” for all values of Y, since mathematical expressions involving missing values always return “NA”. You must now use the \texttt{smpl} statement as in the first example above, or the \texttt{@recode} function.
Cross-references


<table>
<thead>
<tr>
<th>nrnd</th>
<th>Series Expression</th>
</tr>
</thead>
</table>

Normal random number generator.

When used in a series expression, `nrnd` generates (pseudo) random draws from a normal distribution with zero mean and unit variance.

Examples

```
smpl @first @first
series y = 0
smpl @first+1 @last
series y = .6*y(-1) + .5*nrnd
```

generates a Y series that follows an AR(1) process with initial value zero. The innovations are normally distributed with mean zero and standard deviation 0.5.

```
series u = 10 + @sqr(3)*nrnd
series z = u + .5*u(-1)
```

generates a Z series that follows an MA(1) process. The innovations are normally distributed with mean 10 and variance 3.

```
series x = nrnd^2 + nrnd^2 + nrnd^2
```

generates an X series as the sum of squares of three independent standard normal random variables, which has a $\chi^2(3)$ distribution. Note that adding the sum of the three series is not the same as issuing the command

```
series x = 3*nrnd^2
```

since the latter involves the generation of a single random variable.

The command

```
series x = @qchisq(rnd, 3)
```

provides an alternative method of simulating random draws from a $\chi^2(3)$ distribution.

Cross-references

See “Statistical Distribution Functions” on page 584 for a list of other random number generating functions from various distributions.
See also \texttt{rnd} (p. 525), \texttt{rndint} (p. 388) and \texttt{rndseed} (p. 389).

<table>
<thead>
<tr>
<th>\textbf{pdl}</th>
<th>\textbf{Equation Expression}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Polynomial distributed lag specification. This expression allows you to estimate polynomial distributed lag specifications in \texttt{ls} or \texttt{tsls} estimation. \texttt{pdl} forces the coefficients of a distributed lag to lie on a polynomial. The expression can only be used in estimation by list.</td>
</tr>
</tbody>
</table>

**Syntax**

Expression: \texttt{pdl(series\_name, lags, order[,options])}

**Options**

The PDL specification must be provided in parentheses after the keyword \texttt{pdl} in the following order: the name of the series to which to fit a polynomial lag, the number of lags to include, the order (degree) of polynomial to fit, and an option number to constrain the PDL. By default, EViews does not constrain the endpoints of the PDL.

The constraint options are:

1 Constrain the near end of the distribution to zero.
2 Constrain the far end of the distribution to zero.
3 Constrain both the near and far end of the distribution to zero.

**Examples**

\texttt{ls sale c pdl(order,8,3) ar(1) ar(2)}

fits a third degree polynomial to the coefficients of eight lags of the regressor ORDER.

\texttt{tsls sale c pdl(order,12,3,2) \& c pdl(rain,12,6)}

fits a third degree polynomial to the coefficients of twelve lags of ORDER, constraining the far end to be zero. Estimation is by two-stage least squares, using a constant and a sixth degree polynomial fit to twelve lags of RAIN.

\texttt{tsls y c x1 x2 pdl(z,12,3,2) \& c pdl(*) z2 z3 z4}

When the PDL variable is exogenous in 2SLS, you may use “\texttt{pdl(*)}” in the instrument list instead of repeating the full PDL specification.
Cross-references

See “Polynomial Distributed Lags (PDLs)” on page 429 of the User’s Guide for further discussion.

**rnd**

Series Expression

Uniform random number generator.

Generates (pseudo) random draws from a uniform distribution on (0,1). The expression may be included in a series expression or in an equation to be used in `solve`.

Examples

```plaintext
series u=5+(12-5)*rnd
```

generates a U series drawn from a uniform distribution on (5, 12).

Cross-references

See the list of available random number generators in Appendix F, “Operator and Function Reference”, beginning on page 573.

See also `nrnd` (p. 523), `rndint` (p. 388) and `rndseed` (p. 389).

**sar**

Equation Expression

Seasonal autoregressive error specification.

`sar` can be included in `ls` or `tsls` specification to specify a multiplicative seasonal autoregressive term. A `sar(p)` term can be included in your equation specification to represent a seasonal autoregressive term with lag `p`. The lag polynomial used in estimation is the product of that specified by the `ar` terms and that specified by the `sar` terms. The purpose of the `sar` expression is to allow you to form the product of lag polynomials.

Examples

```plaintext
ls tb3 c ar(1) ar(2) sar(4)
```

TB3 is modeled as a second order autoregressive process with a multiplicative seasonal autoregressive term at lag four.

```plaintext
tsls sale c adv ar(1) sar(12) sar(24) @ c gdp
```

In this two-stage least squares specification, the error term is a first order autoregressive process with multiplicative seasonal autoregressive terms at lags 12 and 24.
Cross-references

See “ARIMA Theory” beginning on page 469 of the User’s Guide for details on ARMA and seasonal ARMA modeling.

See also \texttt{sma} (p. 526), \texttt{ar} (p. 521), and \texttt{ma} (p. 521).

<table>
<thead>
<tr>
<th>\texttt{sma}</th>
<th>Equation Expression</th>
</tr>
</thead>
</table>

Seasonal moving average error specification.

\texttt{sma} can be included in a \texttt{ls} or \texttt{tsls} specification to specify a multiplicative seasonal moving average term. A \texttt{sma(p)} term can be included in your equation specification to represent a seasonal moving average term of order \( p \). The lag polynomial used in estimation is the product of that specified by the \texttt{ma} terms and that specified by the \texttt{sma} terms. The purpose of the \texttt{sma} expression is to allow you to form the product of lag polynomials.

Examples

\texttt{ls tb3 c ma(1) ma(2) sma(4)}

TB3 is modeled as a second order moving average process with a multiplicative seasonal moving average term at lag four.

\texttt{tsls(z) sale c adv ma(1) sma(12) sma(24) @ c gdp}

In this two-stage least squares specification, the error term is a first order moving average process with multiplicative seasonal moving average terms at lags 12 and 24. The “\texttt{z}” option turns off backcasting.

Cross-references

See “ARIMA Theory” beginning on page 469 of the User’s Guide for details on ARMA and seasonal ARMA modeling.

See also \texttt{sar} (p. 525), \texttt{ar} (p. 521), and \texttt{ma} (p. 521).
Appendix D. Matrix and String Reference

The following is an alphabetical listing of the functions and commands used when working with the EViews matrix language. For a description of the EViews matrix language, see Chapter 3, “Matrix Language”, on page 23. For a quick summary of these entries, see “Matrix Function and Command Summary” on page 44.

### @cholesky

<table>
<thead>
<tr>
<th>Syntax:</th>
<th>@cholesky(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument:</td>
<td>sym, s</td>
</tr>
<tr>
<td>Return:</td>
<td>matrix</td>
</tr>
</tbody>
</table>

Returns a matrix containing the Cholesky factorization of \( s \). The Cholesky factorization finds the lower triangular matrix \( L \) such that \( LL' \) is equal to the symmetric source matrix. Example:

```r
matrix fact = @cholesky(s1)
matrix orig = fact*transpose(fact)
```

### colplace

<table>
<thead>
<tr>
<th>Syntax:</th>
<th>colplace(m, v, n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument 1:</td>
<td>matrix, m</td>
</tr>
<tr>
<td>Argument 2:</td>
<td>vector, v</td>
</tr>
<tr>
<td>Argument 3:</td>
<td>integer, n</td>
</tr>
</tbody>
</table>

Places the column vector \( v \) into the matrix \( m \) at column \( n \). The number of rows of \( m \) and \( v \) must match, and the destination column must already exist within \( m \). Example:

```r
colplace(m1,v1,3)
```

The third column of \( M1 \) will be set equal to the vector \( V1 \).

See also `rowplace` (p. 550).
### @columnextract

**Matrix Utility Function**

Syntax: \[@\text{columnextract}(m, n)\]

Argument 1: matrix or sym, \(m\)

Argument 2: integer, \(n\)

Return: vector

Extract a vector from column \(n\) of the matrix object \(m\), where \(m\) is a matrix or sym.

Example:

```plaintext```
vector \(v1 = @\text{columnextract}(m1,3)\)
```

Note that while you may extract the first column of a column vector, or any column of a row vector, the command is more easily executed using simple element or vector assignment in these cases.

See also @rowextract (p. 550).

### @columns

**Matrix Utility Function**

Syntax: \[@\text{columns}(o)\]

Argument: matrix, vector, rowvector, sym, scalar, or series, \(o\)

Return: integer

Returns the number of columns in the matrix object \(o\). Example:

```plaintext```
scalar \(sc2 = @\text{columns}(m1)\)
vec1(2) = @\text{columns}(s1)
```

See also @rows (p. 550).

### @cond

**Matrix Algebra Function**

Syntax: \[@\text{cond}(o, n)\]

Argument 1: matrix or sym, \(o\)

Argument 2: (optional) scalar \(n\)

Return: scalar
Returns the condition number of a square matrix or sym, $o$. If no norm is specified, the infinity norm is used to determine the condition number. The condition number is the product of the norm of the matrix divided by the norm of the inverse. Possible norms are “-1” for the infinity norm, “0” for the Frobenius norm, and an integer “n” for the $L^n$ norm.

Example:

```plaintext
scalar sc1 = @cond(m1)
matl(1,4) = @cond(s1,2)
```

<table>
<thead>
<tr>
<th>Syntax:</th>
<th>$@convert(o, smp)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument 1:</td>
<td>series or group, $o$</td>
</tr>
<tr>
<td>Argument 2:</td>
<td>(optional) sample, $smp$</td>
</tr>
<tr>
<td>Return:</td>
<td>vector or matrix</td>
</tr>
</tbody>
</table>

If $o$ is a series, $@convert$ returns a vector from the values of $o$ using the optional sample $smp$ or the current workfile sample. If any observation has the value “NA”, the observation will be omitted from the vector. Examples:

```plaintext
vector v2 = @convert(ser1)
vector v3 = @convert(ser2, smp1)
```

If $o$ is a group, $@convert$ returns a matrix from the values of $o$ using the optional sample object $smp$ or the current workfile sample. The series in $o$ are placed in the columns of the resulting matrix in the order they appear in the group spreadsheet. If any of the series for a given observation, has the value “NA”, the observation will be omitted for all series. For example,

```plaintext
matrix ml=@convert(grp1)
matrix m2=@convert(grp1, smp1)
```

For a conversion method that preserves NAs, see $stomna$ (p. 552).
### @cor

**Syntax:** `@cor(v1, v2)`  
**Argument 1:** vector, rowvector, or series, `v1`  
**Argument 2:** vector, rowvector, or series, `v2`  
**Return:** scalar

If used with two vector or series objects, `v1` and `v2`, `@cor` returns the correlation between the two vectors or series. Examples:

```plaintext
scalar sc1 = @cor(v1, v2)  
s1(1,2) = @cor(v1, r1)
```

If used with a matrix object or group, `o`, `@cor` calculates the correlation matrix between the columns of the matrix object.

```plaintext
scalar sc2 = @cor(v1, v2)  
mat3(4,2) = 100*@cor(r1, v1)
```

For series and group calculations, EViews will use the current workfile sample. See also `@cov` (p. 530).

### @cov

**Syntax:** `@cov(v1, v2)`  
**Argument 1:** vector, rowvector, or series, `v1`  
**Argument 2:** vector, rowvector, or series, `v2`  
**Return:** scalar

If used with two vector or series objects, `v1` and `v2`, `@cov` returns the covariance between the two vectors or series. Examples:
scalar sc1 = @cov(v1, v2)
s1(1,2) = @cov(v1, r1)

If used with a matrix object or group, o, @cov calculates the covariance matrix between the columns of the matrix object.

!1 = @cov(v1, v2)
mat3(4,2) = 100*@cov(r1, v1)

For series and group calculations, EViews will use the current workfile sample. See also @cor (p. 530).

---

@datestr

String Function

Syntax: @datestr(d[, fmt])

Argument 1: date number, d
Argument 2: date format string, fmt
Return: string

Convert the numeric date value, d, into a string representation of a date, str, using the optional format string fmt.

Example:

@datestr(730088, "mm/dd/yy")
will return “12/1/99”,

@datestr(730088, "DD/mm/yyyy")
will return “01/12/1999”, and

@datestr(730088, "Month dd, yyyy")
will return “December 1, 1999”, and

@datestr(730088, "w")
will produce the string “3”, representing the weekday number for December 1, 1999. See “Dates” on page 875 of the User’s Guide for additional details on date numbers and date format strings.

See also @strdate (p. 553).
**@dateval**  
**String Function**

Syntax: \( \text{@dateval}(\text{str1}, [\text{fmt}]) \)

Argument 1: string, \( \text{str1} \)

Argument 2: date format string, \( \text{fmt} \)

Return: date number

Convert the string representation of a date, \( \text{str} \), into a date number using the optional format string \( \text{fmt} \).

Example:

\[
\text{@dateval}("12/1/1999", "mm/dd/yyyy")
\]

will return the date number for December 1, 1999 (730088) while

\[
\text{@dateval}("12/1/1999", "dd/mm/yyyy")
\]

will return the date number for January 12, 1999 (729765).

See “Dates” on page 875 of the User’s Guide for additional details on date numbers and date format strings.

**@det**  
**Matrix Algebra Function**

Syntax: \( \text{@det}(m) \)

Argument: matrix or sym, \( m \)

Return: scalar

Calculate the determinant of the square matrix or sym, \( m \). The determinant is nonzero for a nonsingular matrix and 0 for a singular matrix. Example:

\[
\text{scalar sc1 = @det(m1)}
\]
\[
\text{vec4(2) = @det(s2)}
\]

See also @rank (p. 547).
**@dtoo**  
String Function

Syntax: \(@dtoo(\text{str})\)  
Argument: string, \(\text{str}\)  
Return: integer

Date-TO-Observation. Returns the observation number corresponding to the date contained in the string. The observation number is relative to the start of the current workfile range, not the current sample. Observation numbers may be used to select a particular element in a vector that has been converted from a series, provided that NAs are preserved (see \texttt{stomna} (p. 552)).

Examples:

```plaintext
scalar obnum = @dtoo("1994:01")
vec1(1) = gdp(@dtoo("1955:01")+10)
```

Suppose that the workfile contains quarterly data. Then the second example places the 1957:02 value of the GDP series in the first element of VEC1.

Note that @DTOO will generate an error if used in a panel structured workfile.

See also @otod (p. 546).

**@eigenvalues**  
Matrix Algebra Function

Syntax: \(@eigenvalues(\text{sym})\)  
Argument: sym, \(\text{sym}\)  
Return: vector

Returns a vector containing the eigenvalues of the sym. The eigenvalues are those scalars \(\lambda\) that satisfy \(Sx = \lambda x\). where \(S\) is the sym associated with the argument \(s\). Associated with each eigenvalue is an eigenvector (see @eigenvectors (p. 534)). The eigenvalues are arranged in ascending order.

Example:
vector v1 = @eigenvalues(s1)

@eigenvectors  | Matrix Algebra Function
--------------|---------------------------------------
Syntax:       | @eigenvectors(s)                     
Argument:     | sym, s                               
Return:       | matrix                               

Returns a square matrix, of the same dimension as the sym, whose columns are the eigenvectors of the source matrix. Each eigenvector \( v \) satisfies \( S v = n v \), where \( S \) is the symmetric matrix given by \( s \), and where \( n \) is the eigenvalue associated with the eigenvector \( v \). The eigenvalues are arranged in ascending order, and the columns of the eigenvector matrix correspond to the sorted eigenvalues. Example:

matrix m2 = @eigenvectors(s1)

See also the function @eigenvalues (p. 533).

@eqna        | String Function
-------------|-----------------------------------
Syntax:      | @eqna(str1, str2)                 
Argument 1:  | string, str1                      
Argument 2:  | string, str2                      
Return:      | integer                           

Tests for equality of \( str1 \) and \( str2 \), treating null strings as ordinary blank strings, and not as missing values. Strings which test as equal return a 1, and 0 otherwise.

Example:

@eqna("abc", "abc")
returns a 1, while

@eqna("", "def")
returns a 0.

See also @isempty (p. 540) and @neqna (p. 545).
@explode | Matrix Utility Function
--- | ---
Syntax: | @explode(s)
Argument: | sym, s
Return: | matrix

Creates a square matrix from a sym, s, by duplicating the lower triangle elements into the upper triangle. Example:

```matlab
diagonalmatrix m2 = @explode(s1)
```

See also @implode (p. 537).

@filledmatrix | Matrix Utility Function
--- | ---
Syntax: | @filledmatrix(n1, n2, n3)
Argument 1: | integer, n1
Argument 2: | integer, n2
Argument 3: | scalar, n3
Return: | matrix

Returns a matrix with n1 rows and n2 columns, where each element contains the value n3. Example:

```matlab
diagonalmatrix m2 = @filledmatrix(3,2,7)
```

creates a $3 \times 2$ matrix where each element is set to 7. See also fill (p. 267).

@filledrowvector | Matrix Utility Function
--- | ---
Syntax: | @filledrowvector(n1, n2)
Argument 1: | integer, n1
Argument 2: | scalar, n2
Return: | rowvector

Returns a rowvector of length n1, where each element contains the value n2. Example:

```matlab
rowvector r1 = @filledrowvector(3,1)
```

creates a 3 element rowvector where each element is set to 1. See also fill (p. 267).
### @filledsym

**Matrix Utility Function**

**Syntax:**  
@filledsym(n1, n2)

**Argument 1:** integer, \( n1 \)

**Argument 2:** scalar, \( n2 \)

**Return:** sym

Returns an \( n1 \times n1 \) sym, where each element contains \( n2 \). Example:

\[
\text{sym } s2 = \text{@filledsym}(3, 9)
\]

creates a \( 3 \times 3 \) sym where each element is set to 9. See also `fill` (p. 267).

### @filledvector

**Matrix Utility Function**

**Syntax:**  
@filledvector(n1, n2)

**Argument 1:** integer, \( n1 \)

**Argument 2:** scalar, \( n2 \)

**Return:** vector

Returns a vector of length \( n1 \), where each element contains the value \( n2 \). Example:

\[
\text{vector } r1 = \text{@filledvector}(5, 6)
\]

creates a 5 element column vector where each element is set to 6. See also `fill` (p. 267).

### @getmaindiagonal

**Matrix Utility Function**

**Syntax:**  
@getmaindiagonal(m)

**Argument:** matrix or sym, \( m \)

**Return:** vector

Returns a vector created from the main diagonal of the matrix or sym object. Example:

\[
\text{vector } v1 = \text{@getmaindiagonal}(m1) \\
\text{vector } v2 = \text{@getmaindiagonal}(s1)
\]
[@identity] Matrix Utility Function

Syntax: @identity(n)
Argument: integer, n
Return: matrix

Returns a square $n \times n$ identity matrix. Example:

```
matrix i1 = @identity(4)
```

[@implode] Matrix Utility Function

Syntax: @implode(m)
Argument: square matrix, m
Return: sym

Forms a sym by copying the lower triangle of a square input matrix, $m$. Where possible, you should use a sym in place of a matrix to take advantage of computational efficiencies. Be sure you know what you are doing if the original matrix is not symmetric—this function does not check for symmetry. Example:

```
sym s2 = @implode(m1)
```

See also @explode (p. 535).

[@inner] Matrix Algebra Function

Syntax: @inner(v1, v2, smp)
Argument 1: vector, rowvector, or series, $v1$
Argument 2: vector, rowvector, or series, $v2$
Argument 3: (optional) sample, $smp$
Return: scalar

Syntax: @inner(o, smp)
Argument 1: matrix object or group, $o$
Argument 2: (optional) sample, $smp$
Return: scalar
If used with two vectors, \( v_1 \) and \( v_2 \), \@inner \) returns the dot or inner product of the two vectors. Examples:

```plaintext
    scalar scl = @inner(v1, v2)
    s1(1,2) = @inner(v1, r1)
```

If used with two series, \@inner \) returns the inner product of the series using observations in the workfile sample. You may provide an optional sample.

If used with a matrix or sym, \( o \), \@inner \) returns the inner product, or moment matrix, \( o' o \). Each element of the result is the vector inner product of two columns of the source matrix. The size of the resulting sym is the number of columns in \( o \). Examples:

```plaintext
    scalar scl = @inner(v1)
    sym sym1 = @inner(m1)
```

If used with a group, \@inner \) returns the uncentered second moment matrix of the data in the group, \( g \), using the observations in the sample, \( smp \). If no sample is provided, the workfile sample is used. Examples:

```plaintext
    sym s2 = @inner(gr1)
    sym s3 = @inner(gr1, smp1)
```

See also \@outer \( (p. 547) \).

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@insert</td>
<td>Inserts the string ( str2 ) into the base string ( str1 ) at the position given by the integer ( n ).</td>
</tr>
</tbody>
</table>

**Syntax:** 
@insert(str1, str2, n)

**Argument 1:** string, \( str1 \)

**Argument 2:** string, \( str2 \)

**Argument 3:** integer, \( n \)

**Return:** string

Example:

```plaintext
    @insert("I believe it can be done", "not ", 16)
```

returns “I believe it cannot be done”.

See also \@replace \( (p. 548) \), \@instr \( (p. 539) \) and \@mid \( (p. 544) \).
@instr

**String Function**

Syntax: \@instr(str1, str2[, n])

Argument 1: string, \(str1\)
Argument 2: string, \(str2\)
Argument 3: integer, \(n\)
Return: string

Finds the starting position of the target string \(str2\) in the base string \(str1\). By default, the function returns the location of the first occurrence of \(str2\) in \(str1\). You may provide an optional integer \(n\) to change the occurrence. If the occurrence of the target string is not found, \@INSTR\ will return a 0.

Example:

\@instr("1.23415", "34")

return the value 4, since the substring “34” appears beginning in the fourth character of the base string.

See also \@mid (p. 544).

@inverse

**Matrix Algebra Function**

Syntax: \@inverse(m)

Argument: square matrix or sym, \(m\)
Return: matrix or sym

Returns the inverse of a square matrix object or sym. The inverse has the property that the product of the source matrix and its inverse is the identity matrix. The inverse of a matrix returns a matrix, while the inverse of a sym returns a sym. Note that inverting a sym is much faster than inverting a matrix.

Examples:

```
matrix m2 = @inverse(m1)
sym s2 = @inverse(s1)
sym s3 = @inverse(@implode(m2))
```

See @solvesystem (p. 551).
@isempty

<table>
<thead>
<tr>
<th>Syntax:</th>
<th>@isempty(str)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument:</td>
<td>string, str</td>
</tr>
<tr>
<td>Return:</td>
<td>integer</td>
</tr>
</tbody>
</table>

Tests for whether str is a blank string, returning a 1 if str is a null string, and 0 otherwise.

Example:

@isempty("1.23415")
returns a 0, while

@isempty("")
return the value 1.

See also @len, @length (p. 541).

@issingular

<table>
<thead>
<tr>
<th>Syntax:</th>
<th>@issingular(o)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument:</td>
<td>matrix or sym, o</td>
</tr>
<tr>
<td>Return:</td>
<td>integer</td>
</tr>
</tbody>
</table>

Returns “1” if the square matrix or sym, o, is singular, and “0” otherwise. A singular matrix has a determinant of 0, and cannot be inverted.

Example:

scalar sc1 = @issingular(m1)

@kronecker

<table>
<thead>
<tr>
<th>Syntax:</th>
<th>@kronecker(o1, o2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument 1:</td>
<td>matrix object, o1</td>
</tr>
<tr>
<td>Argument 2:</td>
<td>matrix object, o2</td>
</tr>
<tr>
<td>Return:</td>
<td>matrix</td>
</tr>
</tbody>
</table>
Calculates the Kronecker product of the two matrix objects, o1 and o2. The resulting matrix has a number of rows equal to the product of the numbers of rows of the two matrix objects and a number of columns equal to the product of the numbers of columns of the two matrix objects. The elements of the resulting matrix consist of submatrices consisting of one element of the first matrix object multiplied by the entire second matrix object.

Example:

\[
\text{matrix m3} = \text{@kronecker(m1,m2)}
\]

@left

<table>
<thead>
<tr>
<th>Syntax:</th>
<th>@left(str, n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument 1:</td>
<td>string, str</td>
</tr>
<tr>
<td>Argument 2:</td>
<td>integer, n</td>
</tr>
<tr>
<td>Return:</td>
<td>string</td>
</tr>
</tbody>
</table>

Returns a string containing n characters from the left end of str. If the string is shorter than n characters, this function returns all of the characters in the source string.

Example:

\[
\text{scalar sc1} = \text{@left("I did not do it",5)}
\]

returns “I did”.

See also @right (p. 549), @mid (p. 544).

@len, @length

<table>
<thead>
<tr>
<th>Syntax:</th>
<th>@len(str), @length(str)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument:</td>
<td>string, str</td>
</tr>
<tr>
<td>Return:</td>
<td>integer</td>
</tr>
</tbody>
</table>

Returns an integer value for the length of the string str.

Example:

\[
\text{@length("I did not do it")}
\]

returns the value 15.

See also @mid (p. 544).
### @lower (String Function)

**Syntax:**
```markdown
@lower(str)
```

**Argument:**
- string, `str`

**Return:**
- string

Returns the lowercase representation of the string `str`.

**Example:**

```markdown
@lower("I did NOT do it")
```

returns the string “i did not do it”.

See also [@upper](p. 557).

### @ltrim (String Function)

**Syntax:**
```markdown
@ltrim(str)
```

**Argument:**
- string, `str`

**Return:**
- string

Returns the string `str` with spaces trimmed from the left.

**Example:**

```markdown
@ltrim(" I doubt that I did it. ")
```

returns “I doubt that I did it. ”. Note that the spaces on the right remain.

See also [@rtrim](p. 551) and [@trim](p. 556).

### @makediagonal (Matrix Utility Function)

**Syntax:**
```markdown
@makediagonal(v, k)
```

**Argument 1:**
- vector or rowvector, `v`

**Argument 2:**
- (optional) integer, `k`

**Return:**
- matrix

Creates a square matrix with the specified vector or rowvector, `v`, in the `k`-th diagonal relative to the main diagonal, and zeroes off the diagonal. If no `k` value is provided or if `k` is set
to 0, the resulting matrix will have the same number of rows and columns as the length of \( v \), and will have \( v \) in the main diagonal. If a value for \( k \) is provided, the matrix has the same number of rows and columns as the number of elements in the vector plus \( k \), and will place \( v \) in the diagonal offset from the main by \( k \). Example:

\[
\begin{align*}
\text{matrix } m1 &= @\text{makediagonal}(v1) \\
\text{matrix } m2 &= @\text{makediagonal}(v1,1) \\
\text{matrix } m4 &= @\text{makediagonal}(r1,-3)
\end{align*}
\]

M1 will contain \( V1 \) in the main diagonal; M2 will contain \( V1 \) in the diagonal immediately above the main diagonal; M4 will contain \( R1 \) in the diagonal 3 positions below the main diagonal. Using the optional \( k \) parameter may be useful in creating covariance matrices for AR models. For example, you can create an AR(1) correlation matrix by issuing the commands:

\[
\begin{align*}
\text{matrix}(10,10) m1 \\
\text{vector}(9) \ rho = .3 \\
m1 &= @\text{makediagonal}(\rho,-1) + @\text{makediagonal}(\rho,+1) \\
m1 &= m1 + @\text{identity}(10)
\end{align*}
\]

### matplace

**Matrix Utility Command**

<table>
<thead>
<tr>
<th>Syntax:</th>
<th>matplace(m1, m2, n1, n2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument 1:</td>
<td>matrix, m1</td>
</tr>
<tr>
<td>Argument 2:</td>
<td>matrix, m2</td>
</tr>
<tr>
<td>Argument 3:</td>
<td>integer, n1</td>
</tr>
<tr>
<td>Argument 4:</td>
<td>integer, n2</td>
</tr>
</tbody>
</table>

Places the matrix object \( m2 \) into \( m1 \) at row \( n1 \) and column \( n2 \). The sizes of the two matrices do not matter, as long as \( m1 \) is large enough to contain all of \( m2 \) with the upper left cell of \( m2 \) placed at row \( n1 \) and column \( n2 \).

Example:

\[
\begin{align*}
\text{matrix}(100,5) m1 \\
\text{matrix}(100,2) m2 \\
\text{matplace}(m1,m2,1,1)
\end{align*}
\]
String Function

@mid
Syntax: @mid(str, n1[, n2])
Argument 1: string, str
Argument 2: integer, n1
Argument 3: integer, n2
Return: string

Returns n2 characters from str, starting at location n1 and continuing to the right. If you omit n2, it will return all of the remaining characters in the string.

Example:
%1 = @mid("I doubt it", 9, 2)
%2 = @mid("I doubt it", 9)

See also @left (p. 541) and @right (p. 549).

mtos
Matrix Utility Command

Convert matrix to a series or group. Fills a series or group with the data from a vector or matrix.

Syntax
Vector Proc:    mtos(vector, series[, sample])
Matrix Proc:    mtos(matrix, group[, sample])

Matrix-TO-Series Object. Include the vector or matrix name in parentheses, followed by a comma and then the series or group name. The number of included observations in the sample must match the row size of the matrix to be converted. If no sample is provided, the matrix is written into the series using the current workfile sample.

Examples
The command

mtos(mom, gr1)

converts the first column of the matrix MOM to the first series in the group GR1, the second column of MOM to the second series in GR1, and so on. The current workfile sample length must match the row length of the matrix MOM. If GR1 is an existing group object, the number of series in GR1 must match the number of columns of MOM. If a group object
named GR1 does not exist, EViews creates GR1 with the first series named SER1, the second series named SER2, and so on.

```plaintext
series col1
series col2
group g1 col1 col2
sample s1 1951 1990
mtos(m1,g1,s1)
```

The first two lines declare series objects, the third line declares a group object, the fourth line declares a sample object, and the fifth line converts the columns of the matrix M1 to series in group G1 using sample S1. This command will generate an error if M1 is not a 40 × 2 matrix.

**Cross-references**

See Chapter 3, “Matrix Language”, on page 23 for further discussions and examples of matrices.

See also `stom` (p. 551) and `stomna` (p. 552).

---

### Syntax: `@neqna` String Function

<table>
<thead>
<tr>
<th>Syntax</th>
<th>@neqna(str1, str2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument 1:</td>
<td>string, str1</td>
</tr>
<tr>
<td>Argument 2:</td>
<td>string, str2</td>
</tr>
<tr>
<td>Return:</td>
<td>integer</td>
</tr>
</tbody>
</table>

Tests for inequality of `str1` and `str2`, treating null strings as ordinary blank strings, and not as missing values. Strings which test as equal return a 0, and 1 otherwise.

**Example:**

```plaintext
@neqna("abc", "abc")
```

returns a 0, while

```plaintext
@eqna("", "def")
```

returns a 1.

See also `@isempty` (p. 540) and `@eqna` (p. 534).
### @norm

**Syntax:** \( @\text{norm}(o, n) \)

**Argument 1:** matrix, vector, rowvector, sym, scalar, or series, \( o \)

**Argument 2:** (optional) integer, \( n \)

**Return:** scalar

Returns the value of the norm of any matrix object, \( o \). Possible choices for the norm type \( n \) include \( \text{“-1”} \) for the infinity norm, \( \text{“0”} \) for the Frobenius norm, and an integer \( \text{“n”} \) for the \( L^n \) norm. If no norm type is provided, this function returns the infinity norm.

**Examples:**

```
scalar sc1 = @norm(m1)
scalar sc2 = @norm(v1,1)
```

---

### @otod

**Syntax:** \( @\text{otod}(n) \)

**Argument:** integer, \( n \)

**Return:** string

Observation-TO-Date. Returns a string containing the date or observation corresponding to observation number \( n \), counted from the beginning of the current workfile. For example, consider the string assignment

```
%1 = @otod(5)
```

For an annual workfile dated 1991–2000, \%1 will contain the string “1995”. For a quarterly workfile dated 1970:1–2000:4, \%1 will contain the string “1971:1”. Note that \( @\text{otod}(1) \) returns the date or observation label for the start of the workfile.

See also \( @\text{dtod} \) (p. 533).
**@outer**

**Matrix Algebra Function**

Syntax:   
@outer(v1, v2)

Argument 1:  vector, rowvector, or series, v1
Argument 2:  vector, rowvector, or series, v2

Return:  matrix

Calculates the cross product of \( v1 \) and \( v2 \). Vectors may be either row or column vectors. The outer product is the product of \( v1 \) (treated as a column vector) and \( v2 \) (treated as a row vector), and is a square matrix of every possible product of the elements of the two inputs. Example:

\[
\begin{align*}
\text{matrix } m1 &= @outer(v1, v2) \\
\text{matrix } m4 &= @outer(r1, r2)
\end{align*}
\]

See also @inner (p. 537).

**@permute**

**Matrix Utility Function**

Syntax:   
@permute(m1)

Input:  matrix m1

Return:  matrix

This function returns a matrix whose rows are randomly drawn without replacement from rows of the input matrix \( m1 \). The output matrix has the same size as the input matrix.

\[
\text{matrix } xp = @permute(x)
\]

To draw with replacement from rows of a matrix, use @resample (p. 548).

**@rank**

**Matrix Algebra Function**

Syntax:   
@rank(o, n)

Argument 1:  vector, rowvector, matrix, sym, or series, \( o \)
Argument 2:  (optional) integer, \( n \)

Return:  integer

Returns the rank of the matrix object \( o \). The rank is calculated by counting the number of singular values of the matrix which are smaller in absolute value than the tolerance, which is given by the argument \( n \). If \( n \) is not provided, EViews uses the value given by the largest
dimension of the matrix multiplied by the norm of the matrix multiplied by machine epsilon (the smallest representable number).

```plaintext
scalar rank1 = @rank(m1)
scalar rank2 = @rank(s1)
```

See also @svd (p. 555).

### @replace

<table>
<thead>
<tr>
<th>Syntax:</th>
<th>@replace(str1, str2, str3[,n])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument 1:</td>
<td>string, str1</td>
</tr>
<tr>
<td>Argument 2:</td>
<td>string, str2</td>
</tr>
<tr>
<td>Argument 3:</td>
<td>string, str3</td>
</tr>
<tr>
<td>Argument 4:</td>
<td>integer, n</td>
</tr>
<tr>
<td>Return:</td>
<td>string</td>
</tr>
</tbody>
</table>

Returns the base string \(str1\), with the replacement \(str3\) substituted for the target string \(str2\). By default, all occurrences of \(str2\) will be replaced, but you may provide an optional integer \(n\) to specify the number of occurrences to be replaced.

Example:

```plaintext
@replace("Do you think that you can do it?", "you", "I")
```

returns the string “Do I think that I can do it?”, while

```plaintext
@replace("Do you think that you can do it?", "you", "I", 1)
```

returns “Do I think that you can do it?".

See also @insert (p. 538) and @mid (p. 544).

### @resample

<table>
<thead>
<tr>
<th>Syntax:</th>
<th>@resample(m1, n2, n3, v4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input 1:</td>
<td>matrix m1</td>
</tr>
<tr>
<td>Input 2:</td>
<td>(optional) integer n2</td>
</tr>
<tr>
<td>Input 3:</td>
<td>(optional) positive integer n3</td>
</tr>
<tr>
<td>Input 4:</td>
<td>(optional) vector v4</td>
</tr>
<tr>
<td>Output:</td>
<td>matrix</td>
</tr>
</tbody>
</table>
This function returns a matrix whose rows are randomly drawn with replacement from rows of the input matrix.

\( n_2 \) represents the number of “extra” rows to be drawn from the matrix. If the input matrix has \( r \) rows and \( c \) columns, the output matrix will have \( r + n_2 \) rows and \( c \) columns. By default, \( n_2 = 0 \).

\( n_3 \) represents the block size for the resample procedure. If you specify \( n_3 > 1 \), then blocks of consecutive rows of length \( n_3 \) will be drawn with replacement from the first \( r - n_3 + 1 \) rows of the input matrix.

You may provide a name for the vector \( v_4 \) to be used for weighted resampling. The weighting vector must have length \( r \) and all elements must be non-missing and non-negative. If you provide a weighting vector, each row of the input matrix will be drawn with probability proportional to the weights in the corresponding row of the weighting vector. (The weights need not sum to 1. EViews will automatically normalize the weights).

\[
\text{matrix } \mathbf{x}_b = \text{@bootstrap}(\mathbf{x})
\]

To draw without replacement from rows of a matrix, use \text{@permute} (p. 547).

### \text{@right}

<table>
<thead>
<tr>
<th>Syntax:</th>
<th>\text{@right}(str, n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument 1:</td>
<td>string, ( \text{str} )</td>
</tr>
<tr>
<td>Argument 2:</td>
<td>integer, ( n )</td>
</tr>
<tr>
<td>Return:</td>
<td>same as source</td>
</tr>
</tbody>
</table>

Returns a string containing \( n \) characters from the right end of \( \text{str} \). If the source is shorter than \( n \), the entire string is returned. Example:

\[
\%1 = \text{@right}("I doubt it", 8)
\]

returns the string “doubt it”.

See also \text{@left} (p. 541), \text{@mid} (p. 544).
### @rowextract

<table>
<thead>
<tr>
<th>Syntax</th>
<th>@rowextract(m, n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument 1</td>
<td>matrix or sym, m</td>
</tr>
<tr>
<td>Argument 2</td>
<td>integer, n</td>
</tr>
<tr>
<td>Return</td>
<td>rowvector</td>
</tr>
</tbody>
</table>

Extracts a rowvector from row \( n \) of the matrix object \( m \). Example:

\[ \text{rowvector } r1 = @\text{rowextract}(m1,3) \]

See also @columnextract (p. 528).

### rowplace

<table>
<thead>
<tr>
<th>Syntax</th>
<th>rowplace(m, r, n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument 1</td>
<td>matrix, m</td>
</tr>
<tr>
<td>Argument 2</td>
<td>rowvector, r</td>
</tr>
<tr>
<td>Argument 3</td>
<td>integer</td>
</tr>
</tbody>
</table>

Places the rowvector \( r \) into the matrix \( m \) at row \( n \). The number of columns in \( m \) and \( r \) must match, and row \( n \) must exist within \( m \). Example:

\[ \text{rowplace}(m1,r1,4) \]

See also colplace (p. 527).

### @rows

<table>
<thead>
<tr>
<th>Syntax</th>
<th>@rows(o)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>matrix, vector, rowvector, sym, series, or group, o</td>
</tr>
<tr>
<td>Return</td>
<td>scalar</td>
</tr>
</tbody>
</table>

Returns the number of rows in the matrix object, \( o \).

Example:

\[ \text{scalar } sc1=@\text{rows}(m1) \]
\[ \text{scalar } size=@\text{rows}(m1)*@\text{columns}(m1) \]
For series and groups @rows (p. 550) returns the number of observations in the workfile range. See also @columns (p. 528).

@rtrim | String Function
---|---
Syntax: @rtrim(str)
Argument: string, str
Return: string

Returns the string str with spaces trimmed from the right.

Example:
    @rtrim(" I doubt that I did it. ")
returns the string “I doubt that I did it.”. Note that the spaces on the left remain.

See also @ltrim (p. 542) and @trim (p. 556).

@solvesystem | Matrix Algebra Function
---|---
Syntax: @solvesystem(o, v)
Argument 1: matrix or sym, o
Argument 2: vector, v
Return: vector

Returns the vector x that solves the equation $Mx = p$ where the matrix or sym $M$ is given by the argument o. Example:
    vector v2 = @solvesystem(m1, v1)
See also @inverse (p. 539).

stom | Matrix Utility Command
---|---
Syntax: stom(o1, o2, smp)
Argument 1: series or group, o1
Argument 2: vector or matrix, o2
Argument 3: (optional) sample smp
Series-TO-Matrix Object. If \( o1 \) is a series, stom fills the vector \( o2 \) with data from the \( o1 \) using the optional sample object \( smp \) or the workfile sample. \( o2 \) will be resized accordingly. If any observation has the value “NA”, the observation will be omitted from the vector. Example:

\[
\text{stom}(\text{ser1,v1}) \\
\text{stom}(\text{ser1,v2, smp1})
\]

If \( o1 \) is a group, stom fills the matrix \( o2 \) with data from \( o1 \) using the optional sample object \( smp \) or the workfile sample. \( o2 \) will be resized accordingly. The series in \( o1 \) are placed in the columns of \( o2 \) in the order they appear in the group spreadsheet. If any of the series in the group has the value “NA” for a given observation, the observation will be omitted for all series. Example:

\[
\text{stom}(\text{grp1,m1}) \\
\text{stom}(\text{grp1,m2, smp1})
\]

For a conversion method that preserves NAs, see \texttt{stomna} (p. 552).

<table>
<thead>
<tr>
<th>stomna</th>
<th>Matrix Utility Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>stomna(o1, o2, smp)</td>
</tr>
<tr>
<td>Argument 1:</td>
<td>series or group, ( o1 )</td>
</tr>
<tr>
<td>Argument 2:</td>
<td>vector or matrix, ( o2 )</td>
</tr>
<tr>
<td>Argument 3:</td>
<td>(optional) sample ( smp )</td>
</tr>
</tbody>
</table>

Series-TO-Matrix Object with NAs. If \( o1 \) is a series, stom fills the vector \( o2 \) with data from \( o1 \) using the optional sample object \( smp \) or the workfile sample. \( o2 \) will be resized accordingly. All “NA” values in the series will be assigned to the corresponding vector elements. Example:

\[
\text{stom}(\text{ser1,v1}) \\
\text{stom}(\text{ser1,v2, smp1})
\]

If \( o1 \) is a group, stom fills the matrix \( o2 \) with data from \( o1 \) using the optional sample object \( smp \) or the workfile sample. \( o2 \) will be resized accordingly. The series in \( o1 \) are placed in the columns of \( o2 \) in the order they appear in the group spreadsheet. All NAs will be assigned to the corresponding matrix elements. Example:

\[
\text{stomna}(\text{grp1,m1}) \\
\text{stomna}(\text{grp1,m2, smp1})
\]

For conversion methods that automatically remove observations with NAs, see \texttt{@convert} (p. 529) and \texttt{stom} (p. 551).
### @str

<table>
<thead>
<tr>
<th>Syntax</th>
<th>@str(d,[fmt])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument 1:</td>
<td>scalar, d</td>
</tr>
<tr>
<td>Argument 2:</td>
<td>numeric format string, fmt</td>
</tr>
<tr>
<td>Return:</td>
<td>string</td>
</tr>
</tbody>
</table>

Returns a string representing the given number. You may provide an optional format string.

Example:

```plaintext
%1 = @str(15.23456)
alpha newa = @str(32.56)
```

See also @val (p. 557).

### @strdate

<table>
<thead>
<tr>
<th>Syntax</th>
<th>@strdate(fmt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument:</td>
<td>date format string, fmt</td>
</tr>
<tr>
<td>Return:</td>
<td>string corresponding to each element in workfile</td>
</tr>
</tbody>
</table>

Return the set of workfile row dates as strings, using the date format string fmt.

See also @datestr (p. 531) and @strnow (p. 554).

### @strlen

<table>
<thead>
<tr>
<th>Syntax</th>
<th>@strlen(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument:</td>
<td>string, s</td>
</tr>
<tr>
<td>Return:</td>
<td>number n</td>
</tr>
</tbody>
</table>

Same as @len, @length (p. 541).
**Syntax:** @strnow(fmt)

**Argument:** date format string, `fmt`

**Return:** string

Returns a string representation of the current date number (at the moment the function is evaluated) using the date format string, `fmt`.

**Example:**

```plaintext
@strnow("DD/mm/yyyy")
```

returns the date associated with the current time in string form with 2-digit days, months, and 4-digit years separated by a slash, “24/12/2003”.

See also @strdate (p. 553) and @datestr (p. 531).

---

**Syntax:** @subextract(o, n1, n2, n3, n4)

**Argument 1:** vector, rowvector, matrix or sym, `o`

**Argument 2:** integer, `n1`

**Argument 3:** integer, `n2`

**Argument 4:** (optional) integer, `n3`

**Argument 5:** (optional) integer, `n4`

**Return:** matrix

Returns a submatrix of a specified matrix, `o`. `n1` is the row and `n2` is the column of the upper left element to be extracted. The optional arguments `n3` and `n4` provide the row and column location of the lower right corner of the matrix. Unless `n3` and `n4` are provided this function returns a matrix containing all of the elements below and to the right of the starting element.

**Examples:**

```plaintext
matrix m2 = @subextract(m1, 5, 9, 6, 11)
matrix m2 = @subextract(m1, 5, 9)
```
**@svd**

**Matrix Algebra Function**

Syntax: \( \text{@svd}(m1, v1, m2) \)

Argument 1: matrix or sym, \( m1 \)

Argument 2: vector, \( v1 \)

Argument 3: matrix or sym, \( m2 \)

Return: matrix

Performs a singular value decomposition of the matrix \( m1 \). The matrix \( U \) is returned by the function, the vector \( v1 \) will be filled (resized if necessary) with the singular values and the matrix \( m2 \) will be assigned (resized if necessary) the other matrix, \( V \), of the decomposition. The singular value decomposition satisfies

\[
m1 = U W V^\top
\]

\[
U^\top U = V V^\top = I
\]

where \( W \) is a diagonal matrix with the singular values along the diagonal. Singular values close to zero indicate that the matrix may not be of full rank. See the \text{@rank} (p. 547) function for a related discussion.

Examples:

```plaintext
matrix m2
vector v1
matrix m3 = @svd(m1,v1,m2)
```

**@trace**

**Matrix Algebra Function**

Syntax: \( \text{@trace}(m) \)

Argument: matrix or sym, \( m \)

Return: scalar

Returns the trace (the sum of the diagonal elements) of a square matrix or sym, \( m \). Example:
scalar scl = @trace(m1)

**@transpose**

**Matrix Algebra Function**

Syntax: @transpose(o)
Argument: matrix, vector, rowvector, or sym, o
Return: matrix, rowvector, vector, or sym

Forms the transpose of a matrix object, o. o may be a vector, rowvector, matrix, or a sym. The result is a matrix object with a number of rows equal to the number of columns in the original matrix and number of columns equal to the number of rows in the original matrix. This function is an identity function for a sym, since a sym by definition is equal to its transpose. Example:

```plaintext
definition: matrix m2 = @transpose(m1)
definition: rowvector r2 = @transpose(v1)
```

**@trim**

**String Function**

Syntax: @trim(str)
Argument: string, str
Return: string

Returns the string str with spaces trimmed from the both the left and the right.

Example:

```plaintext
@rtrim(" I doubt that I did it. ")
```
returns the string “I doubt that I did it.”.

See also @ltrim (p. 542) and @rtrim (p. 551).

**@unitvector**

**Matrix Utility Function**

Syntax: @unitvector(n1, n2)
Argument 1: integer, n1
Argument 2: integer, n2
Return: vector

Creates an n1 element vector with a “1” in the n2-th element, and “0” elsewhere. Example:
vec v1 = @unitvector(8, 5)

creates an 8 element vector with a “1” in the fifth element and “0” for the other 7 elements. Note: if you wish to create an $nI$ element vector of ones, you should use a declaration statement of the form

vector(n1) v1=1

---

<table>
<thead>
<tr>
<th>@upper</th>
<th>String Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax: @upper(str)</td>
<td>Argument: string, str</td>
</tr>
<tr>
<td>Return: string</td>
<td></td>
</tr>
</tbody>
</table>

Returns the uppercase representation of the string str.

Example:

@length("I did not do it")

returns the string “I DID NOT DO IT”.

See also @lower (p. 542).

---

<table>
<thead>
<tr>
<th>@val</th>
<th>String Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax: @val(str[, fmt])</td>
<td>Argument 1: string, str</td>
</tr>
<tr>
<td>Argument 2: numeric format string, fmt</td>
<td>Return: scalar</td>
</tr>
</tbody>
</table>

Returns the number that a string str represents. You may provide an optional numeric format string fmt. By default, EViews will attempt to interpret the string as a number using standard formats; if this is not possible, the function returns “NA”.

Example:

scalar sc1 = @val("17.4648")
scalar sc2 = @val(tabl(3,4))
scalar sc3 = @val("-234.35")

See also @str (p. 553).
### @vec

**Syntax:** `@vec(o)`  
**Argument:** matrix, sym, `o`  
**Return:** vector

Creates a vector from the columns of the given matrix stacked one on top of each other. The vector will have the same number of elements as the source matrix. Example:

```
vector v1 = @vec(m1)
```

### @vech

**Syntax:** `@vech(o)`  
**Argument:** matrix, sym, `o`  
**Return:** vector

Creates a vector from the columns of the lower triangle of the source square matrix `o` stacked on top of each another. The vector has the same number of elements as the source matrix has in its lower triangle. Example:

```
vector v1 = @vech(m1)
```
The following reference is an alphabetical listing of the program statements and support functions used by the EViews programming language.

For details on the EViews programming language, see Chapter 6, “EViews Programming”, on page 83. For a quick summary of these entries, see “Programming Summary” on page 115.

### call

Call a subroutine within a program.

The call statement is used to call a subroutine within a program.

**Cross-references**

See “Calling Subroutines” on page 110. See also subroutine (p. 568), endsub (p. 560).

<table>
<thead>
<tr>
<th>@date</th>
<th>Support Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>@date</td>
</tr>
<tr>
<td>Return:</td>
<td>string</td>
</tr>
</tbody>
</table>

Returns a string containing the current date in “mm/dd/yy” format.

**Examples**

%y = @date

assigns a string of the form “10/10/00”.

**Cross-references**

See also @time (p. 569).
**else** | Program Statement
---|---

ELSE clause of IF statement in a program.

Starts a sequence of commands to be executed when the IF condition is false. The `else` keyword must be terminated with an `endif`.

**Syntax**

```plaintext
if [condition] then
    [commands to be executed if condition is true]
else
    [commands to be executed if condition is false]
endif
```

**Cross-references**

See “IF Statements” on page 99. See also, `if` (p. 562), `endif` (p. 560), `then` (p. 569).

---

**endif** | Program Statement
---|---

End of IF statement. Marks the end of an IF, or an IF-ELSE statement.

**Syntax**

```plaintext
if [condition] then
    [commands if condition true]
endif
```

**Cross-references**

See “IF Statements” on page 99. See also, `if` (p. 562), `endif` (p. 560), `then` (p. 569).

---

**endsub** | Program Statement
---|---

Mark the end of a subroutine.

**Syntax**

```plaintext
subroutine name(arguments)
    commands
endsub
```
endsub

Cross-references
See “Defining Subroutines” beginning on page 108. See also, subroutine (p. 568), return (p. 566).

<table>
<thead>
<tr>
<th>Support Function</th>
<th>@errorcount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>@errorcount</td>
</tr>
<tr>
<td>Argument:</td>
<td>none</td>
</tr>
<tr>
<td>Return:</td>
<td>integer</td>
</tr>
</tbody>
</table>

Number of errors encountered. Returns a scalar containing the number of errors encountered during program execution.

<table>
<thead>
<tr>
<th>Support Function</th>
<th>@evpath</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>@evpath</td>
</tr>
<tr>
<td>Return:</td>
<td>string</td>
</tr>
</tbody>
</table>

Returns a string containing the directory path for the EViews executable.

Examples
If your currently executing copy of EViews is installed in “D:\EVIEWS”, then

\[ %y = @evpath \]

assigns a string of the form “D:\EVIEWS”.

Cross-references
See also cd (p. 213) and @temppath (p. 568).

<table>
<thead>
<tr>
<th>Program Statement</th>
<th>exitloop</th>
</tr>
</thead>
</table>

Exit from current loop in programs.

exitloop causes the program to break out of the current FOR or WHILE loop.

Syntax
Command: exitloop
Examples
for !i=1 to 107
    if !i>6 then exitloop
next

Cross-references
See “The FOR Loop” on page 101. See also, stop (p. 568), return (p. 566), for (p. 562), next (p. 563), step (p. 567).

<table>
<thead>
<tr>
<th>for</th>
<th>Program Statement</th>
</tr>
</thead>
</table>

FOR loop in a program.
The for statement is the beginning of a FOR...NEXT loop in a program.

Syntax
for counter=start to end [step stepsize]
    commands
next

Cross-references
See “The FOR Loop” on page 101. See also, exitloop (p. 561), next (p. 563), step (p. 567).

<table>
<thead>
<tr>
<th>if</th>
<th>Program Statement</th>
</tr>
</thead>
</table>

IF statement in a program.
The if statement marks the beginning of a condition and commands to be executed if the statement is true. The statement must be terminated with the beginning of an ELSE clause, or an endif.

Syntax
if [condition] then
    [commands if condition true]
endif
Cross-references

See “IF Statements” on page 99. See also else (p. 560), endif (p. 560), then (p. 569).

### include

<table>
<thead>
<tr>
<th>Program Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include another file in a program.</td>
</tr>
<tr>
<td>The <code>include</code> statement is used to include the contents of another file in a program file.</td>
</tr>
<tr>
<td>Syntax</td>
</tr>
<tr>
<td><code>include filename</code></td>
</tr>
<tr>
<td>Cross-references</td>
</tr>
<tr>
<td>See “Multiple Program Files” on page 107. See also call (p. 559).</td>
</tr>
</tbody>
</table>

### @isobject

<table>
<thead>
<tr>
<th>Support Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax: @isobject(str)</td>
</tr>
<tr>
<td>Argument: string, str</td>
</tr>
<tr>
<td>Return: integer</td>
</tr>
<tr>
<td>Check for an object’s existence. Returns a “1” if the object exists in the current workfile, and a “0” if it does not exist.</td>
</tr>
</tbody>
</table>

### next

<table>
<thead>
<tr>
<th>Program Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>End of FOR loop. next marks the end of a FOR loop in a program.</td>
</tr>
<tr>
<td>Syntax</td>
</tr>
<tr>
<td><code>for [conditions of the FOR loop]</code></td>
</tr>
<tr>
<td><code>[commands]</code></td>
</tr>
<tr>
<td><code>next</code></td>
</tr>
<tr>
<td>Cross-references</td>
</tr>
<tr>
<td>See “The FOR Loop” beginning on page 101. See also, exitloop (p. 561), for (p. 562), step (p. 567).</td>
</tr>
</tbody>
</table>
@obsrange
Support Function

Syntax: @obsrange
Return: number

returns number of observations in the current active workfile range (0 if no workfile in memory)

Examples
!z = @obsrange
will place the number of observations in the workfile range in the replacement variable !Z.

Cross-references
See also @obssmpl (p. 564).

@obssmpl
Support Function

Syntax: @obssmpl
Return: number

returns number of observations in the current active workfile sample (0 if no workfile in memory)

Examples
!z = @obssmpl
will place the number of observations in the sample in the replacement variable !Z.

Cross-references
See also @obsrange (p. 564).

open
Command

Open a file. Opens a workfile, database, program file, or ASCII text file.

See open (p. 342).
output  Command

Redirects printer output or display estimation output.

See output (p. 347).

poff  Program Statement

Turn off automatic printing in programs.

poff turns off automatic printing of all output. In programs, poff is used in conjunction with pon to control automatic printing; these commands have no effect in interactive use.

Syntax
   Command:      poff

Cross-references
See “Print Setup” on page 903 for a discussion of printer control.
See also pon (p. 565).

pon  Program Statement

Turn on automatic printing in programs.

pon instructs EViews to send all statistical and data display output to the printer (or the redirected printer destination; see output (p. 347)). It is equivalent to including the “p” option in all commands that generate output. pon and poff only work in programs; they have no effect in interactive use.

Syntax
   Command:      pon

Cross-references
See “Print Setup” on page 903 of the User’s Guide for a discussion of printer control.
See also poff (p. 565).
Create a program.

See program (p. 374).

Exit subroutine.

The return statement forces an exit from a subroutine within a program. A common use of return is to exit from the subroutine if an unanticipated error has occurred.

Syntax

\[
\text{if } \{\text{condition}\} \text{ then }
\]

\[
\text{return}
\]

\[
\text{endif}
\]

Cross-references

See “Subroutines” beginning on page 108. See also exitloop (p. 561), stop (p. 568).

Run a program. The run command executes a program. The program may be located in memory or stored in a program file on disk.

See run (p. 391).

Send a text message to the EViews statusline.

Syntax

\[
\text{for } !i = 1 \text{ to } 10
\]

\[
\text{statusline Iteration } !i
\]

\[
\text{next}
\]
Step size of a FOR loop.

Syntax

```plaintext
for !i = a to b step n
    [commands]
next
```

*step* may be used in a FOR loop to specify the size of the step in the looping variable. If no step is provided, FOR assumes a step of “+1”.

If a given step exceeds the end value *b* in the FOR loop specification, the contents of the loop will not be executed.

**Examples**

```plaintext
for !j = 5 to 1 step -1
    series x = nrnd*!j
next
```

Repetitively executes the commands in the loop with the control variable !j set to “5”, “4”, “3”, “2”, “1”.

```plaintext
for !j = 0 to 10 step 3
    series z = z/!j
next
```

Loops the commands with the control variable !j set to “0”, “3”, “6”, and “9”.

You should take care when using non-integer values for the stepsize since round-off error may yield unanticipated results. For example:

```plaintext
for !j = 0 to 1 step .01
    series w = !j
next
```

May stop before executing the loop for the value !j = 1 due to round-off error.

**Cross-references**

See “The FOR Loop” beginning on page 101. See also exitloop (p. 561), for (p. 562), next (p. 563).
stop  

Program Statement

Break out of program.

The `stop` command halts execution of a program. It has the same effect as hitting the F1 (break) key.

Syntax

Command:  

`stop`

Cross-references

See also, `exitloop` (p. 561), `return` (p. 566).

subroutine  

Program Statement

Declare a subroutine within a program.

The `subroutine` statement marks the start of a subroutine.

Syntax

```
subroutine name(arguments)  
  [commands]  
endsub
```

Cross-references

See “Subroutines” beginning on page 108. See also `endsub` (p. 560).

@temppath  

Support Function


Syntax:

`@temppath`

Return:  

string

Returns a string containing the directory path for the EViews temporary files as specified in the global options `File Locations`... menu.

Examples

If your currently executing copy of EViews puts temporary files in “D:\EVIEWS”, then

```
$y = @temppath
```
assigns a string of the form “D:\EVIWES”.

Cross-references
See also cd (p. 213) and @evpath (p. 561).

then

Part of IF statement.
then marks the beginning of commands to be executed if the condition given in the IF statement is satisfied.

Syntax
if [condition] then
    [commands if condition true]
endif

Cross-references
See “IF Statements” on page 99. See also, else (p. 560), endif (p. 560), if (p. 562).

@time

Syntax:  @time
Return:  string

Returns a string containing the current time in “hh:mm” format.

Examples
%y = @time
assigns a string of the form “15:35”.

Cross-references
See also @date (p. 559).
to | Expression || Program Statement

Upper limit of for loop OR lag range specifier.

to is required in the specification of a FOR loop to specify the upper limit of the control variable; see “The FOR Loop” on page 101.

When used as a lag specifier, to may be used to specify a range of lags to be used in estimation.

Syntax

Used in a FOR loop:

```
for !i = n to m
    [commands]
next
```

Used as a Lag specifier:

```
series_name(n to m)
```

Examples

```
ls cs c gdp(0 to -12)
```

Runs an OLS regression of CS on a constant, and the variables GDP, GDP(-1), GDP(-2), ..., GDP(-11), GDP(-12).

Cross-references

See “The FOR Loop” beginning on page 101. See also, exitloop (p. 561), for (p. 562), next (p. 563).

@toc | Support Function

Syntax: @toc

Return: integer

Compute elapsed time (since timer reset) in seconds.

Examples

```
tic
    [some commands]
!elapsed = @toc
```
resets the timer, executes commands, and saves the elapsed time in the control variable !ELAPSED.

Cross-references
See also tic (p. 469) and toc (p. 470).

wend

End of WHILE clause.

wend marks the end of a set of program commands that are executed under the control of a WHILE statement.

Syntax
while [condition]
[commands while condition true]
wend

Cross-references
See “The WHILE Loop” on page 104. See also while (p. 571).

while

Conditional control statement. The while statement marks the beginning of a WHILE loop.

The commands between the while keyword and the wend keyword will be executed repeatedly until the condition in the while statement is false.

Syntax
while [condition]
[commands while condition true]
wend

Cross-references
See “The WHILE Loop” on page 104. See also wend (p. 571).
Appendix F. Operator and Function Reference

The reference material in this section describes the operators and functions that may be used with series and (in some cases) matrix objects. A general description of the use of these operators and functions may be found in Chapter 6, “Working with Data”, beginning on page 111 of the User’s Guide.

This material is divided into several topics:

- Operators.
- Date and observation functions.
- Basic mathematical functions.
- Time series functions.
- Descriptive statistics.
- By-group statistics.
- Additional and special functions.
- Trigonometric functions.
- Statistical distribution functions.

For a list of functions specific to matrices, see “Matrix Function and Command Summary” on page 44.

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. The precedence of evaluation is listed in “Operators” on page 111 of the User’s Guide. Note that you can enforce order-of-evaluation using parentheses.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>add</td>
<td>x+y  adds the contents of X and Y.</td>
</tr>
<tr>
<td>-</td>
<td>subtract</td>
<td>x–y  subtracts the contents of Y from X.</td>
</tr>
<tr>
<td>*</td>
<td>multiply</td>
<td>x*y  multiplies the contents of X by Y.</td>
</tr>
<tr>
<td>/</td>
<td>divide</td>
<td>x/y  divides the contents of X by Y.</td>
</tr>
<tr>
<td>^</td>
<td>raise to the power</td>
<td>x^y  raises X to the power of Y.</td>
</tr>
</tbody>
</table>
Date and Observation Functions

These functions allow you to identify the period associated with a given observation, or the value associated with a given date/observation.

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@day</td>
<td>observation day</td>
<td>for daily or weekly workfiles, returns the observation day in the month for each observation.</td>
</tr>
<tr>
<td>@elem(x,&quot;d&quot;), @elem(x, s)</td>
<td>element</td>
<td>returns the value of the series $x$, at observation or date, $d$, or string $s$.</td>
</tr>
<tr>
<td>@month</td>
<td>observation month</td>
<td>returns the month of observation (for monthly, daily, and weekly data) for each observation.</td>
</tr>
<tr>
<td>@quarter</td>
<td>observation quarter</td>
<td>returns the quarter of observation (except for annual, semi-annual, and undated data) for each observation.</td>
</tr>
<tr>
<td>@year</td>
<td>observation year</td>
<td>returns the year associated with each observation (except for undated data) for each observation.</td>
</tr>
</tbody>
</table>

For example, if you create the series

\[
\begin{align*}
&> \\
<x \\
= \\
<> \\
<= \\
>= \\
and \\
or \\
\end{align*}
\]
Basic Mathematical Functions

These 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. The functions will return NA values for observations where the input values are NAs, or where the input values are not valid. For example, the square-root function \( @\sqrt{\text{t}} \), will return NAs for all observations less than zero.

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Examples/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@abs(x), abs(x)</td>
<td>absolute value</td>
<td>@abs(-3) = 3.</td>
</tr>
<tr>
<td>@ceiling(x)</td>
<td>smallest integer not less than</td>
<td>@ceiling(2.34) = 3; @ceiling(4) = 4.</td>
</tr>
<tr>
<td>@exp(x), exp(x)</td>
<td>exponential, ( e^x )</td>
<td>@exp(1) \approx 2.71813.</td>
</tr>
<tr>
<td>@fact(x)</td>
<td>factorial, ( x! )</td>
<td>@fact(3) = 6; @fact(0) = 1.</td>
</tr>
<tr>
<td>@factlog(x)</td>
<td>natural logarithm of the factorial, ( \log(x!) )</td>
<td>@factlog(3) \approx 1.7918; @factlog(0) = 0.</td>
</tr>
<tr>
<td>@floor(x)</td>
<td>largest integer not greater than</td>
<td>@floor(1.23) = 1; @floor(-3.1) = -4.</td>
</tr>
<tr>
<td>@inv(x)</td>
<td>reciprocal, ( 1/x )</td>
<td>@inv = 0.5.</td>
</tr>
<tr>
<td>@mod(x, y)</td>
<td>floating point remainder</td>
<td>returns the remainder of ( x/y ) with the same sign as ( x ). If ( y = 0 ) the result is 0.</td>
</tr>
<tr>
<td>@log(x), log(x)</td>
<td>natural logarithm, ( \log_e(x) )</td>
<td>@log(2) \approx 0.693; log(2.71813) \approx 1.</td>
</tr>
</tbody>
</table>
### Time Series Functions

The following functions facilitate working with time series data. Note that NAs will be returned for observations for which lagged values are not available. For example, \( d(x) \) returns a missing value for the first observation in the workfile, since the lagged value is not available.

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( @\log_{10}(x) )</td>
<td>base-10 logarithm, ( \log_{10}(x) )</td>
<td>( @\log_{10}(100) = 2 )</td>
</tr>
<tr>
<td>( @\log_b(x) )</td>
<td>base-( b ) logarithm, ( \log_b(x) )</td>
<td>( @\log(256, 2) = 8 )</td>
</tr>
<tr>
<td>( @\text{nan}(x, y) )</td>
<td>recode NAs in ( X ) to ( Y ) returns ( x ) if ( x &lt; \text{NA} ), and ( y ) if ( x = \text{NA} ).</td>
<td></td>
</tr>
<tr>
<td>( @\text{recode}(s, x, y) )</td>
<td>recode by condition returns ( x ) if condition ( s ) is true; otherwise returns ( y ): ( @\text{recode}(y &gt; 0, x, 2) ).</td>
<td></td>
</tr>
<tr>
<td>( @\text{round}(x) )</td>
<td>round to the nearest integer</td>
<td>( @\text{round}(-97.5) = -98 ); ( @\text{round}(3.5) = 4 ).</td>
</tr>
<tr>
<td>( @\sqrt(x), \text{sqr}(x) )</td>
<td>square root</td>
<td>( @\sqrt(9) = 3 ).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d(x) )</td>
<td>first difference</td>
<td>( (1 - L)X = X - X(-1) ) where ( L ) is the lag operator.</td>
</tr>
<tr>
<td>( d(x,n) )</td>
<td>( n )-th order difference</td>
<td>( (1 - L)^n X ).</td>
</tr>
<tr>
<td>( d(x,n,s) )</td>
<td>( n )-th order difference with a seasonal difference at ( s )</td>
<td>( (1 - L)^n (1 - L^s)X ).</td>
</tr>
<tr>
<td>( d\log(x) )</td>
<td>first difference of the logarithm</td>
<td>( (1 - L)\log(X) = \log(X) - \log(X(-1)) )</td>
</tr>
<tr>
<td>( d\log(x,n) )</td>
<td>( n )-th order difference of the logarithm</td>
<td>( (1 - L)^n \log(X) ).</td>
</tr>
<tr>
<td>( d\log(x,n,s) )</td>
<td>( n )-th order difference of the logarithm with a seasonal difference at ( s )</td>
<td>( (1 - L)^n (1 - L^s)\log(X) ).</td>
</tr>
<tr>
<td>( @\text{movav}(x,n) )</td>
<td>( n )-period backward moving average</td>
<td>( @\text{movav}(x, 3) = (X + X(-1) + X(-2)) / 3 ).</td>
</tr>
</tbody>
</table>
Descriptive Statistics

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, 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. For example,

\[ \text{@movsum}(x, n) \]  
- \( n \)-period backward moving sum  
  \[ \text{@movsum}(x, 3) = (X + X(-1) + X(-2)) \]

\[ \text{@pc}(x) \]  
- one-period percentage change (in percent)  
  \[ \text{equals @pch}(x) \times 100 \]

\[ \text{@pch}(x) \]  
- one-period percentage change (in decimal)  
  \[ \frac{(X - X(-1))}{X(-1)} \]

\[ \text{@pca}(x) \]  
- one-period percentage change—annualized (in percent)  
  \[ \text{equals @pcha}(x) \times 100 \]

\[ \text{@pcha}(x) \]  
- one-period percentage change—annualized (in decimal)  
  \[ (1 + \text{@pch}(x))^{n} - 1 \]
  where \( n \) is the lag associated with one-year \( (n = 4) \) for quarterly data, etc.

\[ \text{@pcy}(x) \]  
- one-year percentage change (in percent)  
  \[ \text{equals @pchy}(x) \times 100 \]

\[ \text{@pchy}(x) \]  
- one-year percentage change (in decimal)  
  \[ \frac{(X - X(-n))}{X(-n)}, \text{where } n \text{ is the lag associated with one-year } (n = 12) \text{ for annual data, etc.} \]

\[ \text{@seas}(n) \]  
- seasonal dummy  
  returns 1 when the quarter or month equals \( n \) and 0 otherwise.

\[ \text{@trend}, \text{@trend}(n) \]  
- time trend  
  returns a trend series, normalized to 0 in period \( n \), where \( n \) is a date or observation number; if \( n \) is omitted, then the series is normalized at the first observation in the workfile.
series z = @mean(x,"1945:01 1979:12")

or

w = @var(y, s2)

where S2 is the name of a sample object and W and X are series. Note that you may not use a sample argument if the results are assigned into a matrix, vector, or scalar object. For example, the following assignment

vector(2) a
series x
a(1) = @mean(x, "1945:01 1979:12")

is not valid since the target A(1) is a vector element. To perform this latter computation you must explicitly set the global sample prior to performing the calculation performing the assignment:

smpl 1945:01 1979:12
a(1) = @mean(x)

To determine the number of observations available for a given series, use the @obs function. Note that where appropriate, EViews will perform casewise exclusion of data with missing values. For example, @cov(x,y) and @cor(x,y) will use only observations for which data on both X and Y are valid.

In the following table, arguments in square brackets [ ] are optional arguments:

- [s]: sample expression in double quotes or name of a sample object. The optional sample argument may only be used if the result is assigned to a series. For @quantile, you must provide the method option argument in order to include the optional sample argument.

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@cor(x,y[,s])</td>
<td>correlation</td>
<td>the correlation between X and Y.</td>
</tr>
<tr>
<td>@cov(x,y[,s])</td>
<td>covariance</td>
<td>the covariance between X and Y.</td>
</tr>
<tr>
<td>@inner(x,y[,s])</td>
<td>inner product</td>
<td>the inner product of X and Y.</td>
</tr>
<tr>
<td>@obs(x[,s])</td>
<td>number of observations</td>
<td>the number of non-missing observations for X in the current sample.</td>
</tr>
<tr>
<td>@mean(x[,s])</td>
<td>mean</td>
<td>average of the values in X.</td>
</tr>
</tbody>
</table>
By-Group Statistics

The following “by group”-statistics are available in EViews. They may be used as part of a series expression statements to compute the statistic for subgroups, and to assign the value of the relevant statistic to each observation.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@obsby(arg1, arg2[, s])</td>
<td>Number of observations in each group.</td>
</tr>
<tr>
<td>@sumsby(arg1, arg2[, s])</td>
<td>Sum of observations in each group.</td>
</tr>
<tr>
<td>@meansby(arg1, arg2[, s])</td>
<td>Mean of observations in each group.</td>
</tr>
<tr>
<td>@minsby(arg1, arg2[, s])</td>
<td>Minimum value of observations in each group.</td>
</tr>
<tr>
<td>@maxsby(arg1, arg2[, s])</td>
<td>Maximum value of observations in each group.</td>
</tr>
<tr>
<td>@mediansby(arg1, arg2[, s])</td>
<td>Median of observations in each group.</td>
</tr>
<tr>
<td>@varsby(arg1, arg2[, s])</td>
<td>Variance of observations in each group.</td>
</tr>
<tr>
<td>@stdevsby(arg1, arg2[, s])</td>
<td>Standard deviation of observations in each group.</td>
</tr>
<tr>
<td>@sumsqby(arg1, arg2[, s])</td>
<td>Sum of squares of observations in each group.</td>
</tr>
<tr>
<td>@quantilesby(arg1, arg2, q[, s])</td>
<td>Quantiles of observations in each group.</td>
</tr>
</tbody>
</table>
With the exception of @QUANTILEBY, all of the functions take the form:

```
@STATBY(arg1, arg2[, s])
```

where @STATBY is one of the by-group function keyword names, arg1 is a series expression, arg2 is a numeric or alpha series expression identifying the subgroups, and s is an optional sample literal (a quoted sample expression) or a named sample. With the exception of @OBSBY, arg1 must be a numeric series.

By default, EViews will use the workfile sample when computing the descriptive statistics. You may provide the optional sample s as a literal (quoted) sample expression or a named sample.

The @QUANTILEBY function requires an additional argument for the quantile value that you wish to compute

```
@QUANTILEBY(arg1, arg2, q[, s])
```

For example, to compute the first quartile, you should use the q value of .25.

There are two related functions of note:

```
@GROUPID(arg[, smpl])
```

returns an integer indexing the group ID for each observation of the series or alpha expression arg, while

```
@NGROUPS(arg[, smpl])
```

returns a scalar indicating the number of groups (distinct values) for the series or alpha expression arg.

### Additional and Special Functions

EViews provides a number of utility and “special” functions used in evaluating the properties of various statistical distributions or for returning special mathematical values such as Euler’s constant. For further details on special functions, see the extensive discussions in Temme (1996), Abramowitz and Stegun (1964), and Press, et al. (1992).

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@skewsby()</td>
<td>Skewness of observations in each group.</td>
</tr>
<tr>
<td>@kurtbsby()</td>
<td>Kurtosis of observations in each group.</td>
</tr>
<tr>
<td>@nasbsy()</td>
<td>Number of NA values in each group.</td>
</tr>
</tbody>
</table>
\[
\begin{align*}
@\text{beta}(a,b) & \quad \text{beta integral (Euler integral of the second kind)} \\
& \quad B(a, b) = \int_0^1 t^{a-1}(1-t)^{b-1} dt = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)} \\
& \quad \text{for } a, b > 0. \\
@\text{betainc}(x,a,b) & \quad \text{incomplete beta integral} \\
& \quad \frac{1}{B(a, b)}\int_0^x t^{a-1}(1-t)^{b-1} dt \\
& \quad \text{for } 0 \leq x \leq 1 \text{ and } a, b > 0. \\
@\text{betaincder}(x,a,b,s) & \quad \text{derivative of the incomplete beta integral:} \\
& \quad \text{Evaluates the derivatives of the incomplete beta integral } B(x, a, b), \text{ where } s \text{ is an integer from 1 to 9 corresponding to the desired derivative:} \\
& \quad \left[ \begin{array}{cccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{array} \right] = \\
& \quad \left[ \begin{array}{ccc}
\frac{\partial B}{\partial x} & \frac{\partial B}{\partial a} & \frac{\partial B}{\partial b} \\
\frac{\partial^2 B}{\partial x^2} & \frac{\partial^2 B}{\partial x \partial a} & \frac{\partial^2 B}{\partial x \partial b} \\
\frac{\partial^2 B}{\partial a^2} & \frac{\partial^2 B}{\partial a \partial b} & \frac{\partial^2 B}{\partial b^2} \\
\end{array} \right] \\
@\text{betaincinv}(p,a,b) & \quad \text{inverse of the incomplete beta integral: returns an } x \text{ satisfying:} \\
& \quad p = \frac{1}{B(a, b)}\int_0^x t^{a-1}(1-t)^{b-1} dt \\
& \quad \text{for } 0 \leq p \leq 1 \text{ and } a, b > 0. \\
@\text{betalog}(a,b) & \quad \text{natural logarithm of the beta integral:} \\
& \quad \log B(a, b) = \log \Gamma(a) + \log \Gamma(b) - \log \Gamma(a+b). \\
@\text{binom}(n,x) & \quad \text{binomial coefficient} \\
& \quad \binom{n}{x} = \frac{n!}{x!(n-x)!} \\
& \quad \text{for } n \text{ and } x \text{ positive integers, } 0 \leq x \leq n. \\
@\text{binomlog}(n,x) & \quad \text{natural logarithm of the binomial coefficient:} \\
& \quad \log(n!) - \log(x!) - \log((n-x)!) \\
\end{align*}
\]
### Appendix F. Operator and Function Reference

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| @cloglog(x) | Complementary log-log function: \( \log(-\log(1-x)) \)  
|  | See also @gextreme. |
| @digamma(x), @psi(x) | First derivative of the log gamma function:  
|  | \( \psi(x) = \frac{d\log \Gamma(x)}{dx} = \frac{1}{\Gamma(x)} \frac{d\Gamma(x)}{dx} \) |
| @erf(x) | Error function:  
|  | \( \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \)  
|  | for \( x \geq 0 \). |
| @erfc(x) | Complementary error function:  
|  | \( \text{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt = 1 - \text{erf}(x) \)  
|  | for \( x \geq 0 \). |
| @gamma(x) | Complete gamma function:  
|  | \( \Gamma(x) = \int_0^\infty e^{-t} t^{x-1} dt \)  
|  | for \( x \geq 0 \). |
| @gammader(x) | First derivative of the gamma function:  
|  | \( \Gamma'(x) = \frac{d\Gamma(x)}{dx} \)  
|  | Note: Euler’s constant, \( \gamma \approx 0.5772 \), may be evaluated as \( \gamma = -\text{gammader}(1) \). See also @digamma and @trigamma. |
| @gammainc(x,a) | Incomplete gamma function:  
|  | \( G(x, a) = \frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1} dt \)  
|  | for \( x \geq 0 \) and \( a > 0 \). |
Trigonometric Functions

When applied to a series, all of the trigonometric functions operate on every observation in the current sample and return a value for every observation. Where relevant, the input and results should/will be expressed in radians. All results are real valued—complex values will return NAs.

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Examples/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@acos(x)</td>
<td>arc cosine (real results in radians)</td>
<td>@acos(-1) = \pi</td>
</tr>
<tr>
<td>@asin(x)</td>
<td>arc sine (real results in radians)</td>
<td>@asin(-1) = \pi /2</td>
</tr>
<tr>
<td>@atan(x)</td>
<td>arc tangent (results in radians)</td>
<td>@atan(1) = \pi /4</td>
</tr>
</tbody>
</table>
Statistical Distribution Functions

The following functions provide access to the density or probability functions, cumulative distribution, quantile functions, and random number generators for a number of standard statistical distributions.

There are four functions associated with each distribution. The first character of each function name identifies the type of function:

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Beginning of Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cumulative distribution (CDF)</td>
<td>@c</td>
</tr>
<tr>
<td>Density or probability</td>
<td>@d</td>
</tr>
<tr>
<td>Quantile (inverse CDF)</td>
<td>@q</td>
</tr>
<tr>
<td>Random number generator</td>
<td>@r</td>
</tr>
</tbody>
</table>

The remainder of the function name identifies the distribution. For example, the functions for the beta distribution are @cbeta, @dbeta, @qbeta and @rbeta.

When used with series arguments, EViews will evaluate the function for each observation in the current sample. As with other functions, NA or invalid inputs will yield NA values. For values outside of the support, the functions will return zero.

Note that the CDFs are assumed to be right-continuous: \( F_X(k) = \Pr(X \leq k) \). The quantile functions will return the smallest value where the CDF evaluated at the value equals or exceeds the probability of interest: \( q_X(p) = q^* \), where \( F_X(q^*) \geq p \). The inequalities are only relevant for discrete distributions.

The information provided below should be sufficient to identify the meaning of the parameters for each distribution. For further details, see the *Command and Programming Reference*.

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Functions</th>
<th>Density/Probability Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beta</td>
<td>@cbeta(x, a, b), @dbeta(x, a, b), @qbeta(p, a, b), @rbeta(a, b)</td>
<td>( f(x, a, b) = \frac{x^{a-1}(1-x)^{b-1}}{B(a, b)} ) for ( 0 \leq p \leq 1 ) and for ( a, b &gt; 0 ), where ( B ) is the @beta function.</td>
</tr>
</tbody>
</table>
**Binomial**  
\( \text{Pr}(x, n, p) = \binom{n}{x} p^x (1-p)^{n-x} \)  
if \( x = 0, 1, \ldots, n, \ldots \), and 0 otherwise, for \( 0 \leq p \leq 1 \).

**Chi-square**  
\[ f(x, v) = \frac{1}{2^{v/2} \Gamma(v/2)} x^{v/2-1} e^{-x/2} \]  
where \( x \geq 0 \), and \( v > 0 \). Note that the degrees-of-freedom parameter \( v \) need not be an integer.

**Exponential**  
\[ f(x, m) = \frac{1}{m} e^{-x/m} \]  
for \( x \geq 0 \), and \( m > 0 \).

**Extreme Value**  
**(Type I-minimum)**  
\[ f(x) = \exp(x - e^x) \]  
for \(-\infty < x < \infty\).

**F-distribution**  
\[ f(x, v_1, v_2) = \frac{v_1/2}{B(v_1/2, v_2/2)} \frac{1}{v_2} x^{v_1/2-1} \left( v_2 + v_1 x \right)^{-v_1/2} \]  
where \( x \geq 0 \), and \( v_1, v_2 > 0 \). Note that the functions allow for fractional degrees-of-freedom parameters \( v_1 \) and \( v_2 \).

**Gamma**  
\[ f(x, b, r) = b^r x^{r-1} e^{-x/b} / \Gamma(r) \]  
where \( x \geq 0 \), and \( b, r > 0 \).

**Generalized Error**  
\[ f(x, r) = \frac{r I(\frac{2}{r})^{1/2}}{2 I(\frac{1}{r})^{3/2}} \exp \left( -|x|^r \left( \frac{I(\frac{2}{r})}{I(\frac{1}{r})} \right)^{r/2} \right) \]  
where \(-\infty < x < \infty\), and \( r > 0 \).
### Appendix F. Operator and Function Reference

#### Laplace

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@claplace(x), @dlaplace(x), @qlaplace(x), @rlaplace(x)</td>
<td>( f(x) = \frac{1}{2} e^{-</td>
</tr>
</tbody>
</table>

#### Logistic

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@clogistic(x), @dlogistic(x), @qlogistic(p), @rlogistic(x)</td>
<td>( f(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x} ) for (-\infty &lt; x &lt; \infty).</td>
</tr>
</tbody>
</table>

#### Log-normal

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@clognorm(x,m,s), @dlognorm(x,m,s), @qlognorm(p,m,s), @rlognorm(m,s)</td>
<td>( x &gt; 0, -\infty &lt; m &lt; \infty, ) and ( s &gt; 0 ).</td>
</tr>
</tbody>
</table>

#### Negative Binomial

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@cnegbin(x,n,p), @dnegbin(x,n,p), @qnegbin(s,n,p), @rnegbin(n,p)</td>
<td>( \Pr(x,n,p) = \frac{\Gamma(x + n)}{\Gamma(x + 1)\Gamma(n)} p^n (1 - p)^x ) if ( x = 0, 1, \ldots, n, \ldots, ) and 0 otherwise, for ( 0 \leq x \leq 1 ).</td>
</tr>
</tbody>
</table>

#### Normal (Gaussian)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@cnorm(x), @dnorm(x), @qnorm(p), @rnorm, nrnd</td>
<td>( f(x) = (2\pi)^{-1/2} e^{-x^2/2} ) for (-\infty &lt; x &lt; \infty).</td>
</tr>
</tbody>
</table>

#### Poisson

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@cpoisson(x,m), @dpoisson(x,m), @qpoisson(p,m), @rpoisson(m)</td>
<td>( \Pr(x,m) = m^x e^{-m} / x! ) if ( x = 0, 1, \ldots, n, \ldots, ) and 0 otherwise, for ( m &gt; 0 ).</td>
</tr>
</tbody>
</table>

#### Pareto

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@pareto(x,a,k), @dpareto(x,a,k), @qpareto(p,a,k), @rpareto(a,k)</td>
<td>( f(x,a,k) = (ak^a) / x^{a+1} ) for ( a &gt; 0, ) and ( 0 \leq k \leq x ).</td>
</tr>
</tbody>
</table>

#### Student’s t-distribution

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ctdist(x,v), @dtdist(x,v), @qtdist(p,v), @rtdist(v)</td>
<td>( f(x,v) = \frac{1}{\Gamma\left(\frac{v+1}{2}\right)} \left(1 + \left(\frac{x^2}{v}\right)\right)^{-\frac{v+1}{2}} ) for (-\infty &lt; x &lt; \infty, ) and ( v &gt; 0 ). Note that ( v = 1 ) yields the Cauchy distribution.</td>
</tr>
</tbody>
</table>
Additional Distribution Related Functions

The following utility functions were designed to facilitate the computation of p-values for common statistical tests. While these results may be derived using the distributional functions above, they are retained for convenience and backward compatibility.

<table>
<thead>
<tr>
<th>Function</th>
<th>Distribution</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@chisq(x,v)</td>
<td>Chi-square</td>
<td>Returns the probability that a Chi-squared statistic with v degrees of freedom exceeds x: $@chisq(x,v) = 1 - @cchisq(x,d)$</td>
</tr>
<tr>
<td>@fdist(x,v1,v2)</td>
<td>F-distribution</td>
<td>Probability that an F-statistic with $v_1$ numerator degrees of freedom and $v_2$ denominator degrees of freedom exceeds x: $@fdist(x,v1,v2) = 1 - @cfdist(x,v1,v2)$</td>
</tr>
<tr>
<td>@tdist(x,v)</td>
<td>t-distribution</td>
<td>Probability that a t-statistic with v degrees of freedom exceeds x in absolute value (two-sided p-value): $@tdist(x,v) = 2*(1 - @ctdist(x,v))$</td>
</tr>
</tbody>
</table>
Index

Symbols

! (exclamation) control variable 88
% (percent sign)
  program arguments 96, 97
  string variable 89
* (asterisk) multiplication 35
+ (plus)
  addition 35
/ (slash) division 36
@abs 575
@beta 581
@betainc 581
@betaincder 581
@betaincinv 581
@betalog 581
@binom 581
@binomlog 581
@ceiling 575
@cholesky 527
@cloglog 582
@columnextract 528
@columns 528
@cond 528
@convert 529
@cor 530, 578
@cov 530, 578
@date 559
@datestr 531
@dateval 532
@day 574
@det 532
@digamma 582
@dtoc 533
@eigenvalues 533
@eigenvectors 534
@elem 574
@eqna 534
@erf 582
@erfc 582
@errorcount 561
@evpath 561
@exp 575
@expand 519
@expode 525
@fact 575
@factlog 575
@filledmatrix 535
@filledrowvector 535
@filledsym 536
@filledvector 536
@floor 575
@-functions
  by-group statistics 579
  date functions 574
  descriptive statistics 577
  integrals and other special functions 580
  mathematical functions 575
  special p-value functions 587
  statistical distribution functions 584
  time series functions 576
  trigonometric functions 583
@gamma 582
@gammader 582
@gammainc 582
@gammaincder 583
@gammaincinv 583
@gammalog 583
@getmaindiagonal 536
@groupid 580
@identity 537
@implode 539
@inner 537, 578
@insert 538
@instr 539
@inv 575
@inverse 539
@isempty 540
@isobject 563
@issingular 540
@kronecker 540
@kurt5 580
@left 541, 549
@length 541
@log 575
@log10 576
@logit 583
@logx 576
@lower 542
@trim 542
@makediagonal 542
@max 579
@maxsby 579
@meansby 579
@median 579
@mediansby 579
@mid 544
@min 579
@minsby 579
@mod 575
@month 574
@movav 576
@movsum 577
@nan 576
@nasby 580
@negna 545
@ngroups 580
@norm 546
@obs 578
@obsby 579
@obsrange 564
@obssmpl 564
@otod 546
@outer 547
@permute 547
@psi 582
@quantile 579
@quantilesby 579
@quarter 574
@rank 547
@recode 576
@replace 548
@resample 548
@round 576
@rowextract 550
@rows 550
@rtrim 551
@seas 577
@skewsby 580
@solve system 551
@sqrt 576
@stdev 579
@stdevsby 579
@str 553
@strdate 553
@strlen 553
@strown 554
@subextract 554
@sum 579
@sumsby 579
@sumsq 579
@sumsqsbys 579
@svd 555
@temppath 568
@time 569
@toc 570
@trace 555
@transpose 556
@trend 577
@trigamma 583
@trim 556
@unitvector 556
@upper 557
@val 557
@var 579
@varsby 579
@vec 558
@vech 558
@year 574
_ (continuation character) 84
= (dash)
    negation 34
    subtraction 35

**Numerics**

2sls (Two-Stage Least Squares) 474
3sls (Three-stage least squares) 178

**A**

abs 575
add 179
Add factor
  initialization 181
Add text to graph 183
addassign 180
addinit 181
Addition operator (+) 35
addtext 183
align 184
Align multiple graphs 184
alpha 185
Alpha (object) 136, 171
  data members 136
  declare 185
  element function 136
  views 136
And operator 574
Anderson-Darling test 257
Andrews test 466
append 186
Append specification line 186
AR
  autoregressive error 521
  inverse roots of polynomial 195
  seasonal 525
ar 521
Arc cosine 583
Arc sine 583
Arc tangent 583
ARCH
  residual LM test for ARCH 190
  see also GARCH 187
archtest 190
area 191
Area graph 191
Arguments
  in programs 97
  in subroutines 109
arlm 193
ARMA
  structure view 194
arroots 195
Assign values to matrix objects 24
  by element 24
  converting series or group 30
  copy 27
  copy submatrix 29
  fill procedure 25
Augmented Dickey-Fuller test 481
auto 196
Autocorrelation
  compute and display 231
  multivariate VAR residual test 376
Autoregressive error. See AR.
  Autoregressive conditional heteroskedasticity
    See ARCH and GARCH.
  Autowrap text 83
Auxiliary commands 10
  summary 16
Axis
  rename label 340
  set axis characteristics 197
  set scaling in graph 397
  axis 197
B
Background color 416
Band-pass filter 207
Bar graph 199
Batch mode
  See Program.
  Baxter-King band-pass filter 207
BDS test 201
bdstest 201
Beta
  distribution 584
  integral 581
  Beta integral
    logarithm 581
binary 202
Binary dependent variables 202
Binary models
  prediction table 372
Binomial
  coefficient function 581
  distribution 585
  Binomial coefficients 581
  logarithm 581
  block 203
Bootstrap rows of matrix 548
Boxplot 204, 205
  customize 210
customize individual elements 407
bplabel 210
Breusch-Godfrey test
See also Serial correlation.
by-group 579
By-group statistics 579
C
call 559
Call subroutine 110
Causality test 212
cause 212
ccopy 213
cd 213
cdfplot 214
Cell
background color 416
borders 427
display format 419
font selection 418
height 423
indentation 424
justification 425
merging multiple 428
set text color 430
width 431
censored 217
Censored dependent variable models 217
cfetch 218
Change default directory 213, 219
checkderivs 219
Chi-square distribution 585
Cholesky factor
function to compute 527
chow 220
Chow test 220
Christiano-Fitzgerald band-pass filter 207
clabel 221
cleartext 221
close 222
Close EViews application 264
Close window 222
coeff 223
Coef (object) 137
data members 138
declare 223
fill values 267
procs 138
views 137
coeefcov 224
Coefficient
covariance matrix of estimates 224
See Coef (object).
update default coef vector 480
coint 224
Cointegration
make cointegrating relations from VEC 319
Cointegration test 224
colplace 527
Column
extract from matrix 528
number of columns in matrix 528
place in matrix 527
stack matrix 558
stack matrix (lower triangle) 558
Column width 50
Column width of table 410
Commands
auxiliary 10, 16
basic command summary 16
batch mode 2
execute without opening window 252
interactive mode 1
object assignment 9
object command 6
object declaration 5, 9
save record of 2
window 1
comment 226
Comments in tables 226
Comparison operators 36
Complementary log-log function 582
Condition number of matrix 528
Conditional standard deviation
display graph of 280
Conditional variance
make series from ARCH 322
Confidence ellipse 215
Container (object) 11
database 12
workfile 11
Continuation character in programs 84
control 227
Control variable 88
   as replacement variable 92
   in while statement 105
   save value to scalar 88
Convert
   date to observation number 533
   matrix object to series or group 544
   matrix objects 30, 42, 529
   matrix to sym 537
   observation number to date 546
   scalar to string 553
   series or group to matrix (drop NAs) 551
   series or group to matrix (keep NAs) 552
   string to scalar 90, 557
   sym to matrix 535
Coordinates
   for legend in graph 305
Copy
   database 240
   objects 228
   workfile page 353
Copy objects 14
cor 230
correl 231
Correlation 530, 578
   cross 235
Correlation matrix 230
Correlogram 231, 235
   squared residuals 232
corrsq 232
Cosine 584
count 233
Count models 233
cov 234
Covariance 530, 578
Covariance matrix 234
Cramer-von Mises test 257
Create
   database 240
   workfile 491
   workfile page 355
cross 235
Cross correlations 235
Cross product 547
Cross section member
add to pool 179
   define list of 247
Current date function 559
Current time function 569
CUSUM test 387
   of squares 387
D
Data
   enter from keyboard 236
data 236
Data members
   coef 138
   equation 141
   group 146
   matrix 151
   pool 155
   rowvector 157
   series 136, 160
   sspace 162
   sym 166
   system 167
   table 170
   var 172
   vector 175
Database
   copy 13, 240
   create 12, 240
   delete 13, 241
   fetch 264
   Haver Analytics 290, 291, 294
   open 13
   open existing 242
   open or create 239
   pack 243
   rebuild 243
   rename 13, 244
   repair 245
   store object in 452
Date string 531, 532
Dated data report table 255
Dated data table 324
datelabel 237
Dates
   convert from observation number 546
   convert to observation number 536
   current as string 533
functions 574
workfile dates as strings 553
dates 239
db 239
dbcopy 240
dbcreate 240
dbdelete 241
dbopen 242
dbpack 243
dbrebuild 243
dbrename 244
dbrepair 245
Declare
  matrix 23
  object 5, 9
decomp 245
define 247
Delete
  database 241
  object 15
  objects or pool identifiers 248
  workfile page 356
delete 248
Derivatives
  examine derivs of specification 249
  make series or group containing 320
derivs 249
describe 250
Descriptive statistics 449, 579
  @-functions 577, 579
  by category of dependent variable 336
  by classification 444
  make series 332
  matrix functions 45
  pool 250
Determinant 532
Diagonal matrix 542
Dickey-Fuller test 481
Difference 576
Directory
  change working 213, 219
  EViews executable 561
Display
  action 6
  and print 8
Display numbers 432
Display object 432
displayname 251
Distribution function
  empirical cumulative, survivor and quantiles 214
  Quantile-quantile plot 375
Division operator (/) 36
do 252
Double exponential smoothing 434
draw 252
Draw lines in graph 252
DRI database
  convert to EViews database 254
  copy from 213
  fetch series 218
  read series description 221
dricontvert 254
Drop
  group series or cross-section from pool definition 255
drop 255
dtable 255
Dummy variable
  seasonal 577
Dummy variables
  automatic creation 519
Durbin’s h 108
Dynamic forecast 272
E
ec 256
edftest 257
Eigenvalues 533
Eigenvectors 534
Elapsed time 470, 570
Element
  assign in matrix 24
  matrix functions 45
else 560
Else clause in if statement 100, 560
Empirical distribution functions 214
Empirical distribution test 257
Empty string 540
endif 560
endog 259
Endogenous variables
make series or group 321
Endogenous variables of specification 259
endsub 560
eqs 260
equals comparison 36
equation 260
Equation (object) 139
data members 141
declare 260
methods 139
procs 140
views 139
erbar 261
Error bar graph 261
Error correction model
See VEC and VAR. Vector error correction model
Error count in programs 561
Error function 582
complementary 582
Error handling 106
Estimate
2sls 474
least squares 315
nonlinear least squares 315
show results 347
Estimation methods
(single) equation 139
pool 317
state space 148, 161
system (of equations) 166
VAR 171
Euler’s constant 582
EViews
running application in batch mode 86
Excel file
export data to file 498
reading data from 377
exclude 263
Exclude variables from model solution 263
Execute program 84
abort 86
quiet mode 85
verbose mode 85
with arguments 96, 97
Exit
from EViews 264
loop 107, 561
subroutine 566
exit 264
exitloop 561
exp 575
expand 264
Expand workfile 264
Exponential
distribution 585
function 575
Exponential smoothing 434
Export
matrix 41
workfile data to file 498
workfile page 358
Extract
row vector 550
submatrix from matrix 554
Extreme value distribution 585
F
Factorial 575
F-distribution 585
fetch 264
Fetch object 16, 264
Files
open program or text file 342
temporary location 568
Fill
values of matrix 25
values of object 267
fill 267
Filled
matrix 535
row vector 535
symmetric matrix 536
vector 536
fiml 269
fit 270
Font selection 418
for 562
For loop
accessing elements of a series 102
accessing elements of a vector 102
changing samples within 101
define using control variables 101
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>define using scalars</td>
<td>103</td>
</tr>
<tr>
<td>define using string variables</td>
<td>103</td>
</tr>
<tr>
<td>exit loop</td>
<td>107</td>
</tr>
<tr>
<td>mark end</td>
<td>563</td>
</tr>
<tr>
<td>nesting</td>
<td>102</td>
</tr>
<tr>
<td>roundoff error in</td>
<td>567</td>
</tr>
<tr>
<td>start loop</td>
<td>562</td>
</tr>
<tr>
<td>step size</td>
<td>567</td>
</tr>
<tr>
<td>upper limit</td>
<td>570</td>
</tr>
<tr>
<td>Forecast</td>
<td></td>
</tr>
<tr>
<td>dynamic (multi-period)</td>
<td>272</td>
</tr>
<tr>
<td>static (one-period ahead)</td>
<td>270</td>
</tr>
<tr>
<td>forecast</td>
<td>272</td>
</tr>
<tr>
<td>Format number</td>
<td>49</td>
</tr>
<tr>
<td>Formula series</td>
<td>278</td>
</tr>
<tr>
<td>Freeze</td>
<td></td>
</tr>
<tr>
<td>table</td>
<td>47</td>
</tr>
<tr>
<td>freeze</td>
<td>275</td>
</tr>
<tr>
<td>Freeze view</td>
<td>275</td>
</tr>
<tr>
<td>freq</td>
<td>276</td>
</tr>
<tr>
<td>Frequency conversion</td>
<td>228, 264, 309</td>
</tr>
<tr>
<td>default method for series</td>
<td>410</td>
</tr>
<tr>
<td>using links</td>
<td>310</td>
</tr>
<tr>
<td>Frequency table</td>
<td>one-way</td>
</tr>
<tr>
<td>frml</td>
<td>278</td>
</tr>
<tr>
<td>Full information maximum likelihood</td>
<td>269</td>
</tr>
<tr>
<td><strong>G</strong></td>
<td></td>
</tr>
<tr>
<td>Gamma</td>
<td></td>
</tr>
<tr>
<td>distribution</td>
<td>585</td>
</tr>
<tr>
<td>Gamma function</td>
<td>582</td>
</tr>
<tr>
<td>derivative</td>
<td>582</td>
</tr>
<tr>
<td>incomplete</td>
<td>582</td>
</tr>
<tr>
<td>incomplete derivative</td>
<td>583</td>
</tr>
<tr>
<td>incomplete inverse</td>
<td>583</td>
</tr>
<tr>
<td>logarithm</td>
<td>583</td>
</tr>
<tr>
<td>second derivative</td>
<td>583</td>
</tr>
<tr>
<td>GARCH</td>
<td></td>
</tr>
<tr>
<td>display conditional standard deviation</td>
<td>280</td>
</tr>
<tr>
<td>estimate model</td>
<td>187</td>
</tr>
<tr>
<td>generate conditional variance series</td>
<td>322</td>
</tr>
<tr>
<td>garch</td>
<td>280</td>
</tr>
<tr>
<td>Gaussian distribution</td>
<td>586</td>
</tr>
<tr>
<td>Generalized autoregressive conditional heteroskedasticity</td>
<td>See ARCH and GARCH.</td>
</tr>
<tr>
<td>Generalized error distribution</td>
<td>585</td>
</tr>
<tr>
<td>Generate series</td>
<td></td>
</tr>
<tr>
<td>for pool</td>
<td>281</td>
</tr>
<tr>
<td>genr</td>
<td>281</td>
</tr>
<tr>
<td>See also series.</td>
<td></td>
</tr>
<tr>
<td>Global</td>
<td></td>
</tr>
<tr>
<td>subroutine</td>
<td>111</td>
</tr>
<tr>
<td>variable</td>
<td>111</td>
</tr>
<tr>
<td>GMM</td>
<td></td>
</tr>
<tr>
<td>estimate by</td>
<td>282</td>
</tr>
<tr>
<td>gmm</td>
<td>282</td>
</tr>
<tr>
<td>Gompit models</td>
<td>202</td>
</tr>
<tr>
<td>Goodness of fit (for binary models)</td>
<td>466</td>
</tr>
<tr>
<td>Gradients</td>
<td></td>
</tr>
<tr>
<td>display</td>
<td>287</td>
</tr>
<tr>
<td>saving in series</td>
<td>322</td>
</tr>
<tr>
<td>grads</td>
<td>287</td>
</tr>
<tr>
<td>Granger causality test</td>
<td>212, 465</td>
</tr>
<tr>
<td>Graph</td>
<td></td>
</tr>
<tr>
<td>align multiple graphs</td>
<td>184</td>
</tr>
<tr>
<td>area graph</td>
<td>191</td>
</tr>
<tr>
<td>axis labeling</td>
<td>237, 239</td>
</tr>
<tr>
<td>bar graph</td>
<td>199</td>
</tr>
<tr>
<td>change legend or axis name</td>
<td>340</td>
</tr>
<tr>
<td>create</td>
<td>59</td>
</tr>
<tr>
<td>drawing lines and shaded areas</td>
<td>252</td>
</tr>
<tr>
<td>error bar</td>
<td>261</td>
</tr>
<tr>
<td>high-low-open-close</td>
<td>292</td>
</tr>
<tr>
<td>legend appearance and placement</td>
<td>304</td>
</tr>
<tr>
<td>line graph</td>
<td>306</td>
</tr>
<tr>
<td>merging graphs</td>
<td>288</td>
</tr>
<tr>
<td>options for bottom axis in boxplotst</td>
<td>210</td>
</tr>
<tr>
<td>options for individual elements</td>
<td>412</td>
</tr>
<tr>
<td>options for individual elements of a boxplot</td>
<td>407</td>
</tr>
<tr>
<td>pie graph</td>
<td>370</td>
</tr>
<tr>
<td>place text in</td>
<td>183</td>
</tr>
<tr>
<td>save to disk</td>
<td>393</td>
</tr>
<tr>
<td>scatterplot graph</td>
<td>399</td>
</tr>
<tr>
<td>set axis characteristics</td>
<td>197</td>
</tr>
<tr>
<td>set axis scale</td>
<td>397</td>
</tr>
<tr>
<td>spike graph</td>
<td>442</td>
</tr>
<tr>
<td>templates</td>
<td>460</td>
</tr>
<tr>
<td>XY graph</td>
<td>510</td>
</tr>
<tr>
<td>XY line graph</td>
<td>512</td>
</tr>
<tr>
<td>XY pairs graph</td>
<td>515</td>
</tr>
</tbody>
</table>
graph 288
Graph (object) 143
  create using freeze 275
declaring 288
graph commands 143
merge multiple 336
procs 143, 144
set options 343
greater than comparison 36
greater than or equal to comparison 36
Group
  add series 179
  convert to matrix 450, 451, 544
  convert to matrix (with NAs) 552
  group 290
Group (object) 145
  data members 146
  declare 290
  procs 146
  views 145

H
Haver Analytics Database
  convert to EVIEWS database 290
  display series description 294
  fetch series from 291
hconvert 290
Heteroskedasticity test (White) 495
hfetch 291
High-Low-(Open-Close) graph 292
hilo 292
hist 294
Histogram 294
hlabel 294
Hodrick-Prescott filter 295
Holt-Winters 434
Hosmer-Lemeshow test 466
hpf 295
HTML
  save table as 57
I
Identity matrix 537
  extract column 556
if 562
  If statement 99
else clause 100, 560
  end of condition 560
  start of condition 562
  then 569
Import data
  matrix 40
Import data from file 377
impulse 296
Impulse response function 296
Include
  file in a program file 563
  program line 107
include 563
Incomplete beta 581
  derivative 581
  integral 581
  inverse 581
Incomplete gamma 582
  derivative 583
  inverse 583
Independence test 201
Initial parameter values 368
Inner product 537, 578
Insertion point in command line 1
Integer random number 388
Interactive mode 1
Inverse of matrix 539

J
Jarque-Bera
  multivariate normality test 299
jbera 299
Johansen cointegration test 224

K
Kalman filter 321
kdensity 300
kerfit 301
Kernel
  bivariate regression 301
density 300
Keyboard focus
  defaults 2
Kolmogorov-Smirnov test 257
KPSS unit root test 481
Kronecker product 540

L

label 302
Label object 251, 302
Label values 334
Lag
  specify as range 570
  VAR lag order selection 303
Lag exclusion test 467
laglen 303
Lagrange multiplier
  test for ARCH in residuals 190
Landscape printing 8
Laplace distribution 586
Least squares estimation 315
Legend
  appearance and placement 304
  rename 340
legend 304
Length of string 541
less than comparison 36
less than or equal to comparison 36
Lilliefors test 257
line 306
Line drawing 252
Line graph 306
Line pattern 252
Line style 252
linefit 308
link 309
Link (object) 147
  declare 309
  specification 310
Link object
  convert to ordinary series 479
linito 310
load 313
Local
  subroutine 113
  variable 111
log
  arbitrary base 576
  base 10 576
  natural 575
Log difference 576
Logistic
  logit function 583
Logistic distribution 586
logit 314
Logit models 202
logl 314
Logl (object) 148
  append specification line 186
  check user-supplied derivatives 219
  data members 149
  declare 314
  method 148
  procs 148
  statements 149
  views 148
Log-normal distribution 586
Loop
  exit loop 107, 561
  for (control variables) 101
  for (scalars) 103
  for (string variables) 103
  nest 102
  over matrix elements 41, 102
  while 104
Lotus file
  export data to file 498
Lowercase 542
ls 315

M

MA
  seasonal 526
ma 521
Make model object 327
makecoint 319
makederivs 320
makeendog 321
makefilter 321
makegarch 322
makegrads 322
makegraph 323
makegroup 324
makelimits 326
makemodel 327
makeregs 327
makeresids 328
makesignals 330
makestates 331
makestats 332
makesystem 333
map 334
Match merge 309, 310
Mathematical functions 575
matplace 543
Matrix
  assign values 24
  convert to other matrix objects 42
  convert to series or group 30
  copy 27
  copy submatrix 29
  declare 23
  export data 41
  filled 535
  import data 40
  main diagonal 536
  objects 23
  permute rows of 547
  place submatrix 543
  resample rows from 548
  singular value decomposition 555
  stack columns 558
  stack lower triangular columns 558
  matrix 335
Matrix (object)
  data members 151
  declare 335
  fill values 267
  procs 151
  views 40, 150
Matrix commands and functions
  commands 38
  descriptive statistics 37, 45
  difference 38
  element 37, 45
  functions 38
  matrix algebra 37, 45
  missing values 39
  utility 37, 44
Matrix operators
  addition (+) 35
  and loop operators 41
  comparison operators 36
division (/) 36
  multiplication (*) 35
  negation (-) 34
  order of evaluation 34
  subtraction (-) 35
Maximum 579
Maximum likelihood estimation 338
  Logl (object) 148
  Sspace (object) 161
Mean 578
  equality test 462
means 336
Median 579
  equality test 462
Merge
  graph objects 288, 336
  into model 336
  using links 310
merge 336
Messages
  model solution 339
  suppress during program execution 85
Minimum 579
Missing values 39
  code for missing 522
  mathematical functions 575
  recoding 576
ml 338
model 339
Model (object) 152
  append specification line 186
  break all model links 479
  declare 339
  equation view 260
  merge into 336
  procs 152
  scenarios 401
  update specification 480
  variable view 488
  views 152
Models
  add factor assignment and removal 180
  add factor initialization 181
  block structure 203
  exclude variables from solution 263
  make from estimation object 323
  make graph of model series 327
make graph of model series 327
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>make group of model series</td>
<td>324</td>
</tr>
<tr>
<td>options for solving</td>
<td>349</td>
</tr>
<tr>
<td>overrides in model solution</td>
<td>349</td>
</tr>
<tr>
<td>solution messages</td>
<td>339</td>
</tr>
<tr>
<td>solve</td>
<td>438</td>
</tr>
<tr>
<td>solve control to match target</td>
<td>227</td>
</tr>
<tr>
<td>text representation</td>
<td>469</td>
</tr>
<tr>
<td>trace iteration history</td>
<td>471</td>
</tr>
<tr>
<td>modulus</td>
<td>575</td>
</tr>
<tr>
<td>Moving average</td>
<td>521, 576</td>
</tr>
<tr>
<td>Moving sum</td>
<td>577</td>
</tr>
<tr>
<td>msg</td>
<td>339</td>
</tr>
<tr>
<td>mtos</td>
<td>544</td>
</tr>
<tr>
<td>Multiplication operator (*)</td>
<td>35</td>
</tr>
<tr>
<td>N</td>
<td></td>
</tr>
<tr>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>recode</td>
<td>576</td>
</tr>
<tr>
<td>na</td>
<td>522</td>
</tr>
<tr>
<td>name</td>
<td>340</td>
</tr>
<tr>
<td>Nearest neighbor regression</td>
<td>341</td>
</tr>
<tr>
<td>Negation operator (-)</td>
<td>34</td>
</tr>
<tr>
<td>Negative binomial</td>
<td></td>
</tr>
<tr>
<td>count model</td>
<td>233</td>
</tr>
<tr>
<td>distribution</td>
<td>586</td>
</tr>
<tr>
<td>next</td>
<td>563</td>
</tr>
<tr>
<td>nnfit</td>
<td>341</td>
</tr>
<tr>
<td>Nonlinear least squares</td>
<td>315</td>
</tr>
<tr>
<td>Norm of a matrix</td>
<td>546</td>
</tr>
<tr>
<td>Normal distribution</td>
<td>586</td>
</tr>
<tr>
<td>random number</td>
<td>523</td>
</tr>
<tr>
<td>not equal to comparison</td>
<td>36</td>
</tr>
<tr>
<td>nrmd</td>
<td>523</td>
</tr>
<tr>
<td>Number</td>
<td></td>
</tr>
<tr>
<td>evaluate a string</td>
<td>557</td>
</tr>
<tr>
<td>formatting in tables</td>
<td>49</td>
</tr>
<tr>
<td>Number of observations</td>
<td>578</td>
</tr>
<tr>
<td>object</td>
<td></td>
</tr>
<tr>
<td>assignment</td>
<td>9</td>
</tr>
<tr>
<td>command</td>
<td>6</td>
</tr>
<tr>
<td>containers</td>
<td>11</td>
</tr>
<tr>
<td>copy</td>
<td>14, 228</td>
</tr>
<tr>
<td>create using freeze</td>
<td>275</td>
</tr>
<tr>
<td>declaration</td>
<td>5, 9</td>
</tr>
<tr>
<td>delete</td>
<td>15</td>
</tr>
<tr>
<td>fetch</td>
<td>16</td>
</tr>
<tr>
<td>fetch from database or databank</td>
<td>264</td>
</tr>
<tr>
<td>label</td>
<td>302</td>
</tr>
<tr>
<td>print</td>
<td>373</td>
</tr>
<tr>
<td>rename</td>
<td>15, 380</td>
</tr>
<tr>
<td>save</td>
<td>16</td>
</tr>
<tr>
<td>store</td>
<td>16, 452</td>
</tr>
<tr>
<td>test for existence</td>
<td>563</td>
</tr>
<tr>
<td>Observations</td>
<td></td>
</tr>
<tr>
<td>number in workfile range</td>
<td>564</td>
</tr>
<tr>
<td>OLS (ordinary least squares)</td>
<td>315</td>
</tr>
<tr>
<td>Omitted variables test</td>
<td>461</td>
</tr>
<tr>
<td>One-way frequency table</td>
<td>276</td>
</tr>
<tr>
<td>Open</td>
<td></td>
</tr>
<tr>
<td>database</td>
<td>242</td>
</tr>
<tr>
<td>text or program files</td>
<td>342</td>
</tr>
<tr>
<td>open</td>
<td>342</td>
</tr>
<tr>
<td>Open foreign source data</td>
<td>492</td>
</tr>
<tr>
<td>Operator</td>
<td>573</td>
</tr>
<tr>
<td>options</td>
<td>343</td>
</tr>
<tr>
<td>Or operator</td>
<td>574</td>
</tr>
<tr>
<td>ordered</td>
<td>345</td>
</tr>
<tr>
<td>Ordered dependent variable</td>
<td></td>
</tr>
<tr>
<td>estimating models with</td>
<td>345</td>
</tr>
<tr>
<td>make vector of limit points from equation</td>
<td>326</td>
</tr>
<tr>
<td>Outer product</td>
<td>547</td>
</tr>
<tr>
<td>Output</td>
<td></td>
</tr>
<tr>
<td>display estimation results</td>
<td>347</td>
</tr>
<tr>
<td>extracting results from views</td>
<td>124</td>
</tr>
<tr>
<td>printing</td>
<td>8</td>
</tr>
<tr>
<td>redirection</td>
<td>347</td>
</tr>
<tr>
<td>output</td>
<td>347</td>
</tr>
<tr>
<td>Output redirection</td>
<td>565</td>
</tr>
<tr>
<td>override</td>
<td>349</td>
</tr>
<tr>
<td>Override variables in model solution</td>
<td>349</td>
</tr>
<tr>
<td>P</td>
<td></td>
</tr>
<tr>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>contract</td>
<td>352</td>
</tr>
<tr>
<td>copy from</td>
<td>353</td>
</tr>
<tr>
<td>create new</td>
<td>355</td>
</tr>
<tr>
<td>define structure</td>
<td>363</td>
</tr>
<tr>
<td>delete page</td>
<td>356</td>
</tr>
<tr>
<td>rename</td>
<td>358</td>
</tr>
<tr>
<td>resize</td>
<td>363</td>
</tr>
</tbody>
</table>
save or export 358
set active 360
stack 360
subset from 352
pageappend 350
pagecontract 352
pagecopy 353
pagecreate 355
pagedelete 356
pageload 357
pagename 358
pagesave 358
pageselect 360
pagestack 360
pagestruct 363
pageunstack 366
Panel
        unit root test 481
        param 368
Parameters 368
Pareto distribution 586
Partial autocorrelation 231
Partial correlation 231
pdl 524
PDL (polynomial distributed lag) 524
Percent change 577
Percentage change 577
Permute rows of matrix 547
Phillips-Perron test 481
Pi 583
Pi (constant) 584
pie 370
Pie graph 370
plot 372
poff 565
Poisson
        count model 233
distribution 586
Polynomial distributed lags 524
pon 565
Pool
        add cross section member 179
generate series using identifiers 281
make group of pool series 324
pool 372
Pool (object) 153
data members 155
declare 372
delete identifiers 248
members 153
procs 154
views 154
Portrait (print orientation) 8
Power (raise to) 573
Precedence of evaluation 34
predict 372
Prediction table 372
Presentation table 255
Principal components 369
Print
        and display 8
        automatic printing 565
        landscape 8
        portrait 8
turn off in program 565
print 373
Printing
        automatic printing 565
        probit 374
Probit models 202
Program 83
        abort 86
        arguments 96, 97
call subroutine 110, 559
counting execution errors 561
create 83
create new file 374
entering text 83
exit loop 107
if statement 99
include file 107, 563
line continuation character 84
modes 96
open 84
place subroutine 109
quiet mode 85
run 391
running 84
save 84
stop 106
stop execution 568
verbose mode 85
version 4 compatibility 96
program 374
P-value functions 587

Q

QQ-plot
See Distribution function.
qqplot 375
Q-statistic 231
qstats 376
Quantile function 579
Quantile-Quantile plot. See Distribution function
Quiet mode 85

R

Random number
integer 388
seed 389
uniform 525
Random number generator
normal 523
range 377
Rank 547
Read
data from foreign file 377
read 377
Recode values 576
Recursive least squares 387
CUSUM 387
CUSUM of squares 387
Redirect output to file 8, 347
Redundant variables test 464
Regressors
make group containing 327
Rename
database 244
object 15, 380
rename 380
Repair database 245
Replacement variable 91
naming objects 92
Representations view 381
Resample
observations 381
rows from matrix 548
resample 381
reset 383
RESET test 383
Reset timer 469
residcor 384
residcov 385
resids 385
Residuals
correlation matrix of 384
covariance matrix of 385
display of 385
make series or group containing 328
resize page 363
Restricted VAR text 221
Results
display or retrieve 386
results 386
return 566
Rich Text Format 57
rls 387
rnd 525
rndint 388
rndseed 389
Roots of the AR polynomial 195
Roundoff error in for loops 567
Row
numbers 550
place in matrix 550
Row height 50
rowplace 550
rowvector 390
Rowvector (object)
data members 157
declare 390
extract 550
filled rowvector function 535
views 156
run 391
Run batch program 391
Run program
multiple files 107

S

Sample
change using for loop 101
number of observations 564
set current 436
sample 392
Sample (object)
  declare 392
  procs 158
  set sample specification in 407
sar 525
Save
  commands in file 2
  objects to disk 16
save 393
scalar 396
Scalar (object) 158
  declare 396
scale 397
scat 399
scatmat 401
Scatter diagrams
  matrix of 401
Scatterplot 399
  with kernel fit 301
  with nearest neighbor fit 341
  with regression line fit 308
scenario 401
seas 403
Seasonal adjustment
  moving average 403
  Tramo/Seats 471
  X11 502
  X12 504
Seasonal autoregressive error 525
Seasonal dummy variable 577
Seasonal line graphs 404
seasplot 404
Second moment matrix 537
Seed random number generator 389
Seemingly unrelated regression. See SUR
Sequential LR tests 107
Serial correlation
  Breusch-Godfrey LM test 196
  multivariate VAR LM test 193
Series
  auto-updating 278
  convert to matrix 450, 451, 544, 551
  convert to matrix (with NAs) 552
  extract observation 574
  formula 278
  frequency conversion default method 410
  series 405
Series (object) 159
  data members 160
  declare 405
  element function 160
  fill values 267
  views 159
set 407
Set graph date labeling formats 237, 239
setbpelem 407
setcell 408
setcolwidth 410
setconvert 410
setelem 412
setfillcolor 416
setfont 418
setformat 419
setheight 423
setindent 424
setjust 425
setline 426
setlines 427
setmerge 428
settextcolor 430
setwidth 431
Shade region of graph 252
sheet 432
show 432
Show object view 432
Signal variables
  display graphs 433
  saving 330
signalgraph 433
Sine 584
Singular matrix
  test for 540
Singular value decomposition 555
sma 526
smooth 434
Smoothing
  exponential smooth series 434
  signal series 330
  state series 331
smpl 436
Solve
  linear system 551
  simultaneous equations model 438
solve 438
Solve. See Models.
solveopt 439
sort 440
Sort workfile 440
spec 441
Specification 381
Specification view 441
spike 442
Spike graph 442
Spreadsheet view 432
sqrt 576
sspace 444
Sspace (object)
  append specification line 186
  data members 162
  declare 444
  display signal graphs 433
  make Kalman filter from 321
  method 161
  procs 162
  specification display 454
  state graphs 446
  views 161
Stack matrix by column 558
  lower triangle 558
Stack workfile page 360
Standard deviation 579
Starting values 368
statby 444
State space
  specification 454
State variables
  display graphs of 446
  final one-step ahead predictions 447
  initial values 448
  smoothed series 331
statefinal 447
stategraphs 446
stateinit 448
Static forecast 270
Statistical distribution functions 584
Statistics 250
  compute for subgroups 444
stats 449
Status line 450
statusline 450, 566
step 567
stom 450, 551
stomna 451, 552
stop 568
Stop program execution 106, 568
store 452
Store object in database or databank 16, 452
String 89
  assign to table cell 48
  convert date value into 531
  convert from a number 553
  convert into date value 532
  convert to a scalar 557
  find substring in 539
  insert string into 538
  length 541
  length of 553
  lowercase 542
  replace substring in 548
  substring 541, 549
  test for blank 540
  test for equality 534
  test for inequality 545
  trim spaces from ends 542, 551, 556
  uppercase 557
String variable 89
  as replacement variable 91
  convert to a scalar 90
  in for loop 103
  program arguments 96, 97
structure 454
Structure workfile page 363
Subroutine 108
  arguments 109
  call 110, 559
  declare 568
  define 108
  global 111
  local 113
  mark end 560
  placement 109
  return from 108, 566
The text is too large to display here in a readable format.
omitted variables 461
redundant variables 464
RESET 383
simple mean, median, variance hypotheses 468
unit root 481
Wald 490
testadd 461
testbtw 462
testby 462
testdrop 464
testexog 465
testfit 466
testlags 467
teststat 468
text 469
Text (object) 170
declare 469
views 170
Then 569
Three stage least squares 178
tic 469
Time
  current as string 554
Time trend 577
Timer 469, 470, 570
to 570
Tobit models 217
toc 470
trace 471
Trace of a matrix 555
Tramo/Seats 471
tramoses 471
Transpose 556
Trend series 577
Trigonometric functions 583
Truncated dependent variable models 217
tslls 474
Two-stage least squares
  see 2sls

U
Uniform distribution 587
Uniform random number generator 525
Unit root test 481
Unit vector 556
unlink 479
Unstack workfile page 366
Untitled objects 11
update 480
updatecoefs 480
Uppercase 557
uroot 481
usage 486

V
Valmap
  apply to series 334
valmap 487
Valmap (object)
    append specification line 186
declare 487
find series that use map 486
procs 171
views 171
VAR
  estimate factorization matrix 456
  impulse response 296
  lag exclusion test 467
  lag length test 303
  multivariate autocorrelation test 376
  variance decomposition 245
var 487
VAR (object) 171
data members 172
methods 171
procs 172
views 171
Var (object)
  append specification line 186
clear restrictions 221
declare 487
Variance 579
equality test 462
Variance decomposition 245
vars 488
VEC
  estimating 256
vector 489
Vector (object) 174
data members 175
declare 489
procs 175
return filled 536
views 174
Vector autoregression
See VAR.
Verbose mode 85
Views
extracting results from 124

W
wald 490
Wald test 490
Watson test 257
Weibull distribution 587
Weighted least squares 497
Weighted two-stage least squares 501
wend 571
wcreate 491
wfopen 492
wfsave 493
wfselect 495
while 571

While loop 104
abort 105
end of 571
exit loop 107
start of 571
white 495
Wildcards 14

wls 497

Workfile 11, 363
append contents of workfile page to current
page 350
close 12
contract page 352
convert repeated obs to repeated series 366
convert repeated series to repeated obs 360
copy from page 353
create 235
create new workfile 491
create page in 355
define structured page 363
delete page 356
expand 264
export page 493
frequency 11
load workfile pages into 357
open existing 12, 492
open foreign source into 492
panel to pool 366
pool page to panel page 360
rename page 358
save 12, 493
save or export page 358
set active page 360
set active workfile and page 495
sort observations 440
stack page 360
unstack page 366
workfile 498
Write
data to text file 498
write 498
wtsls 501

X
x11 502
x12 504
xy 510
XY (line) graph 512
XY (pairs) graph 515
XY graph 510
xyline 512
xypair 515