

EViews 6 Command Reference



Quantitative Micro Software

EViews 6 Command Reference

Copyright © 1994–2007 Quantitative Micro Software, LLC

All Rights Reserved

Printed in the United States of America

This software product, including program code and manual, is copyrighted, and all rights are reserved by Quantitative Micro Software, LLC. The distribution and sale of this product are intended for the use of the original purchaser only. Except as permitted under the United States Copyright Act of 1976, no part of this product may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of Quantitative Micro Software.

Disclaimer

The authors and Quantitative Micro Software assume no responsibility for any errors that may appear in this manual or the EViews program. The user assumes all responsibility for the selection of the program to achieve intended results, and for the installation, use, and results obtained from the program.

Trademarks

Windows, Windows 95/98/2000/NT/Me/XP, and Microsoft Excel are trademarks of Microsoft Corporation. PostScript is a trademark of Adobe Corporation. X11.2 and X12-ARIMA Version 0.2.7 are seasonal adjustment programs developed by the U. S. Census Bureau. Tramo/Seats is copyright by Agustin Maravall and Victor Gomez. All other product names mentioned in this manual may be trademarks or registered trademarks of their respective companies.

Quantitative Micro Software, LLC

4521 Campus Drive, #336, Irvine CA, 92612-2621

Telephone: (949) 856-3368

Fax: (949) 856-2044

e-mail: sales@eviews.com

web: www.eviews.com

January 9, 2007

Table of Contents

PREFACE	1
CHAPTER 1. OBJECT VIEW AND PROCEDURE REFERENCE	3
Alpha	5
Coef	15
Equation	27
Factor	97
Graph	143
Group	183
Link	223
Logl	233
Matrix	247
Model	271
Pool	295
Rowvector	337
Sample	349
Scalar	353
Series	355
Spool	409
Sspace	427
Sym	453
System	475
Table	509
Text	533
Valmap	537
Var	543
Vector	575
CHAPTER 2. OBJECT SUMMARY FOR COMMANDS	591
CHAPTER 3. GRAPH CREATION COMMANDS	601
Optional Graph Components	668

CHAPTER 4. COMMAND REFERENCE	675
CHAPTER 5. SPECIAL EXPRESSION REFERENCE	815
CHAPTER 6. STRING AND DATE FUNCTION REFERENCE	823
CHAPTER 7. MATRIX LANGUAGE REFERENCE	839
CHAPTER 8. PROGRAMMING LANGUAGE REFERENCE	869
APPENDIX A. OPERATOR AND FUNCTION LISTING	883
INDEX	895

Preface

The EViews 6 *Command Reference* contains reference material for working with commands in EViews.

The material is divided into several sections. The first two chapters document the commands used for working with EViews objects. The next two chapters describe expressions, operators, and functions that are used as building blocks in estimation or for working with data. Together, the first four chapters constitute the core of the reference material:

- [Chapter 1. “Object View and Procedure Reference,” on page 3](#) provides a cross-referenced listing of commands, views, and procedures associated with each object.
- [Chapter 2. “Object Summary for Commands,” on page 591](#) offers an alternative indexing of the object views and procedures described in the previous chapters, pairing each entry with a list of the objects to which it may be applied.
- [Chapter 3. “Graph Creation Commands,” on page 601](#) documents the specialized object view commands for producing graph views from various EViews data objects.
- [Chapter 4. “Command Reference,” on page 675](#) provides a full description of auxiliary and interactive commands.
- [Chapter 5. “Special Expression Reference,” on page 815](#) special expressions that may be used in series assignment and series generation, or as terms in estimation specifications.

The remaining chapters describe the specialized commands, keywords and functions used by the EViews string and date libraries, and the matrix and programming languages:
























- [Chapter 6. “String and Date Function Reference,” on page 823](#) describes the syntax of the functions used when working with strings and dates in EViews.
- [Chapter 7. “Matrix Language Reference,” on page 839](#) is an alphabetical listing of the commands and functions associated with the EViews matrix language.
- [Chapter 8. “Programming Language Reference,” on page 869](#) contains an alphabetical listing of the keywords and functions associated with the EViews programming language.
- [Appendix A. “Operator and Function Listing,” on page 883](#) lists the operators and functions that may be used with series and (in some cases) matrix objects. A more detailed description of these operators and functions is available in the *User’s Guide I*.

Note also that the last few chapters of *User’s Guide I* contain additional material that should be of interest when working with commands. Specifically:

- [Chapter 16. “Object and Command Basics,” on page 577](#) provides additional background on the use of commands in EViews.
- [Chapter 17. “EViews Programming,” on page 593](#) describes the basics of using programs for batch processing and documents the programming language.
- [Chapter 18. “Matrix Language,” beginning on page 627](#) describes the EViews matrix language.
- [Chapter 19. “Working with Graphs,” on page 651](#) describes the use of commands to customize graph objects.
- [Chapter 20. “Working with Tables,” beginning on page 675](#) documents the table object and describes the basics of working with tables in EViews.
- [Chapter 21. “Working with Spools,” beginning on page 687](#) discusses commands for working with spools.
- [Chapter 22. “Strings and Dates,” on page 695](#) describes the syntax and functions available for manipulating text strings and dates.
- [Chapter 23. “Workfile Functions,” beginning on page 727](#) describes special functions for obtaining information about observations in the workfile.
- [Appendix A. “Operator and Function Reference,” on page 733](#) describes the operators and functions that may be used with series and (in some cases) matrix objects.

Chapter 1. Object View and Procedure Reference

This chapter contains is a reference guide to the views, procedures, and data members for each of the objects found in EViews:

 Alpha (p. 5)	 Model (p. 271)	 Sym (p. 453)
 Coef (p. 15)	 Pool (p. 295)	 System (p. 475)
 Equation (p. 27)	 Rowvector (p. 337)	 Table (p. 509)
 Factor (p. 97)	 Sample (p. 349)	 Text (p. 533)
 Graph (p. 143)	 Scalar (p. 353)	 Valmap (p. 537)
 Group (p. 183)	 Series (p. 355)	 Var (p. 543)
 Logl (p. 233)	 Spool (p. 409)	 Vector (p. 575)
 Matrix (p. 247)	 Sspace (p. 427)	

To use these views, procedures, and data members, you should provide an optional action (described below), then list the name of the object followed by a period, and then the name of the method, view, procedure, or data member, along with any options or arguments:

object_name.method_name(options) arguments

object_name.view_name(options) arguments

object_name.proc_name(options) arguments

output_name = object_name.data_member

The first three types of expressions are collectively referred to as object commands. An *object command* is a command which displays a view of or performs a procedure using a specific object. Object commands have two main parts: an *action* followed by a *view or procedure specification*. The display action determines what is to be done with the output from the view or procedure. The view or procedure specification may provide for options and arguments to modify the default behavior.

The complete syntax for an object command has the form:

action (action_opt) object_name.view_or_proc(options_list) arg_list

where:

actionis one of the four verb commands (do, freeze, print, show).

action_optan 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 performed.

options_list an option that modifies the default behavior of the view or procedure.

arg_list a list of view or procedure arguments.

The four possible action commands behave as follows:

- `show` displays the object view in a window.
- `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.

In most cases, you need not specify an action explicitly. If no action is provided, the `show` action is assumed for views and the `do` action is assumed for most procedures (though some procedures will display newly created output in windows unless run in a batch program).

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
```

See [Chapter 16. “Object and Command Basics,” on page 577](#) of the *User's Guide I* for additional discussion of using commands in EViews.

Alpha

Alpha (alphanumeric) series. An EViews alpha series contains a set of observations on a variable containing string values.

Alpha Declaration

alphadeclare alpha series (p. 6).
frmlcreate alpha series object with a formula for auto-updating (p. 8).
genrcreate alpha or numeric series object (p. 10).

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

freqone-way tabulation (p. 7).
labellabel information for the alpha (p. 10).
sheetspreadsheet view of the alpha (p. 13).

Alpha Procs

displaynameset display name (p. 7).
makemapcreate numeric classification series and valmap from alpha series (p. 11).
mapassign or remove value map setting (p. 12).
setindentset the indentation for the alpha series spreadsheet (p. 12).
setjustset the justification for the alpha series spreadsheet (p. 13).

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:

```
alpha name = first + " " + last
```

creates an alpha series containing the full names.

Alpha Entries

The following section provides an alphabetical listing of the commands associated with the “Alpha” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

alpha	Alpha Declaration
-------	-----------------------------------

Declare an alpha series object.

The `alpha` command creates and optionally initializes an alpha series or modifies an existing series.

Syntax

```
alpha ser_name  
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 the text “initial value.”

Once an alpha is declared, you need not include the `alpha` keyword prior to entering the formula (alternatively, you may use [Alpha::genr](#) (p. 10)). 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 and LAST are alpha series containing first and last names, respectively, the commands:

```
alpha name = first + " " + last  
genr name = name + " " + last
```

create an alpha series containing the full names.

Cross-references

See [“Alpha Series” on page 150](#) of the *User’s Guide I* for additional discussion.

See also [Alpha::genr](#) (p. 10).

displayname	Alpha Procs
-------------	-----------------------------

Display name for an alpha object.

Attaches a display name to an alpha object which may be used to label output in tables and graphs in place of the standard alpha object name.

Syntax

```
alpha_name.displayname display_name
```

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in alpha object names.

Examples

```
names.displayname Employee Name
names.label
```

The first line attaches a display name “Employee Name” to the alpha object NAMES, and the second line displays the label view of NAMES, including its display name.

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels and display names. See also [Alpha::label](#) (p. 10).

freq	Alpha Views
------	-----------------------------

Compute frequency tables.

`freq` performs a one-way frequency tabulation. The options allow you to control binning (grouping) of observations.

Syntax

```
alpha_name.freq(options)
```

Options

dropna (<i>default</i>) / keepna	[Drop/Keep] NA as a category.
n, obs, count (<i>default</i>)	Display frequency counts.
nocount	Do not display frequency counts.
p	Print the table.
total (<i>default</i>) / nototal	[Display / Do not display] totals.
pct (<i>default</i>) / nopct	[Display / Do not display] percent frequencies.
cum (<i>default</i>) / nocum	(Display/Do not) display cumulative frequency counts/percentages.

Examples

```
names.freq
```

tabulates each value of NAMES in ascending order with counts, percentages, and cumulatives.

Cross-references

See [“One-Way Tabulation” on page 323](#) of the *User’s Guide I* for a discussion of frequency tables.

frml	Alpha Declaration
------	-----------------------------------

Declare an alpha series object with a formula for auto-updating, or specify a formula for an existing alpha series.

Syntax

```
frml alpha_name = alpha_expression
frml alpha_name = @clear
```

Follow the `frml` keyword with a name for the alpha series, and an assignment statement. The special keyword “@CLEAR” is used to return the auto-updating series to an alpha series.

Examples

To define an auto-updating alpha series, you must use the `frml` keyword prior to entering an assignment statement. 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 alpha series. The commands:

```
alpha state_info
frml state_info = state_name + state_id
```

makes the previously created alpha series `STATE_INFO` an auto-updating series containing the alpha series `STATE_NAME` and `STATE_ID`. Note that once an alpha series is defined to be auto-updating, it may not be modified directly. Here, you may not edit `STATE_INFO`, nor may you generate data into the alpha series.

Note that the commands:

```
alpha state_info
state_info = state_name + state_id
```

while similar, produce quite different results, since the absence of the `frml` keyword in the second example means that EViews will generate fixed values in the alpha series instead of defining a formula to generate the alpha series values. In this latter case, the values in the alpha series `STATE_INFO` are fixed, and may be modified.

One particularly useful feature of auto-updating series is the ability to reference series in databases. The command:

```
frml states = usdata::states
```

creates an alpha series called `STATES` that obtains its values from the alpha series `STATES` in the database `USDATA`.

To turn off auto-updating for an alpha series, you should use the special expression “@CLEAR” in your `frml` assignment. The command:

```
frml id = @clear
```

sets the series to alpha value format, freezing the contents of the series at the current values.

Cross-references

See “[Auto-Updating Series](#)” on page 145 of the *User’s Guide I*.

See also [Link::link](#) (p. 225).

genr	Alpha Declaration
------	-----------------------------------

Generate alpha series.

Syntax

`genr alpha_name = expression`

Examples

`genr full_name = first_name + last_name`

creates an alpha series formed by concatenating the alpha series FIRST_NAME and LAST_NAME.

Cross-references

See [Alpha::alpha \(p. 6\)](#) for a discussion of the expressions allowed in `genr`.

label	Alpha Views Alpha Procs
-------	---

Display or change the label view of an alpha series, including the last modified date and display name (if any).

As a procedure, label changes the fields in the alpha series label.

Syntax

`alpha_name.label`
`alpha_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 <i>text</i> .
s	Sets the source field to <i>text</i> .
u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of ALPHA1 with “Data from CPS 1988 March File”:

```
alpha1.label(r)
alpha1.label(r) Data from CPS 1988 March File
```

To append additional remarks to ALPHA1, and then to print the label view:

```
alpha1.label(r) Hourly notes
alpha1.label(p)
```

Cross-references

See “[Labeling Objects](#)” on [page 72](#) of the *User’s Guide I* for a discussion of labels.

See also [Alpha::displayname](#) ([p. 7](#)).

makemap	Alpha Procs
----------------	-----------------------------

Create numeric classification series and valmap from alpha series.

Syntax

```
alpha_name.makemap(options) ser_name map_name
```

creates a classification series *ser_name* and an associated valmap *map_name* in the workfile. The valmap will automatically be assigned to the series.

Options

<code>nosort</code>	Do not sort the alpha series values in alphabetical order before assigning the map (default is to sort).
---------------------	--

Examples

```
stateabbrev.makemap statecodes statemap
```

creates a series STATECODES containing numeric coded values representing the states in STATEABBREV, and an associated valmap STATEMAP.

Cross-references

See “[Alpha Series](#)” on [page 150](#) of the *User’s Guide I* for a discussion of alpha series. See “[Value Maps](#)” on [page 159](#) of the *User’s Guide I* for a discussion of valmaps.

map	Alpha Procs
-----	-----------------------------

Assign or remove value map setting.

Syntax

`alpha_name.map [valmap_name]`

If the optional valmap name is provided, the procedure will assign the specified value map to the alpha series. 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.

`alpha2.map`

removes an existing valmap assignment from ALPHA2.

Cross-references

See [“Value Maps” on page 159](#) of the *User’s Guide I* for a discussion of valmap objects in EViews.

setindent	Alpha Procs
-----------	-----------------------------

Set the display indentation for cells in an alpha series spreadsheet view.

Syntax

`alpha_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 spreadsheet (the EViews spreadsheet default font at the time the spreadsheet 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 spreadsheet view is created.

Examples

To set the justification for an alpha series object to 2/5 of a width unit:

`alpha1.setindent 2`

Cross-references

See [Alpha::setjust \(p. 13\)](#) for details on setting spreadsheet justification.

setjust	Alpha Procs
---------	-----------------------------

Set the display justification for cells in an alpha series spreadsheet view.

Syntax

```
alpha_name.setjust just_arg
```

where *just_arg* is a set of arguments used to specify justification settings.

The *just_arg* may be formed using the following:

top / middle / bottom]	Vertical justification setting.
auto / left / cen- ter / right	Horizontal justification setting. “Auto” uses left justification.

You may enter one or both of the justification settings. The default settings are taken from the Global Defaults for spreadsheet views.

Examples

```
a1.setjust middle
```

sets the vertical justification to the middle.

```
a1.setjust top left
```

sets the vertical justification to top and the horizontal justification to left.

Cross-references

See [Alpha::setindent \(p. 12\)](#) for details on setting spreadsheet indentation.

sheet	Alpha Views
-------	-----------------------------

Spreadsheet view of an alpha series object.

Syntax

```
alpha_name.sheet(options)
```

Options

p	Print the spreadsheet view.
---	-----------------------------

Examples

```
al.sheet
```

displays the spreadsheet view of the alpha series A1.

Cross-references

See [“Alpha Series,” beginning on page 150](#) of the *User’s Guide I* for a discussion of the spreadsheet view of alpha series.

Coef

Coefficient vector. Coefficients are used to represent the parameters of equations and systems.

Coef Declaration

[coef](#).....declare coefficient vector ([p. 16](#)).

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:

```
coef alpha
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. 761](#)) 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

[label](#)label view ([p. 19](#)).

[sheet](#).....spreadsheet view of the coefficient ([p. 24](#)).

[stats](#).....descriptive statistics ([p. 25](#)).

Coef Graph Views

Graph creation types are discussed in detail in “[Graph Creation Commands](#)” on [page 601](#).

[area](#)area graph ([p. 603](#)).

[bar](#).....bar graph ([p. 609](#)).

[boxplot](#)boxplot graph ([p. 613](#)).

[distplot](#)distribution graph ([p. 615](#)).

[dot](#).....dot plot graph ([p. 622](#)).

[line](#).....line graph ([p. 630](#)).

[qqplot](#)quantile-quantile graph ([p. 636](#)).

[seasplot](#).....seasonal line graph ([p. 651](#)).

[spike](#).....spike graph ([p. 652](#)).

Coef Procs

[displayname](#).....set display name ([p. 17](#)).

[fill](#).....fill the elements of the coefficient vector ([p. 18](#)).

`read` import data into coefficient vector (p. 20).
`setformat` set the display format for the coefficient vector spreadsheet (p. 21).
`setindent` set the indentation for the coefficient spreadsheet (p. 22).
`setjust` set the justification for the coefficient spreadsheet (p. 23).
`setwidth` set the column width for the coefficient spreadsheet (p. 24).
`write` export data from coefficient vector (p. 25).

Coef Data Members

(i) *i*-th element of the coefficient vector. Simply append “(i)” to the coef 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 10×1 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.

Coef Entries

The following section provides an alphabetical listing of the commands associated with the “Coef” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

coef	Coef Declaration
-------------	----------------------------------

Declare a coefficient (column) vector.

Syntax

```
coef(n) coef_name
```

Follow the `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

```
coef(2) slope
ls lwage = c(1)+slope(1)*edu+slope(2)*edu^2
```

The first line declares a `coef` object of length 2 named `SLOPE`. The second line estimates a least squares regression and stores the estimated slope coefficients in `SLOPE`.

```
arch(2,2) sp500 c
coef beta = c
coef(6) beta
```

The first line estimates a GARCH(2,2) model using the default `coef` vector `C` (note that 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. 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 [Vector::vector](#) (p. 587).

displayname	Coef Procs
-------------	----------------------------

Display name for a coefficient vector.

Attaches a display name to a `coef` object which may be used to label output in tables and graphs in place of the standard `coef` object name.

Syntax

```
coef_name.displayname display_name
```

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in `coef` object names.

Examples

```
c1.displayname Hours Worked
c1.label
```

The first line attaches a display name “Hours Worked” to the `coef` object `C1`, and the second line displays the label view of `C1`, including its display name.

```
c1.displayname Means by State
plot c1
```

The first line attaches a display name “Means by State” to the coef C1. The line graph view of C1 will use the display name as the legend.

Cross-references

See “[Labeling Objects](#)” on page 72 of the *User’s Guide I* for a discussion of labels and display names. See also `Coef::label` (p. 19).

fill	Coef Procs
------	----------------------------

Fill a coef object with specified values.

Syntax

```
coef_name.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.*

Running out of values before the coef vector is completely filled is not an error; the remaining cells or observations will be unaffected, unless the “1” option is specified. If, however, you list more values than the coef vector can hold, EViews will not modify any observations and will return an error message.

Options

1	Loop repeatedly over the list of values as many times as it takes to fill the coef vector.
<code>o = integer</code> (default = 1)	Fill the coef vector from the specified element. Default is the first element.

Examples

The following example declares a four element coefficient vector MC, initially filled with zeros. The second line fills MC with the specified values and the third line replaces from row 3 to the last row with -1.

```
coef(4) mc
mc.fill 0.1, 0.2, 0.5, 0.5
mc.fill(o=3,1) -1
```

Note that the last argument in the fill command above is the *letter* “1”.

Cross-references

See [“Fill assignment” on page 629](#) of the *User’s Guide I* for further discussion of the fill procedure.

label	Coef Views Coef Procs
-------	---

Display or change the label view of the coefficient vector, including the last modified date and display name (if any).

As a procedure, `label` changes the fields in the coef object label.

Syntax

`coef_name.label`

`coef_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 <i>text</i> .
s	Sets the source field to <i>text</i> .
u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of the coefficient vector C1 with “Results from EQ3”:

```
c1.label(r)
c1.label(r) Results from EQ3
```

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels.

See also [Coef::displayname \(p. 17\)](#).

read	Coef Procs
------	----------------------------

Import data from a foreign disk file into a coefficient vector.

May be used to import data into an existing workfile from a text, Excel, or Lotus file on disk.

Syntax

`coef_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. Path specifications may point to local or network drives. If the path specification contains a space, you should enclose the entire expression in double quotation marks.

Options

File type options

<code>t = dat, txt</code>	ASCII (plain text) files.
<code>t = wk1, wk3</code>	Lotus spreadsheet files.
<code>t = xls</code>	Excel spreadsheet files.

If you do not specify the “t” option, EViews uses the file name extension to determine the file type. If you specify the “t” option, the file name extension will not be used to determine the file type.

Options for ASCII text files

<code>na = text</code>	Specify text for NAs. Default is “NA”.
<code>d = t</code>	Treat tab as delimiter (note: you may specify multiple delimiter options). The <i>default</i> is “d = c” only.
<code>d = c</code>	Treat comma as delimiter.
<code>d = s</code>	Treat space as delimiter.
<code>d = a</code>	Treat alpha numeric characters as delimiter.
<code>custom = symbol</code>	Specify symbol/character to treat as delimiter.
<code>mult</code>	Treat multiple delimiters as one.
<code>rect (default) / norect</code>	[Treat / Do not treat] file layout as rectangular.
<code>skipcol = integer</code>	Number of columns to skip. Must be used with the “rect” option.

<code>skiprow = integer</code>	Number of rows to skip. Must be used with the “rect” option.
<code>comment = symbol</code>	Specify character/symbol to treat as comment sign. Everything to the right of the comment sign is ignored. Must be used with the “rect” option.
<code>singlequote</code>	Strings are in single quotes, not double quotes.
<code>dropstrings</code>	Do not treat strings as NA; simply drop them.
<code>negparen</code>	Treat numbers in parentheses as negative numbers.
<code>allowcomma</code>	Allow commas in numbers (note that using commas as a delimiter takes precedence over this option).

Options for spreadsheet (Lotus, Excel) files

<code>letter_number</code> (<i>default</i> = “b2”)	Coordinate of the upper-left cell containing data.
<code>s = sheet_name</code>	Sheet name for Excel 5–8 Workbooks.

Examples

```
c1.read(t=dat,na=.) a:\mydat.raw
```

reads data into coefficient vector C1 from an ASCII file MYDAT.RAW in the A: drive. The missing value NA is coded as a “.” (dot or period).

```
c1.read(s=sheet2) "\\network\dr 1\cps91.xls"
```

reads the Excel file CPS91 into coefficient vector C1 from the network drive specified in the path.

Cross-references

See [“Importing Data” on page 95](#) of the *User’s Guide I* for a discussion and examples of importing data from external files.

For powerful, easy-to-use tools for reading data into a new workfile, see [“Creating a Workfile by Reading from a Foreign Data Source” on page 41](#) of the *User’s Guide I* and [param \(p. 761\)](#).

See also [Coef::write \(p. 25\)](#).

setformat	Coef Procs
-----------	----------------------------

Set the display format for cells in coefficient vector spreadsheet views.

Syntax

```
coef_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.

For coefficient vectors, `setformat` operates on all of the cells in the vector.

You should use one of the following format specifications:

<code>g[.precision]</code>	significant digits
<code>f[.precision]</code>	fixed decimal places
<code>c[.precision]</code>	fixed characters
<code>e[.precision]</code>	scientific/float
<code>p[.precision]</code>	percentage
<code>r[.precision]</code>	fraction

In order to set a format that groups digits into thousands, place a “t” after a specified format. For example, to obtain a fixed number of decimal places, with commas used to separate thousands, use “ft[.precision]”. To obtain a fixed number of characters with a period used to separate thousands, use “ct[.precision]”.

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)”).

Examples

To set the format for all cells in the coefficient vector to fixed 5-digit precision, simply provide the format specification:

```
c1.setformat f.5
```

Other format specifications include:

```
c1.setformat f(.7)
```

```
c1.setformat e.5
```

Cross-references

See [Coef::setWidth \(p. 24\)](#), [Coef::setindent \(p. 22\)](#), and [Coef::setjust \(p. 23\)](#) for details on setting spreadsheet widths, indentation and justification.

setindent	Coef Procs
-----------	----------------------------

Set the display indentation for cells in coefficient vector spreadsheet views.

Syntax

```
coef_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 spreadsheet (the EViews spreadsheet default font at the time the spreadsheet 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 spreadsheet view is created.

Examples

To set the justification for a coef object to 2/5 of a width unit:

```
c1.setindent 2
```

Cross-references

See [Coef::setWidth \(p. 24\)](#) and [Coef::setjust \(p. 23\)](#) for details on setting spreadsheet widths and justification.

setjust	Coef Procs
---------	----------------------------

Set the display justification for cells in coefficient vector spreadsheet views.

Syntax

```
coef_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 *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 taken from the Global Defaults for spreadsheet views.

Examples

```
c1.setjust middle
```

sets the vertical justification to the middle.

```
c1.setjust top left
```

sets the vertical justification to top and the horizontal justification to left.

Cross-references

See [Coef::setwidth](#) (p. 24) and [Coef::setindent](#) (p. 22) for details on setting spreadsheet widths and indentation.

setwidth	Coef Procs
----------	----------------------------

Set the column width in a coefficient object spreadsheet view.

Syntax

```
coef_name.setwidth width_arg
```

where *width_arg* specifies the width unit value. The width unit is computed from representative characters in the default font for the current spreadsheet (the EViews spreadsheet default font at the time the spreadsheet 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

```
c1.setwidth 12
```

sets the width of the coef to 12 width units.

Cross-references

See [Coef::setindent](#) (p. 22) and [Coef::setjust](#) (p. 23) for details on setting indentation and justification.

sheet	Coef Views
-------	----------------------------

Spreadsheet view of a coefficient vector.

Syntax

```
coef_name.sheet(options)
```

Options

p	Print the spreadsheet view.
---	-----------------------------

Examples

```
c01.sheet
```

displays the spreadsheet view of C01.

stats	Coef Views
--------------	----------------------------

Descriptive statistics.

Computes and displays a table of means, medians, maximum and minimum values, standard deviations, and other descriptive statistics for the data in the coef object.

Syntax

```
coef_name.stats(options)
```

Options

p	Print the stats table.
---	------------------------

Examples

```
c1.stats(p)
```

displays and prints the descriptive statistics view of the coefficient vector C1.

Cross-references

See [“Descriptive Statistics & Tests” on page 306](#) and [page 379](#) of the *User’s Guide I* for a discussion of descriptive statistics views.

write	Coef Procs
--------------	----------------------------

Write EViews data to a text (ASCII), Excel, or Lotus file on disk.

Creates a foreign format disk file containing data in a coefficient vector object. May be used to export EViews data to another program.

Syntax

```
coef_name.write(options) [path\filename]
```

Follow the name of the coef object by a period, the keyword, and the name for the output file. 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. The entire coef will be exported.

Note that EViews cannot, at present, write into an existing file. The file that you select will, if it exists, be replaced.

Options

Options are specified in parentheses after the keyword and are used to specify the format of the output file.

File type

<code>t = dat, txt</code>	ASCII (plain text) files.
<code>t = wk1, wk3</code>	Lotus spreadsheet files.
<code>t = xls</code>	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

<code>na = string</code>	Specify text string for NAs. Default is “NA”.
<code>d = arg</code>	Specify delimiter (<i>default</i> is tab): “s” (space), “c” (comma).

Spreadsheet (Lotus, Excel) files

<code>letter_number</code>	Coordinate of the upper-left cell containing data.
----------------------------	--

Examples

```
c1.write(t=txt,na=.) a:\dat1.csv
```

writes the coefficient vector C1 into an ASCII file named DAT1.CSV on the A: drive. NAs are coded as “.” (dot).

```
c1.write(t=txt,na=.) dat1.csv
```

writes the same file in the default directory.

```
c1.write(t=xls) "\\network\drive a\results"
```

saves the contents of C1 in an Excel file “Results.xls” in the specified directory.

Cross-references

See [“Exporting to a Spreadsheet or Text File” on page 105](#) of the *User’s Guide I* for a discussion. See also [Coef::read \(p. 20\)](#).

Equation

Equation object. Equations are used for single equation estimation, testing, and forecasting.

Equation Declaration

equation.....declare equation object (p. 47).

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. 31).

binary.....binary dependent variable models (includes probit, logit, gompit) models (p. 37).

censored.....censored and truncated regression (includes tobit) models (p. 40).

count.....count data modeling (includes poisson, negative binomial and quasi-maximum likelihood count models) (p. 44).

gmm.....generalized method of moments (p. 54).

logit.....logit (binary) estimation (p. 62).

ls.....linear and nonlinear least squares regression (includes weighted least squares and ARMAX) models (p. 62).

ordered.....ordinal dependent variable models (includes ordered probit, ordered logit, and ordered extreme value models) (p. 71).

probit.....probit (binary) estimation (p. 74).

qreg.....estimate a quantile regression specification (p. 74).

steps.....estimation by stepwise least squares (p. 85).

tsls.....linear and nonlinear two-stage least squares (TSLS) regression models (includes weighted TSLS, and TSLS with ARMA errors) (p. 88).

Equation Views

archtest.....LM test for the presence of ARCH in the residuals (p. 34).

arma.....Examine ARMA structure of estimated equation (p. 35).

auto.....Breusch-Godfrey serial correlation Lagrange Multiplier (LM) test (p. 36).

cellipse.....Confidence ellipses for coefficient restrictions (p. 39).

chow.....Chow breakpoint and forecast tests for structural change (p. 41).

coefcov	coefficient covariance matrix (p. 42).
correl	correlogram of the residuals (p. 43).
correlsq	correlogram of the squared residuals (p. 44).
derivs	derivatives of the equation specification (p. 46).
facbreak	factor breakpoint test for stability (p. 48).
fixedtest	test significance of estimates of fixed effects for panel estimators (p. 51).
garch	conditional standard deviation graph (only for equations estimated using ARCH) (p. 53).
grads	examine the gradients of the objective function (p. 58).
hettest	test for heteroskedasticity (p. 59).
hist	histogram and descriptive statistics of the residuals (p. 60).
label	label information for the equation (p. 61).
means	descriptive statistics by category of the dependent variable (only for binary, ordered, censored and count equations) (p. 70).
output	table of estimation results (p. 72).
predict	prediction (fit) evaluation table (only for binary and ordered equations) (p. 73).
qrprocess	display table or graph of quantile process estimates (p. 76).
qrslope	test of equality of slope coefficients across multiple quantile regression estimates (p. 78).
qrsymm	test of coefficients using symmetric quantiles (p. 79).
ranhaus	Hausman test for correlation between random effects and regressors (p. 81).
representations	text showing specification of the equation (p. 82).
reset	Ramsey's RESET test for functional form (p. 82).
resids	display, in tabular form, the actual and fitted values for the dependent variable, along with the residuals (p. 83).
results	table of estimation results (p. 83).
rls	recursive residuals least squares (only for non-panel equations estimated by ordinary least squares, without ARMA terms) (p. 84).
testadd	likelihood ratio test for adding variables to equation (p. 86).
testdrop	likelihood ratio test for dropping variables from equation (p. 87).
testfit	performs Hosmer and Lemeshow and Andrews goodness-of-fit tests (only for equations estimated using binary) (p. 88).
ubreak	Andrews-Quandt test for unknown breakpoint (p. 92).
wald	Wald test for coefficient restrictions (p. 93).
white	White test for heteroskedasticity (p. 94).

Equation Procs

- [displayname](#).....set display name (p. 47).
- [fit](#)static forecast (p. 49).
- [forecast](#)dynamic forecast (p. 52).
- [makederivs](#)make group containing derivatives of the equation specification (p. 65).
- [makegarch](#)create conditional variance series (only for ARCH equations) (p. 66).
- [makegrads](#)make group containing gradients of the objective function (p. 67).
- [makelimits](#)create vector of estimated limit points (only for ordered models) (p. 68).
- [makemodel](#)create model from estimated equation (p. 68).
- [makeregs](#)make group containing the regressors (p. 69).
- [makersids](#)make series containing residuals from equation (p. 69).
- [updatecoefs](#)update coefficient vector(s) from equation (p. 93).

Equation Data Members

Scalar Values

- [@aic](#).....Akaike information criterion.
- [@cofcov\(i,j\)](#)covariance of coefficient estimates i and j .
- [@coefs\(i\)](#) i -th coefficient value.
- [@dw](#).....Durbin-Watson statistic.
- [@f](#) F^2 -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
```

Equation Entries

The following section provides an alphabetical listing of the commands associated with the “Equation” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

arch	Equation Methods
------	------------------

Estimate generalized autoregressive conditional heteroskedasticity (GARCH) models.

Syntax

```
eq_name.arch(p,q,options) y [x1 x2 x3] [@ p1 p2 [@ t1 t2]]
```

```
eq_name.arch(p,q,options) y = expression [@ p1 p2 [@ t1 t2]]
```

The ARCH 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, list them after the mean equation using an “@”-sign to separate the mean from the variance equation.

When estimating component ARCH models, you may specify exogenous variance regressors for the permanent and transitory components. 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 regressor.

Options

egarch	Exponential GARCH.
parch[= <i>arg</i>]	Power ARCH. If the optional <i>arg</i> is provided, the power parameter will be set to that value, otherwise the power parameter will be estimated.
cgarch	Component (permanent and transitory) ARCH.

<i>asy = integer</i> (<i>default = 1</i>)	Number of asymmetric terms in the Power ARCH or EGARCH model. The maximum number of terms allowed is 9.
<i>thrsh = integer</i> (<i>default = 0</i>)	Number of threshold terms for GARCH and Component models. The maximum number of terms allowed is 9. For Component models, “thrsh” must take a value of 0 or 1.
<i>archm = arg</i>	ARCH-M (ARCH in mean) specification with the conditional standard deviation (“archm = sd”), the conditional variance (“archm = var”), or the log of the conditional variance (“archm = log”) entered as a regressor in the mean equation.
<i>tdist [= number]</i>	Estimate the model assuming that the residuals follow a conditional Student’s <i>t</i> -distribution (the default is the conditional normal distribution). Providing the optional number greater than two will fix the degrees of freedom to that value. If the argument is not provided, the degrees of freedom will be estimated.
<i>ged [= number]</i>	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.
<i>h</i>	Bollerslev-Wooldridge robust quasi-maximum likelihood (QML) covariance/standard errors. Not available when using the “tdist” or “ged” options.
<i>z</i>	Turn of backcasting for both initial MA innovations and initial variances.
<i>b</i>	Use Berndt-Hall-Hall-Hausman (BHHH) as maximization algorithm. The default is Marquardt.
<i>m = integer</i>	Set maximum number of iterations.
<i>c = scalar</i>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
<i>s</i>	Use the current coefficient values in “C” as starting values (see also param (p. 761)).
<i>s = number</i>	Specify a number between zero and one to determine starting values as a fraction of preliminary LS estimates (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.
deriv = <i>keyword</i>	Set derivative method. The argument <i>keyword</i> should be a one-letter string (“f” or “a” corresponding to fast or accurate numeric derivatives, respectively).
coef = <i>arg</i>	Specify the name of the coefficient vector of a system’s variance component; the default behavior is to use the “C” coefficient vector.
backcast = <i>n</i>	Backcast weight to calculate value used as the presample conditional variance. Weight needs to be greater than 0 and less than or equal to 1; the default value is 0.7. Note that a weight of 1 is equivalent to no backcasting, i.e. using the unconditional residual variance as the presample conditional variance.
p	Print estimation results.

GARCH options

vt	Variance target of the constant term. (Can’t be used with integrated specifications).
integrated	Restrict GARCH model to be integrated, <i>i.e.</i> IGARCH. (Can’t be used with variance targeting).

Saved results

Most of the results saved for the `ls` command are also available after ARCH estimation; see [Equation::ls \(p. 62\)](#) for details.

Examples

```
equation arcl.arch(4, 0, m=1000, h) sp500 c
```

estimates an ARCH(4) model with a mean equation consisting of the series SP500 regressed on a constant. The procedure will perform up to 1000 iterations, and will report Bollerslev-Wooldridge robust QML standard errors upon completion.

The commands:

```
c = 0.1
equation arcl.arch(thrsh=1, s, mean=var) @pch(nys) c ar(1)
```

estimate a TARCH(1, 1)-in-mean specification with the mean equation relating the percent change of NYS to a constant, an AR term of 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:

```
equation arch1.arch(1, 2, asy=0, parch=1.5, ged=1.2)
      dlog(ibm)=c(1)+c(2)* 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 consists of 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 may save the estimated conditional variance as a series named GARCH1.

```
arch1.makegarch garch1
```

Cross-references

See [Chapter 29. “ARCH and GARCH Estimation,” on page 185](#) of the *User’s Guide II* for a discussion of ARCH models. See also [Equation::garch \(p. 53\)](#) and [Equation::makegarch \(p. 66\)](#).

archtest	Equation Views
----------	--------------------------------

Test for autoregressive conditional heteroskedasticity (ARCH).

Carries out Lagrange Multiplier (LM) tests for ARCH in the residuals of a single least squares equation.

Syntax

```
eq_name.archtest(options)
```

Options

You must specify the order of ARCH for which you wish to test. The number of lags to be included in the test equation should be provided in parentheses after the `arch` keyword.

Other Options:

p	Print output from the test.
---	-----------------------------

Examples

```
equation eq1.ls output c labor capital
eq1.archtest(4)
```

Regresses OUTPUT on a constant, LABOR, and CAPITAL, and tests for ARCH up to order 4.

```
equation eq1.arch sp500 c
eq1.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 when performing an `archtest` after an `arch` estimation, EViews uses the standardized residuals (the residual of the mean equation divided by the estimated conditional standard deviation) to form the test.

Cross-references

See “ARCH LM Test” on page 158 of the *User’s Guide II* for further discussion of testing ARCH and Chapter 29. “ARCH and GARCH Estimation,” on page 185 of the *User’s Guide II* for a general discussion of working with ARCH models in EViews.

See also `Equation::hetttest` (p. 59) for a more full-featured version of this test.

arma	Equation Views
------	--------------------------------

Examine ARMA structure of estimated equation.

Provides diagnostic graphical and tabular views that aid you in assessing the structure of the ARMA component of an estimated equation. The view is currently available only for equations specified by list and estimated by least squares that include at least one AR or MA term. There are three views types available: roots, correlogram, and impulse response.

Syntax

```
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.

Options

<code>type = arg</code>	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.
<code>t</code>	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.
<code>hrz = arg</code>	Specifies the maximum lag length for “type = acf”, and the maximum horizon (periods) for “type = imp”.

<code>imp = arg</code>	Specifies the size of the impulse for the impulse response (“type = imp”) view. By default, EViews will use the regression estimated standard error.
<code>save = arg</code>	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.
<code>p</code>	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 partial autocorrelations) for the observations in the sample and the estimated model. The results are computed for a 25 period horizon. 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 83](#) of the *User’s Guide II* for details.

auto	Equation Views
-------------	--------------------------------

Compute serial correlation LM (Lagrange multiplier) test.

Carries out Breusch-Godfrey Lagrange Multiplier (LM) tests for serial correlation in the estimation residuals.

Syntax

```
eq_name.auto(order, options)
```

You must specify the order of serial correlation for which you wish to test. You should specify the number of lags in parentheses after the `auto` keyword, followed by any additional options.

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 you may use the commands:

```
equation eq1.ls output c labor capital
eq1.auto(4)
```

The commands:

```
output(t) c:\result\artest.txt
equation eq1.ls cons c y y(-1)
eq1.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

See [“Serial Correlation LM Test” on page 65](#) of the *User’s Guide II* for further discussion of the Breusch-Godfrey test.

binary	Equation Methods
--------	----------------------------------

Estimate binary dependent variable models.

Estimates models where the binary dependent variable Y is either zero or one (probit, logit, gompit).

Syntax

```
eq_name.binary(options) y x1 [x2 x3 ...]
eq_name.binary(options) specification
```

Options

d = arg (default = “n”)	Specify likelihood: normal likelihood function, probit (“n”), logistic likelihood function, logit (“l”), Type I extreme value likelihood function, Gompit (“x”).
q (default)	Use quadratic hill climbing as the maximization algorithm.
r	Use Newton-Raphson as the maximization algorithm.

b	Use Berndt-Hall-Hall-Hausman (BHHH) for maximization algorithm.
h	Quasi-maximum likelihood (QML) standard errors.
g	GLM standard errors.
m = <i>integer</i>	Set maximum number of iterations.
c = <i>scalar</i>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
s	Use the current coefficient values in C as starting values.
s = <i>number</i>	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 using a constant, WAGE, EDU, and KIDS, and computing QML standard errors, you may use the command:

```
equation eq1.binary(d=1,h) y c wage edu kids
```

Note that this estimation uses the default global optimization options. The commands:

```
param c(1) .1 c(2) .1 c(3) .1  
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:

```
coef beta_probit = probit1.@coefs  
matrix cov_probit = probit1.@coefcov
```

store the estimated coefficients and coefficient covariances in the coefficient vector BETA_PROBIT and matrix COV_PROBIT.

Cross-references

See [“Binary Dependent Variable Models” on page 209](#) of the *User's Guide II* for additional discussion.

cellipse	Equation Views
----------	--------------------------------

Confidence ellipses for coefficient restrictions.

The `cellipse` view displays confidence ellipses for pairs of coefficient restrictions for an equation object.

Syntax

`eq_name.cellipse(options) restrictions`

Enter the equation 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

<code>ind = arg</code>	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.
<code>size = number (default = 0.95)</code>	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.
<code>dist = arg</code>	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 χ^2 distribution.
<code>p</code>	Print the graph.

Examples

The two commands:

```
eq1.cellipse c(1), c(2), c(3)
eq1.cellipse 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).

```
eq1.cellipse(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 [“Confidence Ellipses” on page 142](#) of the *User’s Guide II* for discussion.

See also [Equation::wald \(p. 93\)](#).

censored	Equation Methods
----------	----------------------------------

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

```
eq_name.censored(options) y x1 [x2 x3]
eq_name.censored(options) specification
```

Options

<code>l = number</code> (default = 0)	Set value for the left censoring limit.
<code>r = number</code> (default = none)	Set value for the right censoring limit.
<code>l = series_name, i</code>	Set series name of the indicator variable for the left censoring limit.
<code>r = series_name, i</code>	Set series name of the indicator variable for the right censoring limit.
<code>t</code>	Estimate truncated model.
<code>d = arg</code> (default = “n”)	Specify error distribution: normal (“n”), logistic (“l”), Type I extreme value (“x”).
<code>q (default)</code>	Use quadratic hill climbing as the maximization algorithm.
<code>r</code>	Use Newton-Raphson as the maximization algorithm.
<code>b</code>	Use Berndt-Hall-Hall-Hausman for maximization algorithm.
<code>h</code>	Quasi-maximum likelihood (QML) standard errors.
<code>g</code>	GLM standard errors.
<code>m = integer</code>	Set maximum number of iterations.

<code>c = scalar</code>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
<code>s</code>	Use the current coefficient values in C as starting values.
<code>s = number</code>	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”).
<code>showopts / -showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
<code>p</code>	Print results.

Examples

The command:

```
eq1.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, no right censoring, and the quadratic hill climbing algorithm.

Cross-references

See [Chapter 30. “Discrete and Limited Dependent Variable Models,”](#) on page 209 of the *User’s Guide II* for discussion of censored and truncated regression models.

chow	Equation Views
------	--------------------------------

Chow test for stability.

Carries out Chow breakpoint or Chow forecast tests for parameter constancy.

Syntax

```
eq_name.chow(options) obs1 [obs2 obs3 ...] @ x1 x2 x3
```

You must provide the breakpoint observations (using dates or observation numbers) to be tested. To specify more than one breakpoint, separate the breakpoints by a space. For the Chow breakpoint test, if the equation is specified by list and contains no nonlinear terms, you may specify a subset of the regressors to be tested for a breakpoint after an “@” sign.

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:

```
equation eq1.ls m1 c gdp cpi ar(1)
eq1.chow 1970Q1 1980Q1
```

perform a regression of M1 on a constant, GDP, and CPI with first order autoregressive errors, and employ a Chow breakpoint test to determine whether the parameters before the 1970's, during the 1970's, and after the 1970's are "stable".

To regress the log of SPOT on a constant, the log of P_US, and the log of P_UK, and to 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
```

To test whether only the constant term and the coefficient on the log of P_US prior to and after 1970 are "stable" enter the commands:

```
ppp.chow 1970 @ c log(p_us)
```

Cross-references

See [“Chow's Breakpoint Test” on page 165](#) of the *User's Guide II* for further discussion.

See also [Equation::facbreak \(p. 48\)](#), [Equation::ubreak \(p. 92\)](#), and [Equation::rls \(p. 84\)](#).

coefcov	Equation Views
---------	--------------------------------

Coefficient covariance matrix.

Displays the covariances of the coefficient estimates for an estimated equation.

Syntax

```
eq_name.coefcov(options)
```

Options

p	Print the coefficient covariance matrix.
---	--

Examples

The set of commands:

```
equation eq1.ls lwage c edu edu^2 union
eq1.coefcov
```

declares and estimates equation EQ1 and displays the coefficient covariance matrix in a window. To store the coefficient covariance matrix as a sym object, use “@coefcov”:

```
sym eqcov = eq1.@coefcov
```

Cross-references

See also [Coef::coef](#) (p. 16).

correl	Equation Views
--------	--------------------------------

Display autocorrelation and partial correlations.

Displays the correlogram and partial correlation functions of the residuals of the equation, together with the *Q*-statistics and *p*-values associated with each lag.

Syntax

```
eq_name.correl(n, options)
```

You must specify the largest lag *n* to use when computing the autocorrelations.

Options

p	Print the correlograms.
---	-------------------------

Examples

```
eq1.correl(24)
```

Displays the correlograms of the residuals of EQ1 for up to 24 lags.

Cross-references

See “[Autocorrelations \(AC\)](#)” on page 325 and “[Partial Autocorrelations \(PAC\)](#)” on page 326 of the *User’s Guide I* for a discussion of autocorrelation and partial correlation functions, respectively.

See also [Equation::correlsq](#) (p. 44).

correlsq	Equation Views
-----------------	--------------------------------

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

`equation_name.correlsq(n, options)`

You must specify the largest lag n to use when computing the autocorrelations.

Options

p	Print the correlograms.
----------	-------------------------

Examples

`eq1.correlsq(24)`

displays the correlograms of the squared residuals of EQ1 up to 24 lags.

Cross-references

See [“Autocorrelations \(AC\)” on page 325](#) and [“Partial Autocorrelations \(PAC\)” on page 326](#) of the *User’s Guide I* for a discussion of autocorrelation and partial correlation functions, respectively.

See also [Equation::correl \(p. 43\)](#).

count	Equation Methods
--------------	----------------------------------

Estimates models where the dependent variable is a nonnegative integer count.

Syntax

`eq_name.count(options) y x1 [x2 x3...]`
`eq_name.count(options) specification`

Follow the `count` keyword by the name of the dependent variable and a list of regressors.

Options

<code>d = arg</code> (<i>default</i> = "p")	Likelihood specification: Poisson likelihood ("p"), normal quasi-likelihood ("n"), exponential likelihood ("e"), negative binomial likelihood or quasi-likelihood ("b").
<code>v = positive_num</code> (<i>default</i> = 1)	Specify fixed value for QML parameter in normal and negative binomial quasi-likelihoods.
<code>q</code> (<i>default</i>	Use quadratic hill-climbing as the maximization algorithm.
<code>r</code>	Use Newton-Raphson as the maximization algorithm.
<code>b</code>	Use Berndt-Hall-Hall-Hausman as the maximization algorithm.
<code>h</code>	Quasi-maximum likelihood (QML) standard errors.
<code>g</code>	GLM standard errors.
<code>m = integer</code>	Set maximum number of iterations.
<code>c = scalar</code>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
<code>s</code>	Use the current coefficient values in C as starting values.
<code>s = number</code>	Specify a number between zero and one to determine starting values as a fraction of the EViews default values (out of range values are set to "s = 1").

Examples

The command:

```
equation eq1.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.

```
equation eq1.count arrest c job police
eq1.makeresids(g) res_g
```

estimates a Poisson count model of ARREST on a constant, JOB, and POLICE, and stores the generalized residuals in the series RES_G.

```
equation eq1.count(d=p) y c x1
eq1.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.

```
equation eq1.count(d=p, h) y c x1
```

estimates the same model with QML standard errors and covariances.

Cross-references

See [“Count Models” on page 246](#) of the *User’s Guide II* for additional discussion.

derivs	Equation Views
--------	--------------------------------

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 of the view shows this information in a multiple line graph.

Syntax

```
equation_name.derivs(options)
```

Options

g	Display multiple graphs showing the derivatives of the equation specification with respect to the coefficients, evaluated at each observation.
t	Display spreadsheet view of the values of the derivatives with respect to the coefficients evaluated at each observation.
p	Print results.

Note that the “g” and “t” options may not be used at the same time.

Examples

To show a table view of the derivatives:

```
eq1.derivs(t)
```

To display and print the summary view:

```
eq1.derivs(p)
```

Cross-references

See [“Derivative Computation Options” on page 628](#) of the *User’s Guide II* for details on the computation of derivatives.

See also [Equation::makederivs \(p. 65\)](#) for additional routines for examining derivatives, and [Equation::grads \(p. 58\)](#), and [Equation::makegrads \(p. 67\)](#) for corresponding routines for gradients.

displayname	Equation Procs
--------------------	--------------------------------

Display name for equation objects.

Attaches a display name to an equation which may be used to label output in place of the standard equation object name.

Syntax

```
equation_name.displayname display_name
```

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in equation object names.

Examples

```
eq1.displayname Hours Worked
eq1.label
```

The first line attaches a display name “Hours Worked” to the equation EQ1, and the second line displays the label view of EQ1, including its display name.

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels and display names.

See also [Equation::label \(p. 61\)](#).

equation	Equation Declaration
-----------------	--------------------------------------

Declare an equation object.

Syntax

```
equation eq_name
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 estima-

tion method, and the equation specification. Valid estimation methods are [arch](#) (p. 31), [binary](#) (p. 37), [censored](#) (p. 40), [count](#) (p. 44), [gmm](#) (p. 54), [ls](#) (p. 62), [ordered](#) (p. 71), and [tsls](#) (p. 88). Refer to each method for a description of the available options.

Examples

```
equation cobdoug.ls log(y) c log(k) log(l)
```

declares and estimates an equation object named COBDOUG.

```
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.

```
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

[Chapter 24. “Basic Regression,” on page 5](#) of the *User’s Guide II* provides basic information on estimation and equation objects.

facbreak	Equation Views
----------	--------------------------------

Factor breakpoint test for stability.

Carries out a factor breakpoint test for parameter constancy.

Syntax

```
eq_name.facbreak(options) ser1 [ser2 ser3 ...] @ x1 x2 x3
```

You must provide one or more series to be used as the factors with which to split the sample into categories. To specify more than one factor, separate the factors by a space. If the equation is specified by list and contains no nonlinear terms, you may specify a subset of the regressors to be tested for a breakpoint after an “@” sign.

Options

p	Print the result of the test.
---	-------------------------------

Examples

The commands:

```
equation ppp.ls log(spot) c log(p_us) log(p_uk)
ppp.facbreak season
```

perform a regression of the log of SPOT on a constant, the log of P_US, and the log of P_UK, and employ a factor breakpoint test to determine whether the parameters are stable through the different values of SEASON.

To test whether only the constant term and the coefficient on the log of P_US are “stable” enter the commands:

```
ppp.facbreak season @ c log(p_us)
```

Cross-references

See “Factor Breakpoint Test” on page 152 of the *User’s Guide II* for further discussion.

See also [Equation::chow](#) (p. 41) and [Equation::rls](#) (p. 84).

fit	Equation Procs
-----	--------------------------------

Compute static forecasts or 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 [Equation::makemodel](#) (p. 68) to create a model using the ordered equation results (see example below).

Syntax

```
eq_name.fit(options) yhat [y_se]
eq_name.fit(options) yhat [y_se y_var]
```

Following the `fit` keyword, you should type a name for the forecast series and, optionally, a name for the series containing the standard errors. For ARCH specifications, you may use the second form of the command, and optionally include a name for the conditional variance series.

Forecast standard errors are currently not available for binary, censored, and count models.

Options

d	In models with implicit dependent variables, forecast the entire expression rather than the normalized variable.
u	Substitute expressions for all auto-updating series in the equation.
g	Graph the fitted values together with the ± 2 standard error bands.
e	Produce the forecast evaluation table.

i	Compute the fitted values of the index. Only for binary, censored and count models.
s	Ignore ARMA terms and use only the structural part of the equation to compute the fitted values.
f = <i>arg</i> (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).
p	Print view.

Examples

```
equation eq1.ls cons c cons(-1) inc inc(-1)
eq1.fit c_hat c_se
genr c_up=c_hat+2*c_se
genr c_low=c_hat-2*c_se
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 C_HAT and C_SE. The third and fourth lines compute the ± 2 standard error bounds. The fifth line plots the actual series together with the error bounds.

```
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 specification for Y with a conditional mean that depends 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.makemodel(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 [Equation::forecast](#) (p. 52). See [Equation::makemodel](#) (p. 68) and [Model::solve](#) (p. 287) for forecasting from systems of equations or ordered equations.

See [Chapter 27. “Forecasting from an Equation,”](#) on page 113 of the *User’s Guide II* for a discussion of forecasting in EViews and [Chapter 30. “Discrete and Limited Dependent Variable Models,”](#) on page 209 of the *User’s Guide II* for forecasting from binary, censored, truncated, and count models.

fixedtest	Equation Views
-----------	--------------------------------

Test joint significance of the fixed effects estimates.

Tests the hypothesis that the estimated fixed effects are jointly significant using F and LR test statistics. If the estimated specification involves two-way fixed effects, three separate tests will be performed; one for each set of effects, and one for the joint effects.

Syntax

```
eq_name.fixedtest(options)
```

Options

p	Print output from the test.
---	-----------------------------

Examples

```
equation eq1.ls(cx=f) sales c adver lsales
eq1.fixedtest
```

estimates a specification with cross-section fixed effects and tests whether the fixed effects are jointly significant.

Cross-references

See also [Equation::testadd](#) (p. 86), [Equation::testdrop](#) (p. 87), [Equation::ranhaus](#) (p. 81), and [Equation::wald](#) (p. 93).

forecast	Equation Procs
----------	--------------------------------

Computes (*n*-period ahead) dynamic forecasts of an estimated equation.

`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

```
eq_name.forecast(options) yhat [y_se]
eq_name.forecast(options) yhat [y_se y_var]
```

Enter a name for the forecast series and, optionally, a name for the series containing the standard errors. For ARCH specifications, you may use the second form of the command, and optionally enter a name for the conditional variance series. Forecast standard errors are currently not available for binary or censored models. `forecast` is not available for models estimated using ordered methods.

Options

d	In models with implicit dependent variables, forecast the entire expression rather than the normalized variable.
u	Substitute expressions for all auto-updating series in the equation.
g	Graph the forecasts together with the ± 2 standard error bands.
e	Produce the forecast evaluation table.
i	Compute the forecasts of the index. Only for binary, censored and count models.
s	Ignore ARMA terms and use only the structural part of the equation to compute the forecasts.
b = <i>arg</i>	MA backcast method: “fa” (forecast available). Only for equations estimated with MA terms. This option is ignored if you specify the “s” (structural forecast) option. The default method uses the estimation sample.
f = <i>arg</i> (default = “actual”)	Out-of-forecast-sample fill behavior: “actual” (fill observations outside the forecast sample with actual values for the fitted variable), “na” (fill observations outside the forecast sample with missing values).
p	Print view.

Examples

The following lines:

```
smp1 1970q1 1990q4
equation eq1.ls con c con(-1) inc
smp1 1991q1 1995q4
eq1.fit con_s
eq1.forecast con_d
plot con_s con_d
```

estimate a linear regression over the period 1970Q1–1990Q4, compute static (fitted) and dynamic forecasts for the period 1991Q1–1995Q4, and plot the two forecasts in a single graph.

```
equation eq1.ls m1 gdp ar(1) ma(1)
eq1.forecast m1_bj bj_se
eq1.forecast(s) m1_s s_se
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.

Cross-references

To perform static forecasting with equation objects see [Equation::fit](#) (p. 49). For multiple equation forecasting, see [Equation::makemodel](#) (p. 68), and [Model::solve](#) (p. 287).

For more information on equation forecasting in EViews, see [Chapter 27. “Forecasting from an Equation,”](#) on page 113 of the *User’s Guide II*.

garch	Equation Views
-------	--------------------------------

Conditional variance/covariance of (G)ARCH estimation.

Displays the conditional variance, covariance or correlation of an equation estimated by ARCH.

Syntax

```
eq_name.garch(options)
```

Options

v	Display conditional variance graph instead of the standard deviation graph.
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.

```
eq1.garch(v, p)
```

displays and prints the estimated conditional variance graph.

Cross-references

ARCH estimation is described in [Chapter 29. “ARCH and GARCH Estimation,”](#) on page 185 of the *User’s Guide II*.

gmm	Equation Methods
------------	----------------------------------

Estimation by generalized method of moments (GMM).

The equation object must be specified with a list of instruments.

Syntax

```
eq_name.gmm(options) y x1 [x2 x3 ...] @ z1 [z2 z3 ...]
eq_name.gmm(options) specification @ z1 [z2 z3 ...]
```

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.

In panel settings, you may specify dynamic instruments corresponding to predetermined variables. 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)”.

Note that dynamic instrument specifications may easily generate excessively large numbers of instruments.

Options

General Options

<code>m = integer</code>	Maximum number of iterations.
<code>c = number</code>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
<code>l = number</code>	Set maximum number of iterations on the first-stage iteration to get the one-step weighting matrix.
<code>showopts / -showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
<code>deriv = keyword</code>	Set derivative methods. The argument <i>keyword</i> 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.
<code>p</code>	Print results.

Additional Options for Non-Panel Equation estimation

<code>w</code>	Use White’s diagonal weighting matrix (for cross section data).
<code>b = arg (default = “nw”)</code>	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).
<code>q</code>	Use the quadratic kernel. Default is to use the Bartlett kernel.
<code>n</code>	Prewhiten by a first order VAR before estimation.
<code>i</code>	Iterate simultaneously over the weighting matrix and the coefficient vector.
<code>s (default)</code>	Iterate sequentially over the weighting matrix and coefficient vector.

o	Iterate only on the coefficient vector with one step of the weighting matrix.
c	One step (iteration) of the coefficient vector following one step of the weighting matrix.
e	TOLS estimates with GMM standard errors.

Additional Options for Panel Equation estimation

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 (“per = 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 weights (“wgt = cxdiag”), period diagonal weights (“wgt = perdiag”).
gmm = arg	<p>GMM weighting: 2SLS (“gmm = 2sls”), White period system covariances (Arellano-Bond 2-step/<i>n</i>-step) (“gmm = perwhite”), White cross-section system (“gmm = cxwhite”), White diagonal (“gmm = stackedwhite”), Period system (“gmm = persur”), Cross-section system (“gmm = cxsur”), Period heteroskedastic (“cov = perdiag”), Cross-section heteroskedastic (“gmm = cxdiag”).</p> <p>By default, uses the identity matrix unless estimated with first difference transformation (“cx = fd”), in which case, uses (Arellano-Bond 1-step) difference weighting matrix. In this latter case, you should specify 2SLS weights (“gmm = 2sls”) for Anderson-Hsiao estimation.</p>
cov = 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”).

<code>keepwgts</code>	Keep full set of GLS/GMM weights used in estimation with object, if applicable (by default, only weights which take up little memory are saved).
<code>nodf</code>	Do not perform degree of freedom corrections in computing coefficient covariance matrix. The default is to use degree of freedom corrections.
<code>coef = arg</code>	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the “C” coefficient vector.
<code>iter = arg</code> (<i>default</i> = “onec”)	Iteration control for GLS and GMM weighting 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 some options are only available for a subset of specifications.

Examples

In a non-panel workfile, we may estimate equations using the standard GMM options. The specification:

```
eq.gmm(w) y c f k @ c z1 z2 z3
```

estimates the linear specification using a White diagonal weighting matrix (one-step, with no iteration). The command:

```
eq.gmm(e, b=v) c(1) + c(2)*(m^c(1) + k^(1-c(1))) @ c z1 z2 z3
```

estimates the nonlinear model using two-stage least squares (instrumental variables) with GMM standard errors computed using Newey-West automatic bandwidth selected weights.

When working with a workfile that has a panel structure, you may use the panel equation estimation options. The command

```
eq.gmm(cx=fd, per=f) dj dj(-1) @ @dyn(dj)
```

estimates an Arellano-Bond “1-step” estimator with differencing of the dependent variable DJ, period fixed effects, and dynamic instruments constructed using DJ with observation specific lags from period $t - 2$ to 1.

To perform the “2-step” version of this estimator, you may use:

```
eq.gmm(cx=fd, per=f, gmm=perwhite, iter=oneb) dj dj(-1) @ @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, to use the estimates to form period covariance GMM weights, and then re-estimate the model.

You may iterate the GMM weights to convergence using:

```
eq.gmm(cx=fd, per=f, gmm=perwhite, iter=seq) dj dj (-1) @ @dyn(dj)
```

Alternately:

```
eq.gmm(cx=od, gmm=perwhite, iter=oneb) dj dj (-1) x y @ @dyn(dj,-2,-6) x (-1) y (-1)
```

estimates an Arellano-Bond “2-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

See [Chapter 25. “Additional Regression Methods,” on page 23](#) and [Chapter 39. “Panel Estimation,” on page 541](#) of the *User’s Guide II* for discussion of the various GMM estimation techniques.

grads	Equation Views
-------	--------------------------------

Gradients of the objective function.

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. Evaluating the gradients at current coefficient values allows you to examine 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

```
equation_name.grads(options)
```

Options

g	Display multiple graph showing the gradients of the objective function with respect to the coefficients evaluated at each observation.
t (default)	Display spreadsheet view of the values of the gradients of the objective function with respect to the coefficients evaluated at each observation.
p	Print results.

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 [Equation::derivs](#) (p. 46), [Equation::makederivs](#) (p. 65), and [Equation::makegrads](#) (p. 67).

hettest	Equation Views
---------	--------------------------------

Test for Heteroskedasticity.

Performs a test for heteroskedasticity among the residuals from an equation.

The test performed can be a Breusch-Pagan-Godfrey (the default option), Harvey, Glejser, ARCH or White style test.

Syntax

```
equation_name.hettest(options) variables
```

Options

<code>type = <i>keyword</i></code>	where <i>keyword</i> is either “BPG” (Breusch-Pagan-Godfrey - default), “Harvey”, “Glejser”, “ARCH”, or “White”.
<code>c</code>	include cross terms for White test.
<code>lags = <i>int</i></code>	set number of lags to use for ARCH test. (Only applies when type = “ARCH”.

Variables

A list of series names to be included in the auxiliary regression. Not applicable for ARCH or White type tests. The following keywords may be included:

<code>@regs</code>	include every regressor from the original equation.
<code>@grads</code>	include every gradient in the original equation (non-linear equations only).
<code>@grad(<i>int</i>)</code>	include the <i>int</i> -th gradient.

<code>@white(key)</code>	include white-style regressors (the cross-product of the regressor list, or the gradient list if non-linear). <i>key</i> may be on of the following keywords: “@regs” (include every regressor from the original equation), “@drop(<i>variables</i>)” (drop a variable from those already included. For example, “@white(@regs @drop(x2))” would include all original regressors apart from X2), “@comp” (include the compatible style White regressors, <i>i.e.</i> levels, squares, and cross-products).
<code>@arch(lag_structure)</code>	include an ARCH specification with the number of lags specified by <i>lag_structure</i> . If <i>lag_structure</i> is a single number, then it defines the number of lags to include. Otherwise, the lag structure is in pairs. For example, “@arch(1 5 9 10)” will include lags 1, 2, 3, 4, 5, 9, 10.
<code>@uw(variables)</code>	include unweighted variables (only applicable in a weighted original equation).

Examples

```
eql.hetest(type=harvey) @white(@regs @drop(log(ip)))
```

performs a heteroskedasticity test with an auxiliary regression of the log of squared residuals on the cross product of all the original equation’s variables, except LOG(IP).

Cross-references

See “[Heteroskedasticity Tests,](#)” [beginning on page 156](#) of the *User’s Guide II* for a discussion of heteroskedasticity testing in EViews.

hist	Equation Views
------	--------------------------------

Histogram and descriptive statistics of the residual series of an equation.

Syntax

```
equation_name.hist(options)
```

Options

p	Print the histogram.
---	----------------------

Examples

```
eql.hist
```

Displays the histogram and descriptive statistics of the residual series of equation EQ1.

Cross-references

See “[Histogram and Stats](#)” on page 306 of the *User’s Guide I* for a discussion of the descriptive statistics reported in the histogram view.

label	Equation Views Equation Procs
-------	---

Display or change the label view of an equation, including the last modified date and display name (if any).

As a procedure, `label` changes the fields in the equation label.

Syntax

```
equation_name.label
equation_name.label(options) [text]
```

Options

The first version of the command displays the label view of the equation. The second version may be used to modify the label. 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 <i>text</i> .
s	Sets the source field to <i>text</i> .
u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of EQ1 with “Data from CPS 1988 March File”:

```
eq1.label(r)
eq1.label(r) Data from CPS 1988 March File
```

To append additional remarks to EQ1, and then to print the label view:

```
eq1.label(r) Log of hourly wage
eq1.label(p)
```

To clear and then set the units field, use:

```
eq1.label(u) Millions of bushels
```

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels.

See also [Equation::displayname \(p. 47\)](#).

logit	Equation Methods
-------	----------------------------------

Estimate binary models with logistic errors.

Provide for backward compatibility. Equivalent to issuing the command, `binary` with the option “(d = l)”.

See [binary \(p. 37\)](#).

ls	Equation Methods
----	----------------------------------

Estimation by linear or nonlinear least squares regression.

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

```
eq_name.ls(options) y x1 [x2 x3 ...]
eq_name.ls(options) specification
paneq_name.ls(options) y x1 [x2 x3 ...] [@cxreg z1 z2 ...] [@perreg z3 z4 ...]
paneq_name.ls(options) specification
```

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 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 may also be specified by explicit equation. You should specify the equation as a formula. The parameters to be estimated should be included explicitly: “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.

Options

General options

<code>m = integer</code>	Set maximum number of iterations.
<code>c = scalar</code>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
<code>s</code>	Use the current coefficient values in “C” as starting values for equations with AR or MA terms (see also param (p. 761)).
<code>s = number</code>	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 to be used. Note that out of range values are set to “s = 1”. Specifying “s = 0” initializes coefficients to zero. By default EViews uses “s = 1”.
<code>showopts / -showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
<code>deriv = keyword</code>	Set derivative methods. The argument <i>keyword</i> 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.
<code>w = series_name</code>	Weighted Least Squares. Each observation will be weighted by multiplying by the specified series.
<code>h</code>	White’s heteroskedasticity consistent standard errors.
<code>n</code>	Newey-West heteroskedasticity and autocorrelation consistent (HAC) standard errors.
<code>z</code>	Turn off backcasting in ARMA models.
<code>noconst</code>	Do not include a constant in exogenous regressors list for VARs.
<code>p</code>	Print basic estimation results.

Note: not all options are available for all equation methods. See the User’s Guide II for details on each estimation method.

Additional Options for Panel Equation estimation

<code>cx = arg</code>	Cross-section effects: (default) none, fixed effects (“cx = f”), random effects (“cx = r”).
<code>per = arg</code>	Period effects: (default) none, fixed effects (“per = f”), random effects (“per = r”).
<code>wgt = arg</code>	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”).
<code>cov = arg</code>	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/PCSE (“cov = perdiag”).
<code>keepwghts</code>	Keep full set of GLS weights used in estimation with object, if applicable (by default, only small memory weights are saved).
<code>rancalc = arg</code> (<i>default</i> = “sa”)	Random component method: Swamy-Arora (“rancalc = sa”), Wansbeek-Kapteyn (“rancalc = wk”), Wallace-Hussain (“rancalc = wh”).
<code>nodf</code>	Do not perform degree of freedom corrections in computing coefficient covariance matrix. The default is to use degree of freedom corrections.
<code>b</code>	Estimate using a balanced sample (pool estimation only).
<code>coef = arg</code>	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the “C” coefficient vector.
<code>iter = arg</code> (<i>default</i> = “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.

Examples

```
equation eq1.ls m1 c uemp inf(0 to -4) @trend(1960:1)
```

estimates a linear regression of M1 on a constant, UEMP, INF (from current up to four lags), and a linear trend.

```
equation eq2.ls(z) d(tbill) c inf @seas(1) @seas(1)*inf ma(2)
```

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. The “z” option turns off backcasting.

```
coef(2) beta
param beta(1) .2 beta(2) .5 c(1) 0.1
equation eq3.ls(h) q = beta(1)+beta(2)*(1^c(1) + k^(1-c(1)))
```

estimates the nonlinear regression starting from the specified initial values. The “h” option reports heteroskedasticity consistent standard errors.

```
equation eq4.ls r = c(1)+c(2)*r(-1)+div(-1)^c(3)
sym betacov = eq4.@cov
```

declares and estimates a nonlinear equation and stores the coefficient covariance matrix in a symmetric matrix called BETACOV.

Cross-references

[Chapter 24. “Basic Regression,” on page 5](#) and [Chapter 25. “Additional Regression Methods,” on page 23](#) of the *User’s Guide II* discuss the various regression methods in greater depth.

See [Chapter 39. “Panel Estimation,” on page 541](#) of the *User’s Guide II* for a discussion of panel equation estimation.

See also [Chapter 5. “Special Expression Reference,” on page 815](#), for special terms that may be used in `ls` specifications.

makederivs	Equation Procs
------------	--------------------------------

Make a group containing individual series which hold the derivatives of the equation specification.

Syntax

```
equation_name.makederivs(options) [ser1 ser2 ...]
```

If desired, enclose the name of a new group object to hold the series 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.

names must match the number of target series.

Options

n = <i>arg</i>	Name of group object to contain the series.
----------------	---

Examples

```
eq1.makederivs(n=out)
```

creates a group named OUT containing series named DERIV01, DERIV02, and DERIV03.

```
eq1.makederivs(n=out) d1 d2 d3
```

creates the same group, but names the series D1, D2 and D3.

Cross-references

See [Chapter 35. “State Space Models and the Kalman Filter,” on page 383](#) of the *User’s Guide II* for details on state space estimation.

See also [Equation::derivs \(p. 46\)](#), [Equation::grads \(p. 58\)](#), [Equation::makegrads \(p. 67\)](#).

makegarch	Equation Procs
-----------	--------------------------------

Generate conditional variance series.

Saves the estimated conditional variance (from an equation estimated using ARCH) as a named series.

Syntax

```
eq_name.makegarch series1_name [@ series2_name]
```

You should provide a name for the saved conditional standard deviation series following the `makegarch` keyword. If you do not provide a name, EViews will name the series using the next available name of the form “GARCH##” (if GARCH01 already exists, it will be named GARCH02, and so on).

For component GARCH equations, the permanent component portion of the conditional variance may be saved by adding “@” followed by a series name.

Examples

```
equation eql.arch sp c
eql.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 merely wish to view a plot of the conditional standard deviation without saving the series, use the [Equation::garch](#) (p. 53) view.

The commands

```
equation eql.arch(cgarch) sp c
eql.makegarch cvar @ pvar
```

first estimates a Component GARCH model and then saves both the conditional variance and the permanent component portion of the conditional variance in the series CVAR and PVAR, respectively.

Cross-references

See [Chapter 29. “ARCH and GARCH Estimation,”](#) on page 185 of the *User’s Guide II* for a discussion of GARCH models.

See also [Equation::arch](#) (p. 31), [Equation::archtest](#) (p. 34), and [Equation::garch](#) (p. 53).

makegrads	Equation Procs
-----------	--------------------------------

Make a group containing individual series which hold the gradients of the objective function.

Syntax

```
equation_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

<code>n = arg</code>	Name of group object to contain the series.
----------------------	---

Examples

```
eql.grads(n=out)
```

creates a group named OUT containing series named GRAD01, GRAD02, and GRAD03.

```
eq1.makegrads(n=out) g1 g2 g3
```

creates the same group, but names the series G1, G2 and G3.

Cross-references

See also [Equation::derivs \(p. 46\)](#), [Equation::makederivs \(p. 65\)](#), [Equation::grads \(p. 58\)](#).

makelimits	Equation Procs
-------------------	--------------------------------

Create vector of limit points from ordered models.

`makelimits` creates a vector of the estimated limit points from equations estimated by [Equation::ordered \(p. 71\)](#).

Syntax

```
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 eq1.ordered edu c age race gender
eq1.makelimits cutoff
```

Estimates an ordered probit and saves the estimated limit points in a vector named CUTOFF.

Cross-references

See [“Ordered Dependent Variable Models” on page 226](#) of the *User’s Guide II* for a discussion of ordered models.

makemodel	Equation Procs
------------------	--------------------------------

Make a model from an equation object.

Syntax

```
equation_name.makemodel(name) assign_statement
```

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

```
equation eq3.ls 1 4 m1 gdp tb3
eq3.makemodel(eqmod) @prefix s_
```

estimates an equation and makes a model named EQMOD from the estimated equation object. EQMOD includes an assignment statement “ASSIGN @PREFIX S_”. Use the command “show eqmod” or “eqmod.spec” to open the EQMOD window.

Cross-references

See [Chapter 36. “Models,” on page 407](#) of the *User’s Guide II* for a discussion of specifying and solving models in EViews. See also [solve \(p. 782\)](#).

makeregs	Equation Procs
-----------------	--------------------------------

Make regressor group.

Creates a group containing the dependent and independent variables from an equation specification.

Syntax

```
equation_name.makeregs grp_name
```

Follow the keyword `makeregs` with the name of the group.

Examples

```
equation eq1.ls y c x1 x2 x3 z
eq1.makeregs reggroup
```

creates a group REGGROUP containing the series Y X1 X2 X3 and Z.

Cross-references

See also [Group::group \(p. 202\)](#) and [Equation::equation \(p. 47\)](#).

makeresids	Equation Procs
-------------------	--------------------------------

Create residual series.

Creates and saves residuals in the workfile from an estimated equation object.

Syntax

```
equation_name.makeresids(options) [res1]
```

Follow the equation name with a period and the `makeresids` keyword, then provide a name to be given to the stored residual.

Options

o (<i>default</i>)	Ordinary residuals.
s	Standardized residuals (available only after weighted estimation and GARCH, binary, ordered, censored, and count models).
g (<i>default</i> for ordered models)	Generalized residuals (available only for binary, ordered, censored, and count models).

Examples

```
equation eq1.ls y c m1 inf unemp
eq1.makesresids res_eq1
```

estimates a linear regression of Y on a constant, M1, INF, UNEMP, and saves the residuals as a series named RES_EQ1.

Cross-references

See [“Weighted Least Squares” on page 32](#) of the *User’s Guide II* for a discussion of standardized residuals after weighted least squares and [Chapter 30. “Discrete and Limited Dependent Variable Models,” on page 209](#) of the *User’s Guide II* for a discussion of standardized and generalized residuals in binary, ordered, censored, and count models.

means	Equation Views
-------	--------------------------------

Descriptive statistics by category of dependent variable.

Computes and displays descriptive statistics of the explanatory variables (regressors) of an equation categorized by values of the dependent variable.

Syntax

```
eq_name.means(options)
```

Options

p	Print the descriptive statistics table.
---	---

Examples

```
equation eq1.binary(d=1) work c edu faminc
eq1.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 30. “Discrete and Limited Dependent Variable Models,”](#) on page 209 of the *User’s Guide II* for a discussion of binary dependent variable models.

ordered	Equation Methods
---------	----------------------------------

Estimation of ordered dependent variable models.

Syntax

equation name.**ordered**(*options*) *y* *x1* [*x2* *x3* ...]

equation name.**ordered**(*options*) *specification*

The `ordered` command estimates the model and saves the results as an equation object with the given name.

Options

<code>d = arg</code> (<i>default</i> = “n”)	Specify likelihood: normal likelihood function, ordered probit (“n”), logistic likelihood function, ordered logit (“l”), Type I extreme value likelihood function, ordered Gompit (“x”).
<code>q</code> (<i>default</i>)	Use quadratic hill climbing as the maximization algorithm.
<code>r</code>	Use Newton-Raphson as the maximization algorithm.
<code>b</code>	Use Berndt-Hall-Hall-Hausman as maximization algorithm.
<code>h</code>	Quasi-maximum likelihood (QML) standard errors.
<code>g</code>	GLM standard errors.
<code>m = integer</code>	Set maximum number of iterations.
<code>c = scalar</code>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
<code>s</code>	Use the current coefficient values in C as starting values.
<code>s = number</code>	Specify a number between zero and one to determine starting values as a fraction of preliminary EViews default values (out of range values are set to “s = 1”).

<code>showopts / -showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
<code>deriv = keyword</code>	Set derivative methods. The argument <i>keyword</i> 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.
<code>p</code>	Print results.

If you choose to employ user specified starting values, the parameters corresponding to the limit points must be in ascending order.

Examples

```
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.

```
param c(1) .1 c(2) .2 c(3) .3 c(4) .4 c(5) .5  
equation eq1.binary(s) y c x z  
coef betahat = eq1.@coefs  
eq1.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 [“Ordered Dependent Variable Models” on page 226](#) of the *User’s Guide II* for additional discussion.

See [Equation::binary \(p. 37\)](#) for the estimation of binary dependent variable models. See also [Equation::makelimits \(p. 68\)](#).

output	Equation Views
--------	--------------------------------

Display estimation output.

The `output` command changes the default object view to display the equation output (equivalent to using [Equation::results \(p. 83\)](#)).

Syntax

```
eq_name.output(options)
```

Options

p	Print estimation output for estimation object.
---	--

Examples

```
eq1.output
```

displays the estimation output for equation EQ1.

Cross-references

See [Equation::results](#) (p. 83).

predict	Equation Views
---------	--------------------------------

Prediction table for binary and ordered dependent variable models.

The prediction table displays the actual and estimated frequencies of each distinct value of the discrete dependent variable.

Syntax

```
eq_name.predict(options)
```

For binary models, you may optionally specify how large the estimated probability must be to be considered a success ($y = 1$). Specify the cutoff level as an option in parentheses after the keyword `predict`.

Options

<i>n</i> (default = .5)	Cutoff probability for success prediction in binary models (between 0 and 1).
p	Print the prediction table.

Examples

```
equation eq1.binary(d=1) 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 209 of the *User’s Guide II* for a discussion of binary models, and [“Expectation-Prediction \(Classification\) Table”](#) on page 217 of the *User’s Guide II* for examples of prediction tables.

probit	Equation Methods
--------	----------------------------------

Estimation of binary dependent variable models with normal errors.

Equivalent to “`binary(d=n)`”.

See [binary](#) (p. 37).

qreg	Equation Methods
------	----------------------------------

Estimate a quantile regression specification.

Syntax

`eq_name.qreg(options) y x1 [x2 x3 ...]`
`eq_name.qreg(options) linear_specification`

Options

<code>quant = number</code> (<i>default</i> = 0.5)	Quantile to be fit (where <i>number</i> is a value between 0 and 1).
<code>cov = arg</code> (<i>default</i> = “sandwich”)	Method for computing coefficient covariance matrix: “iid” (ordinary estimates), “sandwich” (Huber sandwich estimates), “boot” (bootstrap estimates). When “cov = iid” or “cov = sandwich”, EViews will use the sparsity nuisance parameter calculation specified in “spmethod = ” when estimating the coefficient covariance matrix.
<code>bwmeth = arg</code> (<i>default</i> = “hs”)	Method for automatically selecting bandwidth value for use in estimation of sparsity and coefficient covariance matrix: “hs” (Hall-Sheather), “bf” (Bofinger), “c” (Chamberlain).
<code>bw = number</code>	Use user-specified bandwidth value in place of automatic method specified in “bwmeth = ”.
<code>bwsz = number</code> (<i>default</i> = 0.05)	Size parameter for use in computation of bandwidth (used when “bw = hs” and “bw = bf”).
<code>spmethod = arg</code> (<i>default</i> = “kernel”)	Sparsity estimation method: “resid” (Siddiqui using residuals), “fitted” (Siddiqui using fitted quantiles at mean values of regressors), “kernel” (Kernel density using residuals) Note: “spmethod = resid” is not available when “cov = sandwich”.

<code>btmethod = arg</code> (<i>default</i> = “pair”)	Bootstrap method: “resid” (residual bootstrap), “pair” (xy-pair bootstrap), “mcmb” (MCMB bootstrap), “mcmba” (MCMB-A bootstrap).
<code>btreps = integer</code> (<i>default</i> = 100)	Number of bootstrap repetitions
<code>btseed = positive integer</code>	Seed the bootstrap random number generator. If not specified, EViews will seed the bootstrap random number generator with a single integer draw from the default global random number generator.
<code>btrnd = arg</code> (<i>default</i> = “kn” or method previously set using <code>rndseed</code> (p. 770))	Type of random number generator for the bootstrap: improved Knuth generator (“kn”), improved Mersenne Twister (“mt”), Knuth’s (1997) lagged Fibonacci generator used in EViews 4 (“kn4”) L’Ecuyer’s (1999) combined multiple recursive generator (“le”), Matsumoto and Nishimura’s (1998) Mersenne Twister used in EViews 4 (“mt4”).
<code>btobs = integer</code>	Number of observations for bootstrap subsampling (when “bsmethod = pair”). Should be significantly greater than the number of regressors and less than or equal to the number of observations used in estimation. EViews will automatically restrict values to the range from the number of regressors and the number of estimation observations. If omitted, the bootstrap will use the number of observations used in estimation.
<code>btout = name</code>	(optional) Matrix to hold results of bootstrap simulations.
<code>k = arg</code> (<i>default</i> = “e”)	Kernel function for sparsity and coefficient covariance matrix estimation (when “spmeth = kernel”): “e” (Epanechnikov), “r” (Triangular), “u” (Uniform), “n” (Normal-Gaussian), “b” (Biweight-Quartic), “t” (Triweight), “c” (Cosinus).
<code>m = integer</code>	Maximum number of iterations.
<code>s</code>	Use the current coefficient values in “C” or explicit coefficients as starting values estimation.
<code>s = number</code> (<i>default</i> = 0)	Determine starting values for equations. Specify a number between 0 and 1 representing the fraction of preliminary least squares coefficient estimates. Note that out of range values are set to the default.
<code>showopts / -showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the estimation output.

p Print estimation results.

Examples

`equation eq1.qreg y c x`

estimates the default least absolute deviations (median) regression for the dependent variable Y on a constant and X. The estimates use the Huber Sandwich method for computing the covariance matrix, with individual sparsity estimates obtained using kernel methods. The bandwidth uses the Hall and Sheather formula.

`equation eq1.qreg(quant=0.6, cov=boot, btmeth=mcmba) y c x`

estimates the quantile regression for the 0.6 quantile using MCMB-A bootstrapping to obtain estimates of the coefficient covariance matrix.

Cross-references

See [Chapter 31. “Quantile Regression,” on page 259](#) of the *User’s Guide II* for a discussion of the quantile regression.

qrprocess	Equation Views
-----------	--------------------------------

Display quantile process coefficient estimates (multiple quantile regression estimates).

Syntax

`eq_name.qrprocess(options) [arg]`

where *arg* is a optional list containing the quantile values and/or vectors containing the quantile values for which you wish to compute estimates.

If *arg* is not specified, EViews will display results for the existing equation and coefficients for equations estimated at a set of equally spaced quantiles specified with the “n = ” option; the default is to display results for the deciles.

You may specify a maximum of 1000 total coefficients (number of coefficients in the equation specification times the number of quantiles).

All estimation will be performed using the settings from the original equation.

Options

<code>n = arg</code> <i>(default = 10)</i>	Number of quantiles for process estimates.
<code>graph</code>	Display process estimate results as graph.
<code>size = arg</code> <i>(default = 0.95)</i>	Confidence interval size for graph display

<code>quantout = name</code>	Save vector containing test quantile values.
<code>coefout = name</code>	Save matrix containing test coefficient estimates. Each column of the matrix corresponds to a different quantile matching the corresponding quantile in “quantout = ”. To match the covariance matrix given in “covout = ” you should take the @vec of the coefficient matrix.
<code>covout = name</code>	Save symmetric matrix containing covariance matrix for the vector set of coefficient estimates.
<code>p</code>	Print output.

Examples

```
equation eq1.qreg log(y) c log(x)
eq1.qrprocess
```

estimates a quantile (median) regression of LOG(Y) on a constant and LOG(X), and displays results for all nine quantiles in a table

Similarly,

```
equation eq1.qreg(quant=.4) log(y) c log(x)
eq1.qrprocess(coefcout=cout)
```

displays the coefficient estimated at the deciles (and at 0.4), and saves the coefficient matrix to COUT.

```
eq1.qrprocess(coefout=cout, n=4, graph)
eq1.qrprocess(coefout=cout, graph) .25 .5 .75
```

both estimate coefficients for the three quartiles and display the results in a graph, as does the equivalent:

```
vector v1(3)
v1.fill .25 .5 .75
eq1.qrprocess(graph) v1
```

Cross-references

See [“Process Coefficients” on page 266](#) of the *User’s Guide II* for a discussion of the quantile process. See also [“qrslope” on page 78](#).

qrslope	Equation Views
---------	--------------------------------

Perform Wald test of equality of slope coefficients across multiple quantile regression estimates. The equality test restrictions are of the form: $\beta_{\tau} = \beta_{\tau'}$ for the slope coefficients β .

Syntax

```
eq_name.qrslope(options) [arg]
```

where *arg* is an optional list containing the quantile values and/or vectors containing the quantile values for which you wish to compare slope coefficients.

Note that the argument *arg* is optional. If *arg* is not specified, EViews will display results for the existing equation and coefficients for equations estimated at a set of equally spaced quantiles; the default is to test the quartiles (.25, .5, .75).

You may specify a maximum of 1000 total coefficients (number of coefficients in the equation specification times the number of quantiles) in the test.

All estimation will be performed using the settings from the original equation.

Options

<code>n = arg</code> (default = 4)	Number of quantiles for process estimates.
<code>quantout = name</code>	Save vector containing test quantile values.
<code>coefout = name</code>	Save matrix containing test coefficient estimates. Each column of the matrix corresponds to a different quantile matching the corresponding quantile in “quantout = ”. To match the covariance matrix given in “covout = ” you should take the @vec of the coefficient matrix.
<code>covout = name</code>	Save symmetric matrix containing covariance matrix for the vector set of coefficient estimates.
<code>p</code>	Print output from the test.

Examples

```
equation eq1.qreg log(y) c log(x)
eq1.qrslope
```

estimates a quantile (median) regression of LOG(Y) on a constant and LOG(X), and tests for the equality of the coefficients of LOG(X) for all three of the quartiles.

Similarly,

```
equation eq1.qreg(quant=.4) log(y) c log(x)
```

```
eq1.qrslope(coefcout=cout)
```

tests for equality of the LOG(X) coefficient estimated at $\{.25, .4, .5, .75\}$, and saves the coefficient matrix to COUT. Both

```
eq1.qrslope(coefout=count, n=10)
```

```
eq1.qrslope(coefout=cout) .1 .2 .3 .4 .5 .6 .7 .8 .9
```

perform the Wald test for equality of the slope coefficient across all of the deciles, as does the equivalent

```
vector v1(9)
```

```
v1.fill .1,.2,.3,.4,.5,.6,.7,.8,.9
```

```
eq1.qrslope v1
```

Cross-references

See [“Slope Equality Test” on page 267](#) of the *User’s Guide II* for a discussion of the slope equality test. See also [“qrsymm” on page 79](#).

qrsymm	Equation Views
--------	--------------------------------

Perform Wald test of coefficients using symmetric quantiles. The symmetric quantile test restrictions are of the form: $\beta_{\tau} + \beta_{1-\tau} = 2\beta_{0.5}$.

Syntax

```
eq_name.qrsymm(options) [arg]
```

where *arg* is an optional list containing the quantile values and/or vectors containing the quantile values for which you wish to form symmetric quantile restrictions.

Note that the argument *arg* is optional. If *arg* is not specified, EViews will perform one of two tests, depending on the original equation specification:

- If the original specification is a median regression ($\tau = 0.5$), EViews will test using estimates obtained at the specified outer quantiles; by default at the outer quantiles $\{0.25, 0.75\}$.
- For specifications estimated for $\tau \neq 0.5$, you may specify 0 as the specified number of quantiles to perform a test using only estimates obtained at the symmetric pair $\{\tau, 1 - \tau\}$.

If *arg* is specified, EViews will test using the specified quantiles and their complements, but will not automatically use the equation estimates. If you wish to use the latter, it must be entered explicitly.

You may specify a maximum of 1000 total coefficients (number of coefficients in the equation specification times the number of quantiles) in the test.

All estimation will be performed using the settings from the original equation.

Options

<code>n = arg</code>	Number of quantiles for testing.
<code>incpt</code>	Test using the intercept only (the default is to test all coefficients).
<code>quantout = name</code>	Save vector containing test quantile values.
<code>coefout = name</code>	Save matrix containing test coefficient estimates. Each column of the matrix corresponds to a different quantile matching the corresponding quantile in “quantout = ”. To match the covariance matrix given in “covout = ” you should take the <code>@vec</code> of the coefficient matrix.
<code>covout = name</code>	Save symmetric matrix containing covariance matrix for the vector set of coefficient estimates.
<code>p</code>	Print output from the test.

Examples

```
equation eql.qreg log(y) c log(x)
eql.qrsymm
```

estimates a quantile (median) regression of LOG(Y) on a constant and LOG(X), and performs a symmetry test using the outer quartiles.

We may restrict the hypothesis to just consider the intercept,

```
eql.qrsymm(incpt)
```

and we may specify alternative quantiles to test

```
eql.qrsymm(quantout=qo) .2 .4 .7
```

Note that the latter command will test using the symmetric quantiles {0.2, 0.3, 0.4, 0.6, 0.7, 0.8}, and at the median. Note that the median is automatically estimated, even though it is not specified explicitly, since it is always required for testing.

Alternatively, the commands

```
equation eql.qreg(quant=.4) log(y) c log(x)
eql.qrsymm(n=0)
```

will perform the test using the symmetric quantiles {0.4, 0.6} and the median.

To performs the test using all of the deciles, you may enter

```
vector v1(10)
v1.fill .1, .2, .3, .4
```

```
eq1.qrsymm v1
```

Cross-references

See [“Symmetric Quantiles Test” on page 269](#) of the *User’s Guide II* for a discussion of the symmetric quantiles test. See also [“qrslope” on page 78](#).

ranhaus	Equation Views
---------	--------------------------------

Test for correlation between random effects and regressors using Hausman test.

Tests the hypothesis that the random effects (components) are correlated with the right-hand side variables in a panel or pool equation setting. Uses Hausman test methodology to compare the results from the estimated random effects specification and a corresponding fixed effects specification. If the estimated specification involves two-way random effects, three separate tests will be performed; one for each set of effects, and one for the joint effects.

Only valid for panel or pool regression equations estimated with random effects. Note that the test results may be suspect in cases where robust standard errors are employed.

Syntax

```
eq_name.ranhaus(options)
```

Options

p	Print output from the test.
---	-----------------------------

Examples

```
equation eq1.ls(cx=r) sales c adver lsales
eq1.ranhaus
```

estimates a specification with cross-section random effects and tests whether the random effects are correlated with the right-hand side variables ADVER and LSALES using the Hausman test methodology.

Cross-references

See also [Equation::testadd \(p. 86\)](#), [Equation::testdrop \(p. 87\)](#), [Equation::fixedtest \(p. 51\)](#), and [Equation::wald \(p. 93\)](#).

representations	Equation Views
-----------------	--------------------------------

Display text of specification for equation objects.

Syntax

`equation_name.representation(options)`

Options

p	Print the representation text.
---	--------------------------------

Examples

```
eq1.representations
```

displays the specifications of the equation object EQ1.

reset	Equation Views
-------	--------------------------------

Compute Ramsey’s regression specification error test.

Syntax

`eq_name.reset(n, options)`

You must provide the number of powers of fitted terms *n* to include in the test regression.

Options

p	Print the test result.
---	------------------------

Examples

```
equation eq1.ls lwage c edu race gender
eq1.reset(2)
```

carries out the RESET test by including the square and the cube of the fitted values in the test equation.

Cross-references

See [“Ramsey’s RESET Test” on page 170](#) of the *User’s Guide II* for a discussion of the RESET test.

resids	Equation Views
--------	--------------------------------

Display residuals.

The `resids` command allows you to display the actual, fitted values and residuals in either tabular or graphical form.

Syntax

```
equation_name.resids(options)
```

Options

<code>g</code> (<i>default</i>)	Display graph(s) of residuals.
<code>t</code>	Display table(s) of residuals.
<code>p</code>	Print the table/graph.

Examples

```
equation eq1.ls m1 c inc tb3 ar(1)
eq1.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.resids(g)
```

displays a graph of the actual, fitted, and residual series.

Cross-references

See also [Equation::makesids](#) (p. 69).

results	Equation Views
---------	--------------------------------

Displays the results view of an estimated equation.

Syntax

```
equation_name.results(options)
```

Options

<code>p</code>	Print the view.
----------------	-----------------

Examples

```
equation eq1.ls m1 c inc tb3 ar(1)
```

```
eq1.results(p)
```

estimates an equation using least squares, and displays and prints the results.

rls	Equation Views
-----	--------------------------------

Recursive least squares regression.

The `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 may plot various statistics from `rls` by choosing an option.

Syntax

```
eq_name.rls(options) c(1) c(2) ...
```

Options

<code>r</code>	Plot the recursive residuals about the zero line with plus and minus two standard errors.
<code>r,s</code>	Plot the recursive residuals and save the residual series and their standard errors as series named <code>R_RES</code> and <code>R_RESSE</code> , respectively.
<code>c</code>	Plot the recursive coefficient estimates with two standard error bands.
<code>c,s</code>	Plot the listed recursive coefficients and save all coefficients and their standard errors as series named <code>R_C1</code> , <code>R_C1SE</code> , <code>R_C2</code> , <code>R_C2SE</code> , and so on.
<code>o</code>	Plot the p -values of recursive one-step Chow forecast tests.
<code>n</code>	Plot the p -values of recursive n -step Chow forecast tests.
<code>q</code>	Plot the CUSUM (standardized cumulative recursive residual) and 5 percent critical lines.
<code>v</code>	Plot the CUSUMSQ (CUSUM of squares) statistic and 5 percent critical lines.
<code>p</code>	Print the view.

Examples

```
equation eq1.ls m1 c tb3 gdp
eq1.rls(r,s)
eq1.rls(c) c(2) c(3)
```


plots and saves the recursive residual series from EQ1 and their standard errors as R_RES and R_RESSE. The third line plots the recursive slope coefficients of EQ1.

```
equation eq2.ls m1 c pdl(tb3,12,3) pdl(gdp,12,3)
eq2.rls(c) c(3)
eq2.rls(q)
```

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 171](#) of the *User’s Guide II*.

steps	Equation Methods
-------	----------------------------------

Estimation by stepwise least squares.

Syntax

```
eq_name.steps(options) y x1 [x2 x3 ...] @ z1 z2 z3
```

Specify the dependent variable followed by a list of variables to be included in the regression, but not part of the search routine, followed by an “@” symbol and a list of variables to be part of the search routine. If no included variables are required, simply follow the dependent variable with an “@” symbol and the list of search variables.

Options

method = <i>arg</i>	Stepwise regression method: “stepwise” (default), “uni” (uni-directional), “swap” (swapwise), “comb” (combinatorial).
nvars = <i>int</i>	Set the number of search regressors. Required for swapwise and combinatorial methods, optional for uni-directional and stepwise methods.

Stepwise and uni-directional method options

back	Set stepwise or uni-directional method to run backward. If omitted, the method runs forward.
tstat	Use <i>t</i> -statistic values as a stopping criterion. (Default uses <i>p</i> -values).
ftol = <i>number</i> (default = 0.5)	Set forward stopping criterion value.

`btol = number` Set backward stopping criterion value.
(*default* = 0.5)

`fmaxstep = int` Set the maximum number of steps forward.
(*default* = 1000)

`bmaxstep = int` Set the maximum number of steps backward.
(*default* = 1000)

`tmaxstep = int` Set the maximum total number of steps.
(*default* = 2000)

Swapwise method options

`minr2` Use minimum R-squared increments. (Default uses maximum R-squared increments.)

Combinatorial method options

`force` Suppress the warning message issued when a large number of regressions will be performed.

Examples

```
eq1.stepsls(method=comb,nvars=3) y c @ x1 x2 x3 x4 x5 x6 x7 x8
```

performs a combinatorial search routine to search for the three variables from the set of X1, X2, ..., X8, yielding the largest R-squared in a regression of Y on a constant and those three variables.

Cross-references

See [“Stepwise Least Squares Regression,” beginning on page 55.](#)

testadd	Equation Views
----------------	--------------------------------

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

```
eq_name.testadd(options) arg1 [arg2 arg3 ...]
```

List the names of the series or groups of series to test for omission after the keyword.

Options

p	Print output from the test.
---	-----------------------------

Examples

```
equation oldeq.ls sales c adver lsales ar(1)
oldeq.testadd gdp gdp(-1)
```

tests whether GDP and GDP(-1) belong in the specification for SALES using the equation OLDEQ.

Cross-references

See [“Coefficient Tests” on page 142](#) of the *User’s Guide II* for further discussion.

See also [Equation::testadd \(p. 87\)](#) and [Equation::wald \(p. 93\)](#).

testdrop	Equation Views
----------	--------------------------------

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

```
eq_name.testdrop(options) arg1 [arg2 arg3 ...]
```

List the names of the series or groups of series to test for omission after the keyword.

Options

p	Print output from the test.
---	-----------------------------

Examples

```
equation oldeq.ls sales c adver lsales ar(1)
oldeq.testdrop adver
```

tests whether ADVER should be excluded from the specification for SALES using a the equation OLDEQ.

Cross-references

See [“Coefficient Tests” on page 142](#) of the *User’s Guide II* for further discussion of testing coefficients.

See also [Equation::testadd \(p. 86\)](#) and [Equation::wald \(p. 93\)](#).

testfit	Equation Views
---------	--------------------------------

Carry out the Hosmer-Lemeshow and/or Andrews goodness-of-fit tests for estimated binary models.

Syntax

binary_equation.testfit(*options*)

Options

h	Group by the predicted values of the estimated equation.
s = <i>series_name</i>	Group by the specified series.
<i>integer</i> (default = 10)	Specify the number of quantile groups in which to classify observations.
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 = <i>integer</i> (default = 100)	Limit the number of values to use for grouping. Should be used with the “v” option.
p	Print the result of the test.

Examples

```
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

See [“Goodness-of-Fit Tests” on page 219](#) of the *User’s Guide II* for a discussion of the Andrews and Hosmer-Lemeshow tests.

tsls	Equation Methods
------	----------------------------------

Two-stage least squares.

Carries out estimation for equations using two-stage least squares.

Syntax

```
eq_name.tsls(options) y x1 [x2 x3 ...] @ z1 [z2 z3 ...]
```

```
eq_name.tsls(options) specification @ z1 [z2 z3 ...]
```

To use the `tsls` command, 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. You may estimate nonlinear equations or equations specified with formulas by first providing a specification, then listing the instrumental variables after an “@”-sign.

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.

Options

General options

<code>m = integer</code>	Set maximum number of iterations.
<code>c = number</code>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
<code>deriv = keyword</code>	Set derivative methods. The argument <i>keyword</i> 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.
<code>showopts / -showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
<code>p</code>	Print estimation results.

Additional Options for Non-Panel Equation estimation

<code>w = series_name</code>	Weighted TSLS. Each observation will be weighted by multiplying by the specified series.
<code>h</code>	White’s heteroskedasticity consistent standard errors.
<code>n</code>	Newey-West heteroskedasticity and autocorrelation consistent (HAC) standard errors.

s	Use the current coefficient values in “C” as starting values for equations with AR or MA terms (see also param (p. 761)).
s = <i>number</i>	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”.
z	Turn off backcasting in ARMA models.

Additional Options for Panel Equation estimation

cx = <i>arg</i>	Cross-section effects. For fixed effects estimation, use “cx = f”; for random effects estimation, use “cx = r”.
per = <i>arg</i>	Period effects. For fixed effects estimation, use “cx = f”; for random effects estimation, use “cx = r”.
wgt = <i>arg</i>	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”).
cov = <i>arg</i>	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 = <i>arg</i> (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 = <i>arg</i>	Specify the name of the coefficient vector (if specified by list); the default is to use the “C” coefficient vector.

<code>iter = arg</code> (<i>default</i> = “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.
<code>s</code>	Use the current coefficient values in “C” as starting values for equations with AR or MA terms (see also param (p. 761)).
<code>s = number</code>	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”.

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), and 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.

Cross-references

See [“Two-stage Least Squares” on page 37](#) and [“Two-Stage Least Squares” on page 309](#) of the *User’s Guide II* for details on two-stage least squares estimation in single equations and systems, respectively. [“Instrumental Variables” on page 502](#) of the *User’s Guide II* discusses estimation using pool objects, while [“Instrumental Variables Estimation” on page 544](#) of the *User’s Guide II* discusses estimation in panel structured workfiles.

See also [Equation::ls \(p. 62\)](#).

ubreak	Equation Views
--------	--------------------------------

Andrews-Quandt test for unknown breakpoint.

Carries out the Andrews-Quandt test for parameter stability at some unknown breakpoint.

Syntax

```
eq_name.ubreak(options) trimlevel @ x1 x2 x3
```

You must provide the level of trimming of the data. The level must be one of the following: 49, 48, 47, 45, 40, 35, 30, 25, 20, 15, 10, or 5. If the equation is specified by list and contains no nonlinear terms, you may specify a subset of the regressors to be tested for a breakpoint after an “@” sign.

Options

wfname = series_name	Store the individual Wald F -statistics into the series <i>series_name</i> .
lname = series_name	Store the individual likelihood ratio F -statistics into the series <i>series_name</i> .
p	Print the result of the test.

Examples

```
equation ppp.lns log(spot) c log(p_us) log(p_uk)
ppp.ubreak 15
```

regresses the log of SPOT on a constant, the log of P_US, and the log of P_UK, and then carries out the Andrews-Quandt test, trimming 15% of the data from each end.

To test whether only the constant term and the coefficient on the log of P_US are subject to a structural break, use:

```
ppp.ubreak @ c log(p_us)
```

Cross-references

See [“Quandt-Andrews Breakpoint Test” on page 166](#) of the *User’s Guide II* for further discussion.

See also [Equation::chow \(p. 41\)](#) and [Equation::rls \(p. 84\)](#).

updatecoefs	Equation Procs
--------------------	--------------------------------

Update coefficient object values from an equation object.

Copies coefficients from the equation object into the appropriate coefficient vector or vectors.

Syntax

```
equation_name.updatecoef
```

Follow the name of the equation object with a period and the keyword `updatecoef`.

Examples

```
equation eq1.ls y c x1 x2 x3
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::coef](#) (p. 16).

wald	Equation Views
-------------	--------------------------------

Wald coefficient restriction test.

The `wald` view carries out a Wald test of coefficient restrictions for an equation object.

Syntax

```
equation_name.wald restrictions
```

Enter the equation name, followed by a period, and the keyword. You must provide a list of the coefficient restrictions, with joint (multiple) coefficient restrictions 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.

```
eq2.wald c(2)=c(3)*c(4)
```

tests the non-linear restriction that the second coefficient in equation EQ2 is equal to the product of the third and fourth coefficients.

Cross-references

See [“Wald Test \(Coefficient Restrictions\)” on page 145](#) of the *User’s Guide II* for a discussion of Wald tests.

See also [Equation::cellipse \(p. 39\)](#), [Equation::testdrop \(p. 87\)](#), [Equation::testadd \(p. 86\)](#).

white	Equation Views
-------	--------------------------------

Performs White’s test for heteroskedasticity of residuals.

Carries out White’s test for heteroskedasticity of the residuals of the specified equation. 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 that includes the cross-products.

White’s test is not available for equations estimated by `binary`, `ordered`, `censored`, or `count`.

Note that a more general version of the White test is available using [Equation::hettest \(p. 59\)](#). We also note that for equations estimated without a constant term, version 6 of the White command will, by default, generate results that differ from version 5. You may obtain version 5 compatible results by adding the `@comp` keyword to `white` as in:

```
eq_name.white @comp
```

Syntax

```
eq_name.white(options)
```

Options

c	Include all possible nonredundant cross-product terms in the test regression.
p	Print the test results.

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 158](#) of the *User's Guide II* for a discussion of White's test. For the multivariate version of this test, see [“White Heteroskedasticity Test” on page 352](#) of the *User's Guide II*.

See also [Equation::hetttest \(p. 59\)](#) for a more full-featured version of this test.

Factor

Factor analysis object.

Factor Declaration

factor factor object declaration (p. 103).

To declare a factor object, use the `factor` keyword, followed by a name to be given to the object. See also `factest` (p. 716).

Factor Methods

gls generalized least squares estimation (p. 105).

ipf iterated principal factors estimation (p. 109).

ml maximum likelihood estimation (p. 117).

pace non-iterative partitioned covariance estimation (PACE) (p. 122).

pf principal factors estimation (p. 125).

uls unweighted least squares estimation (p. 138).

Factor Views

anticov display the anti-image covariance matrix of the observed matrix (p. 100).

eigen display table or graph of eigenvalues of observed, scaled observed, or reduced covariance matrix (p. 102).

fitstats show table of Goodness-of-Fit statistics (p. 104).

fitted show fitted and reproduced covariances (p. 105).

loadings display loadings tables or graphs (p. 113).

maxcor display maximum absolute correlations for the observed covariance matrix (p. 116).

msa compute and display Kaiser's Measure of Sampling Adequacy (MSA) (p. 120).

observed display observed covariance matrix, scaled covariance matrix, or number of observations used in analysis (p. 121).

output display main factor analysis estimation output (p. 121).

partcor show observed partial correlation matrix (p. 125).

reduced display reduced covariance matrix using initial or final uniquenesses (p. 128).

resids display residual covariance estimates (p. 129).

rotateout show rotated factors and rotation estimation results (p. 134).

scores compute factor score coefficients and scores and display results (p. 135).

smc display table of squared multiple correlations for the observed covariance matrix (p. 137).
structure display factor structure matrix (p. 138).

Factor Procs

displayname set display name for factor object (p. 101).
factnames specify names for factors (p. 103).
label label view of factor object (p. 112).
makescores compute and save factor score series (p. 114).
rotate perform an orthogonal or oblique factor rotation (p. 130).
rotateclear clear existing rotation results (p. 134).

Factor Data Members

Scalar values for model

@valid (0, 1) indicator for whether the factor object has valid factor estimates (1 = true).
@nvars number of variables to analyze.
@nfactors number of retained factors.
@obs number of observations.
@balanced (0, 1) indicator for whether the covariance matrix uses a balanced sample (1 = balanced).
@ncondition number of conditioning variables (including the constant term for centered covariances).
@pratio parsimony ratio.
@nnfi Non-normed Fit Index (generalized Tucker-Lewis index).
@rfi Bollen's Relative Fit Index.
@nfi Bentler-Bonnet's Incremental Fit Index.
@ifi Bollen's Incremental Fit Index.
@cfi Bentlers Comparative Fit Index.

Scalar values for model and independence (zero factor) specifications

Each of the following takes an optional argument “(0)” (e.g., “@params(0)”). If no argument is provided, the data member returns the value for the estimated factor specification. If the optional argument is provided, the member returns the value for the independence (zero factor) model.

@params[(0)] number of estimated parameters.
@ncoefs[(0)] same as @parms.
@objective[(0)] value of the objective function in factor extraction.
@discrep[(0)] same as @objective.
@aic[(0)] Akaike Information Criterion.

@sc[(0)] Schwarz Information Criterion.
@hq[(0)] Hannan-Quinn Information Criterion.
@ecvi[(0)] Expected Cross-validation Index.
@chisq[(0)] Chi-square test statistic for model adequacy.
@chisqdf[(0)] Degrees of freedom for the chi-square statistic.
@chisqprob[(0)] p -value for the chi-square statistic
@bartlett[(0)] Bartlett's adjusted version of the Chi-square test statistic.
@bartlettprob[(0)] . p -value for Bartlett's adjusted version of the chi-square statistic.
@rmsr[(0)] Root mean square residuals.
@srmsr[(0)] Standardized root mean square residuals.
@gfi[(0)] Jöreskog and Sörbom Generalized Fit Index.
@agfi[(0)] Jöreskog and Sörbom Adjusted Generalized Fit Index.
@noncent[(0)] Noncentrality parameter.
@gammahat[(0)] ... Gamma hat non-centrality.
@mdnoncent[(0)] .. McDonald non-centrality.
@rmsea[(0)] Root MSE approximation.

Vectors and Matrices for Model

@obsmat matrix of number of observations used for each pair of variables.
@cov observed covariance or correlation matrix.
@scaled scaled covariance matrix.
@fitted fitted covariance matrix.
@common common variance fitted covariance matrix (fitted matrix with communality on the diagonal).
@resid residual matrix (observed–fitted).
@residcommon residual matrix using common variance.
@reduced reduced covariance matrix using final uniqueness estimates.
@ireduced reduced covariance matrix using initial uniqueness estimates.
@anticov Anti-image covariance matrix.
@partcor partial correlation matrix.
@iunique vector of initial uniqueness estimates.
@unique vector of final uniqueness estimates.
@icommunal vector initial communality estimates.
@communal vector of final communality estimates.
@rowadjust vector of row standardization terms (used to rescale results so that the uniqueness and communality estimates add up to the observed diagonals).
@loadings estimated loadings matrix.
@rloadings rotated loadings matrix.

`@rotmat`..... factor rotation matrix: T .
`@rotmatinv` loadings rotation matrix: $(T^{-1})'$.
`@factcor` factor correlation matrix.
`@factstruct`..... factor structure matrix (correlation between factors and the variables).

String Values

`@factnames` factor names.
`@varnames` variable names.

Factor Examples

To declare a factor object named F1:

```
factor f1
```

To declare and estimate by maximum likelihood a factor object F2 using data in the group GROUP01:

```
factor f2.ml group01
```

To declare and estimate, using iterated principal factors, the factor object F3 using the sym matrix SYM01:

```
factor f3.ipf sym01 785
```

In addition to providing the name of the matrix, we indicate that the covariance is computed using 785 observations.

To estimate a factor model by ML using the series X1 X2 and X3 using a command:

```
factest x1 x2 x3
```

EViews will create an untitled factor object containing the results of the estimation.

Factor Entries

The following section provides an alphabetical listing of the commands associated with the “Factor” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

anticov	Factor Views
---------	------------------------------

Display the anti-image covariance matrix based on the observed covariance matrix

Syntax

```
factor_name.anticov(options)
```


The anti-image covariance is obtained by taking the inverse of the covariance matrix, and row and column scaling by the diagonals of the inverse.

The diagonal elements of the matrix are equal to 1 minus the squared multiple correlations (SMCs). The off-diagonal elements of the anti-image covariance are equal to the negative of the partial covariances multiplied by $(1 - \rho^2_{xy|Z})$, where Z are the remaining variables

Options

p	Print the matrix.
---	-------------------

Examples

```
factor f1.ml group01
f1.anticov(p)
```

estimates the factor analysis object F1, then displays and prints the anti-image covariance matrix.

Cross-References

See “Observed Covariances” on page 591 of the *User’s Guide II*. See also [Factor::observed](#) (p. 121), [Factor::partcor](#) (p. 125), [Factor::smc](#) (p. 137).

displayname	Factor Procs
-------------	------------------------------

Set display name for factor object.

Attaches a display name to a factor object which may be used to label output in place of the standard factor object name.

Syntax

```
factor_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

```
f1.displayname Holzinger Example
```

The first line attaches a display name “Holzinger Example” to the factor object F1.

Cross-references

See “Labeling Objects” on page 72 of the *User’s Guide I* for a discussion of labels and display names. See also [Factor::label](#) (p. 112).

eigen	Factor Views
-------	------------------------------

Display table or graph of eigenvalues of observed, scaled observed, or reduced covariance matrix.

Syntax

```
factor_name.eigen(options)
```

By default, `eigen` will display a table of eigenvalues for the specified source matrix. You may add the option keywords “eigvec” and “matrix” to include additional output.

To display a graph of the results, you should some combination of the “scree”, “diff” and “cproport” option keywords.

Options

source = arg (default = observed)	Source matrix to be analyzed: “observed” (observed covariance matrix), “scaled” (scaled observed matrix), “reducedinit” (reduced using initial uniquenesses), “reduced” (reduced using final uniquenesses).
eigvec	Add the eigenvectors to the table of eigenvalue results. May be combined with the “matrix” keyword.
matrix	Display the source matrix along with the table of eigenvalue results. May be combined with the “eigvec” keyword.
scree	Display eigenvalue graph of the ordered eigenvalues (Scree plot). May be combined with the “diff” and “cproport” keywords.
diff	Display graph of the difference in successive eigenvalues. May be combined with the “scree” and “cproport” keywords.
cproport	Display graph of the cumulative proportion of total variance associated with each eigenvalue/eigenvector. May be combined with the “scree” and “diff” keywords.
p	Print results.

Examples

```
f1.eigen(source=observed, scree)
```

displays the scree plot based on the observed covariance matrix.

```
f1.eigen(source=reducedinit, eigvec, matrix)
```

displays a table of eigenvalues and corresponding eigenvectors for the reduced covariance matrix (using the initial uniquenesses). The table also shows the reduced covariance matrix.

```
f1.eigen(source=reducedinit, scree, cproport, diff)
```

shows the scree, cumulative proportion, and eigenvalue difference graphs based on the reduced initial covariance.

Cross-references

See [“Eigenvalues” on page 593](#) of the *User’s Guide II*.

factnames	Factor Procs
------------------	------------------------------

Specify names for the unobserved factors.

Assign names to the unobserved factors in an estimated factor object. These names will subsequently be used in table and graphical output.

Syntax

```
factor_name.factnames [name1 ...]
```

You should follow the keyword with a list of names for the factors. You may clear an existing set of factnames by using the `factnames` keyword with an empty list of factors.

Examples

```
f1.factnames Verbal Visual
```

attaches names “Verbal” and “Visual” to the first two retained factors. The names will be used in subsequent views and procedures.

```
f1.factnames
```

clears the existing list of factor names.

factor	Factor Declaration
---------------	------------------------------------

Declare a factor object.

Syntax

```
factor factor_name
```

```
factor factor_name.method(options) specification
```

Follow the `factor` keyword with a name and an optional specification. If you wish to enter the specification, you should follow the new factor name with a period, an estimation method, and the factor analysis specification. Valid estimation methods are [gls](#) ([p. 105](#)),

[ipf](#) (p. 109), [ml](#) (p. 117), [pace](#) (p. 122), [pf](#) (p. 125), and [uls](#) (p. 138). Refer to each method for a description of the available options.

Examples

```
factor f1.gls(n=map, priors=max) group01
```

declares the factor object F1 and estimates a factor model from the correlation matrix for the series in the group object GROUP01. The default method, Velicer’s MAP, is used for determining the number of factors.

```
factor fac1.ipf(n=2, maxit=4) var1 var2 var3 var4
```

creates the factor object FAC1 then extracts two factors from the variables VAR1–VAR4 by the iterative principal factor method, with a maximum of four iterations.

```
factor f2.ml group01
```

declares the factor object F2 then estimates the factor model using the correlation matrix for the series in GROUP01 by maximum likelihood method.

Cross-references

[Chapter 24. “Basic Regression,” on page 5](#) of the *User’s Guide II* provides basic information on estimation and equation objects.

fitstats	Factor Views
-----------------	------------------------------

Display Goodness-of-fit statistics for an estimated factor analysis object.

Syntax

```
factor_name.fitstats
```

Options

p	Print the results.
---	--------------------

Examples

```
factor f1.ml group01  
  
f1.fitstats(p)
```

estimates a factor model then displays and prints a table of Goodness-of-fit statistics.

Cross-references

See [“Discrepancy and Chi-Square Tests” on page 615](#) of the *User’s Guide II*.

fitted	Factor Views
--------	------------------------------

Display fitted and common covariances from a factor analysis object.

Syntax

```
factor_name.fitted(options)
```

Options

common	Display common covariance.(default is to display the fitted covariance).
p	Print the matrix.

Examples

```
factor f1.ml group01
f1.fitted(p)
```

estimates a factor model for the series in GROUP01, then displays and prints the fitted covariance matrix for the factor object F1.

```
f1.fitted(common)
```

displays the estimate of the fitted common variance.

Cross-references

See “[Matrix Views](#)” on page 591 of the *User’s Guide II*. See also [Factor::reduced](#) (p. 128).

gls	Factor Methods
-----	--------------------------------

Generalized least squares estimation of the factor model.

Syntax

```
factor_name.gls(options) x1 [x2 x3...] [@partial z1 z2 z3...]
factor_name.gls(options) matrix_name [[obs] [conditioning]] [@ name1 name2
name3...]
```

The first method computes the observed dispersion matrix from a set of series or group objects. Simply append a period and the `gls` keyword to the name of your object, followed by the names of your series and groups. You may optionally use the keyword `@partial` and append a list of conditioning series.

In the second method you will provide the name of the observed dispersion matrix, and optionally, the number of observations and the rank of the set of conditioning variables. If

the latter is not provided, it will be set to 1 (representing the constant in the standard centered variance calculations). You may also provide names for the columns of the correlation matrix by entering the @-sign followed by a list of valid series names.

Options

Estimation Options

rescale	Rescale the uniqueness and loadings estimates so that they match the observed variances.
maxit = <i>integer</i>	Maximum number of iterations.
conv = <i>scalar</i>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled estimates. The criterion will be set to the nearest value between 1e-24 and 0.2.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the rotation output.
p	Print basic estimation results.

Number of Factors Options

n = <i>arg</i> (default = "map")	Number of factors: "kaiser" (Kaiser-Guttman greater than mean), "mineigen" (Minimum eigenvalue criterion; specified using "eiglimit"), "varfrac" (fraction of variance accounted for; specified using "varlimit"), "map" (Velicer's Minimum Average Partial method), "bstick" (comparison with broken stick distribution), "parallel" (parallel analysis: number of replications specified using "pnreps"; "pquant" indicates the quantile method value if employed), "scree" (standard error scree method), <i>integer</i> (user-specified integer value).
eiglimit = <i>number</i> (default = 1)	Limit value for retaining factors using the eigenvalue comparison (where "n = mineigen").
varlimit = <i>number</i> (default = 0.5)	Fraction of total variance explained limit for retaining factors using the variance limit criterion (where "n = varlimit").
porig	Use the unreduced matrix for parallel analysis (the default is to use the reduced matrix). For parallel analysis only ("n = parallel").
preps = <i>integer</i> (default = 100)	Number of parallel analysis repetitions. For parallel analysis only ("n = parallel").

<code>pquant = number</code>	Quantile value for parallel analysis comparison (if not specified, the mean value will be employed). For parallel analysis only (“n = parallel”).
<code>pseed = positive integer</code>	Seed the random number generator for parallel analysis. If not specified, EViews will seed the random number generator with a single integer draw from the default global random number generator. For parallel analysis only (“n = parallel”).
<code>prnd = arg</code> (<i>default</i> = “kn” or method previously set using <code>rndseed</code> (p. 770))	Type of random number generator for the simulation: improved Knuth generator (“kn”), improved Mersenne Twister (“mt”), Knuth’s (1997) lagged Fibonacci generator used in EViews 4 (“kn4”) L’Ecuyer’s (1999) combined multiple recursive generator (“le”), Matsumoto and Nishimura’s (1998) Mersenne Twister used in EViews 4 (“mt4”). For parallel analysis only (“n = parallel”).

Initial Communalities Options

<code>priors = arg</code>	Method for obtaining initial communalities: “smc” (squared multiple correlations), “max” (maximum absolute correlation), “pace” (noniterative partitioned covariance estimation), “frac” (fraction of the diagonals of the original matrix; specified using “priorfrac = ”), “random” (random fractions of the original diagonals), “user” (user-specified vector; specified using “priorunique”).
<code>priorfrac = number</code>	User-specified common fraction (between 0 and 1) to be used when “priors = frac”.
<code>priorunique = arg</code>	Vector of initial <i>uniqueness</i> estimates to be used when “priors = user”. By default, the values will be taken from the corresponding elements of the coefficient vector C.

Covariance Options

<code>cov = arg</code> (<i>default</i> = “cov”)	Covariance calculation method: ordinary (Pearson product moment) covariance (“cov”), ordinary correlation (“corr”), Spearman rank covariance (“rcov”), Spearman rank correlation (“rcorr”), Kendall’s tau-b (“taub”), Kendall’s tau-a (“taua”), uncentered ordinary covariance (“ucov”), uncentered ordinary correlation (“ucorr”). User-specified covariances are indicated by specifying a sym matrix object in place of a list of series or groups in the command.
---	--

<code>wgt = name</code> (optional)	Name of series containing weights.
<code>wgtmethod = arg</code> (default = "sstdev")	Weighting method (when weights are specified using "weight = "): frequency ("freq"), inverse of variances ("var"), inverse of standard deviation ("stdev"), scaled inverse of variances ("svar"), scaled inverse of standard deviations ("sstdev"). Only applicable for ordinary (Pearson) calculations. Weights specified by "wgt = " are frequency weights for rank correlation and Kendall's tau calculations.
<code>pairwise</code>	Compute using pairwise deletion of observations with missing cases (pairwise samples).
<code>df</code>	Compute covariances with a degree-of-freedom correction for the mean (for centered specifications), and any partial conditioning variables.

Examples

```
factor f1.gls(n=map, priors=max) group01
```

declares the factor object F1 and estimates a factor model from the correlation matrix for the series in the group object GROUP01. The default method, Velicer's MAP, is used for determining the number of factors.

```
f1.gls(n=map, priors=max) group01 @partial ser1 ser2
```

estimates the same specification using the partial correlation for the series in GROUP01, conditional on the series SER1 and SER2.

```
f1.gls(rescale, maxit=200, n=2, priors=smc, cov=rcorr) x y z
```

estimates a two factor model for the rank correlation computed from the series X, Y, and Z, using generalized least squares with 200 maximum iterations. The result is rescaled if necessary so that estimated uniqueness and the communality sum to 1; the initial uniquenesses are set to the SMCs of the observed correlation matrix.

```
f1.gls sym01 393
```

estimates a factor model using the symmetric matrix object as the observed matrix. The number of observations for the model is set to 393.

Cross-references

See [Chapter 40. "Factor Analysis," on page 579](#) of the *User's Guide II* for a general discussion of factor analysis. The various estimation methods are described in ["Estimation Methods" on page 612](#) of the *User's Guide II*.

See also `Factor::ipf` (p. 109), `Factor::ml` (p. 117), `Factor::pace` (p. 122), `Factor::pf` (p. 125), `Factor::uls` (p. 138).

ipf	Factor Methods
-----	----------------

Iterated principal factors estimation of the factor model.

Syntax

```
factor_name.ipf(options) x1 [x2 x3...] [@partial z1 z2 z3...]
factor_name.ipf(options) matrix_name [[obs] [conditioning]] [@ name1 name2
name3...]
```

The first method computes the observed dispersion matrix from a set of series or group objects. Simply append a period and the `ipf` keyword to the name of your object, followed by the names of your series and groups. You may optionally use the keyword `@partial` and append a list of conditioning series.

In the second method you will provide the name of the observed dispersion matrix, and optionally, the number of observations and the rank of the set of conditioning variables. If the latter is not provided, it will be set to 1 (representing the constant in the standard centered variance calculations). You may also provide names for the columns of the correlation matrix by entering the `@`-sign followed by a list of valid series names.

Options

Estimation Options

<code>heywood = arg</code> (<i>default</i> = “stop”)	Method for handling Heywood cases (negative uniqueness estimates): “stop” (stop and report final results), “last” (stop and report previous iteration results), “reset” (set negative uniquenesses to zero and continue), “ignore” (ignore and continue).
<code>maxit = integer</code>	Maximum number of iterations.
<code>conv = scalar</code>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled estimates. The criterion will be set to the nearest value between 1e-24 and 0.2.
<code>showopts / -showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the rotation output.
<code>p</code>	Print basic estimation results.

Number of Factors Options

<code>n = arg</code> (<i>default</i> = “map”)	Number of factors: “kaiser” (Kaiser-Guttman greater than mean), “mineigen” (Minimum eigenvalue criterion; specified using “eiglimit”), “varfrac” (fraction of variance accounted for; specified using “varlimit”), “map” (Velicer’s Minimum Average Partial method), “bstick” (comparison with broken stick distribution), “parallel” (parallel analysis: number of replications specified using “pnreps”; “pquant” indicates the quantile method value if employed), “scree” (standard error scree method), <i>integer</i> (user-specified integer value).
<code>eiglimit = number</code> (<i>default</i> = 1)	Limit value for retaining factors using the eigenvalue comparison (where “n = mineigen”).
<code>varlimit = number</code> (<i>default</i> = 0.5)	Fraction of total variance explained limit for retaining factors using the variance limit criterion (where “n = varlimit”).
<code>porig</code>	Use the unreduced matrix for parallel analysis (the default is to use the reduced matrix). For parallel analysis only (“n = parallel”).
<code>preps = integer</code> (<i>default</i> = 100)	Number of parallel analysis repetitions. For parallel analysis only (“n = parallel”).
<code>pquant = number</code>	Quantile value for parallel analysis comparison (if not specified, the mean value will be employed). For parallel analysis only (“n = parallel”).
<code>pseed = positive integer</code>	Seed the random number generator for parallel analysis. If not specified, EViews will seed the random number generator with a single integer draw from the default global random number generator. For parallel analysis only (“n = parallel”).
<code>prnd = arg</code> (<i>default</i> = “kn” or method previously set using rndseed (p. 770))	Type of random number generator for the simulation: improved Knuth generator (“kn”), improved Mersenne Twister (“mt”), Knuth’s (1997) lagged Fibonacci generator used in EViews 4 (“kn4”) L’Ecuyer’s (1999) combined multiple recursive generator (“le”), Matsumoto and Nishimura’s (1998) Mersenne Twister used in EViews 4 (“mt4”). For parallel analysis only (“n = parallel”).

Initial Communalities Options

<code>priors = arg</code>	Method for obtaining initial communalities: “smc” (squared multiple correlations), “max” (maximum absolute correlation), “pace” (noniterative partitioned covariance estimation), “frac” (fraction of the diagonals of the original matrix; specified using “priorfrac = ”), “random” (random fractions of the original diagonals), “user” (user-specified vector; specified using “priorunique”).
<code>priorfrac = number</code>	User-specified common fraction (between 0 and 1) to be used when “priors = frac”.
<code>priorunique = arg</code>	Vector of initial <i>uniqueness</i> estimates to be used when “priors = user”. By default, the values will be taken from the corresponding elements of the coefficient vector C.

Covariance Options

<code>cov = arg</code> (default = “cov”)	Covariance calculation method: ordinary (Pearson product moment) covariance (“cov”), ordinary correlation (“corr”), Spearman rank covariance (“rcov”), Spearman rank correlation (“rcorr”), Kendall’s tau-b (“taub”), Kendall’s tau-a (“taua”), uncentered ordinary covariance (“ucov”), uncentered ordinary correlation (“ucorr”). User-specified covariances are indicated by specifying a sym matrix object in place of a list of series or groups in the command.
<code>wgt = name</code> (optional)	Name of series containing weights.
<code>wgtmethod = arg</code> (default = “sstdev”)	Weighting method (when weights are specified using “weight = ”): frequency (“freq”), inverse of variances (“var”), inverse of standard deviation (“stdev”), scaled inverse of variances (“svar”), scaled inverse of standard deviations (“sstdev”). Only applicable for ordinary (Pearson) calculations. Weights specified by “wgt = ” are frequency weights for rank correlation and Kendall’s tau calculations.
<code>pairwise</code>	Compute using pairwise deletion of observations with missing cases (pairwise samples).
<code>df</code>	Compute covariances with a degree-of-freedom correction for the mean (for centered specifications), and any partial conditioning variables.

Examples

```
factor f1.ipf(n=2, maxit=4) var1 var2 var3 var4
```

declares the factor object F1 then extracts two factors from the variables VAR1–VAR4 by the iterative principal factor method, with a maximum of four iterations.

```
f1.ipf(conv=1e-9, heywood=reset) group01
```

sets the convergence criterion to 1e-9, and estimates the factor model for the series in GROUP01. If encountered, negative uniqueness estimates will be set to zero and the estimation will proceed.

```
f1.ipf(conv=1e-9, heywood=reset) group01 @partial ser1 ser2
```

estimates the same specification using the partial correlation for GROUP01, conditional on the series SER1 and SER2.

```
f1.ipf(n=parallel) sym01 424
```

estimates the iterative principal factor model using the observed matrix SYM01. The number of observations is 424, and the number of factors is determined using parallel analysis.

Cross-references

See [Chapter 40. “Factor Analysis,” on page 579](#) of the *User’s Guide II* for a general discussion of factor analysis. The various estimation methods are described in [“Estimation Methods” on page 612](#) of the *User’s Guide II*.

See also [Factor::gls \(p. 105\)](#), [Factor::ml \(p. 117\)](#), [Factor::pace \(p. 122\)](#), [Factor::pf \(p. 125\)](#), [Factor::uls \(p. 138\)](#).

label	Factor Views Factor Procs
-------	---

Display or change the label view of the factor object.

Syntax

```
factor_name.label
factor_name.label(options) [text]
```

Options

The first version of the command displays the label view of the factor. The second version may be used to modify the label. 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 <i>text</i> .
s	Sets the source field to <i>text</i> .

u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

If no options are provided, `label` will display the current values in the label.

Examples

The following lines replace the remarks field of F1 with “Example factor analysis problem”:

```
f1.label(r) Example factor analysis problem
```

To append additional remarks to F1, and then to print the label view:

```
f1.label(r, p) Test evaluation
```

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels.

See also [Factor::displayname \(p. 101\)](#).

loadings	Factor Views
----------	------------------------------

Display factor loadings tables or graphs.

Syntax

```
factor_name.loadings(options)
```

```
factor_name.loadings(graph, options) [graph_list]
```

where the *[graph_list]* is an optional list of integers and/or vectors containing integers identifying the factors to plot. If *graph_list* is not provided, EViews will construct graphs using all of the retained factors.

Multiple pairs are handled using the method specified in the “mult = ” option. Note that the order of elements in the list matters; reversing the order of two indices reverses the axis on which each factor is displayed.

Options

graph	Display graphs of the loadings (default is to display the loadings in a spreadsheet view).
unrotated	Use the unrotated loadings (default is to use the rotated loadings, if available).
p	Print results.

Graph Options

<code>mult = arg</code> (<i>default</i> = “first”)	Multiple series handling: plot first against remainder (“first”), plot as x-y pairs (“pair”), lower-triangular plot (“lt”).
<code>nocenter</code>	Do not center graphs around the origin. By default, EViews centers biplots around (0, 0).

Examples

```
f1.loadings
```

displays the spreadsheet view of the (possibly rotated) loadings.

```
f1.loadings(graph, unrotated) 1 2
```

displays an XY graph of the first two unrotated factor loadings.

Cross-references

See “[Background](#),” [beginning on page 610](#) of the *User’s Guide II* for a general discussion of the factor model, and “[Loadings Views](#)” [on page 592](#) of the *User’s Guide II* for specific discussion of the loadings view.

makescores	Factor Procs
-------------------	------------------------------

Save estimated factor score series in the workfile

Syntax

```
factor_name.makescores(options) [output_list] [@ observed_list]
```

The optional *output_list* describes the factors that you wish to save. There are two formats for the list:

- You may specify *output_list* using a list of integers and/or vectors containing integers identifying the factors that you wish to save (e.g., “1 2 3 5”).

EViews will construct the output series names using the factor names previously specified in the factor object (using `Factor::factnames` (p. 103)) or using the default names “F1”, “F2”, etc. If a name modifier is provided (using the “append = ” option), it will be appended to each name
- You may provide an *output_list* containing names for factors to be saved (e.g., “math science verbal”).

If you provide *k* factor names, EViews will save the first *k* factors to the workfile. The factors will be named using the specified list, appended with the name modifiers, if specified.

By default, EViews will save all of the factors using the names in the factor object, with modifiers if necessary.

The optional *observed_list* of observed input variables will be multiplied by the score coefficients to compute the scores. Note that:

- If an *observed_list* is not provided, EViews will use the observed variables from factor estimation. For user-specified factor models (specified by providing a symmetric matrix) you must provide a list if you wish to obtain score values.
- Scores values will be computed for the current workfile sample. Observations with input values that are missing will generate NAs.

Options

unrotated	Use unrotated loadings in computations (the default is to use the rotated loadings, if available).
type = <i>arg</i> (default = “exact”)	Exact coefficient (“exact”), coarse adjusted factor coefficients (“coefs”), coarse adjusted factor loadings (“loadings”).
coef = <i>arg</i> (default = “reg”)	Method for computing the factor score coefficient matrix: Thurstone regression (“reg”), Ideal Variables (“ideal”), Bartlett weighted least squares (“wls”), generalized Anderson-Rubin-McDonald (“anderson”), Green (“green”). For “type = exact” and “type = coefs” specifications.
cutoff = <i>number</i> (default = 0.3)	Cutoff value for coarse score coefficient calculation (Grice, 1991a). For “type = coef” specifications, the cutoff value represents the fraction of the largest absolute coefficient weight per factor against which the absolute exact score coefficients should be compared. For “type = loadings”, and “type = struct” specifications, the cutoff is the value against which the absolute loadings or structure coefficients should be compared.
moment = <i>arg</i> (default = “est”; if feasible)	Standardize the observables data using means and variances from: original estimation (“est”), or the computed moments from specified observable variables (“obs”). The “moment = est” option is only available for factor models estimated using Pearson or uncentered Pearson correlation and covariances since the remaining models involve unobserved or non-comparable moments.
df	Degrees-of-freedom correct the observables variances computed when “moment = obs” (divide sums-of-squares by $n - 1$ instead of n).

<code>n = arg</code>	(Optional) Name of group object to contain the factor score series.
<code>coefout</code>	(Optional) Name of matrix in which to save the factor score coefficient matrix.

Examples

```
f1.makescores(coef=green, n=outgrp)
```

computes factor scores coefficients using Green’s method, then saves the results into series in the workfile using the names in the factor object. The observed data from the estimation specification will be used as inputs to the procedure. If no names have been specified, the names will be “F1”, “F2”, *etc.* The output series will be saved in the group object OUTGRP.

```
f1.makescores(coef=green, n=outgrp) 1 2
```

computes scores in the same fashion, but only saves factors 1 and 2.

```
f1.makescores(type=coefs) sc1 sc2 sc3
```

computes coarse factor scores using the default (Thurstone) scores coefficients and saves them in the series SC1, SC2, and SC3. The observed data from the estimation specification will be used as inputs.

Cross-references

See “[Estimating Scores,](#)” beginning on [page 587](#) of the *User’s Guide II* and “[Scoring,](#)” on [page 620](#) of the *User’s Guide II*. See also `Factor::scores` ([p. 135](#)).

maxcor	Factor Views
---------------	------------------------------

Display the maximum absolute correlations for each column of the observed covariance matrix.

Syntax

```
factor_name.maxcor(options)
```

The table also displays the observed covariance matrix.

Options

<code>p</code>	Print the matrix.
----------------	-------------------

Examples

```
f1.maxcor(p)
```

displays and prints the maximum absolute covariance matrix for the factor object F1.

Cross-references

See also [Factor::anticov](#) (p. 100), [Factor::observed](#) (p. 121), and [Factor::partcor](#) (p. 125).

ml	Factor Methods
----	--------------------------------

Maximum likelihood estimation of the factor model.

Syntax

```
factor_name.ml(options) x1 [x2 x3...] [@partial z1 z2 z3...]
factor_name.ml(options) matrix_name [[obs] [conditioning]] [@ name1 name2
name3...]
```

The first method computes the observed dispersion matrix from a set of series or group objects. Simply append a period and the `ml` keyword to the name of your object, followed by the names of your series and groups. You may optionally use the keyword `@partial` and append a list of conditioning series.

In the second method you will provide the name of the observed dispersion matrix, and optionally, the number of observations and the rank of the set of conditioning variables. If the latter is not provided, it will be set to 1 (representing the constant in the standard centered variance calculations). You may also provide names for the columns of the correlation matrix by entering the `@`-sign followed by a list of valid series names.

Options

Estimation Options

<code>rescale</code>	Rescale the uniqueness and loadings estimates so that they match the observed variances.
<code>maxit = integer</code>	Maximum number of iterations.
<code>conv = scalar</code>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled estimates. The criterion will be set to the nearest value between 1e-24 and 0.2.
<code>showopts / -showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the rotation output.
<code>p</code>	Print basic estimation results.

Number of Factors Options

<code>n = arg</code> (<i>default</i> = “map”)	Number of factors: “kaiser” (Kaiser-Guttman greater than mean), “mineigen” (Minimum eigenvalue criterion; specified using “eiglimit”), “varfrac” (fraction of variance accounted for; specified using “varlimit”), “map” (Velicer’s Minimum Average Partial method), “bstick” (comparison with broken stick distribution), “parallel” (parallel analysis: number of replications specified using “pnreps”; “pquant” indicates the quantile method value if employed), “scree” (standard error scree method), <i>integer</i> (user-specified integer value).
<code>eiglimit = number</code> (<i>default</i> = 1)	Limit value for retaining factors using the eigenvalue comparison (where “n = mineigen”).
<code>varlimit = number</code> (<i>default</i> = 0.5)	Fraction of total variance explained limit for retaining factors using the variance limit criterion (where “n = varlimit”).
<code>porig</code>	Use the unreduced matrix for parallel analysis (the default is to use the reduced matrix). For parallel analysis only (“n = parallel”).
<code>preps = integer</code> (<i>default</i> = 100)	Number of parallel analysis repetitions. For parallel analysis only (“n = parallel”).
<code>pquant = number</code>	Quantile value for parallel analysis comparison (if not specified, the mean value will be employed). For parallel analysis only (“n = parallel”).
<code>pseed = positive integer</code>	Seed the random number generator for parallel analysis. If not specified, EVIEWS will seed the random number generator with a single integer draw from the default global random number generator. For parallel analysis only (“n = parallel”).
<code>prnd = arg</code> (<i>default</i> = “kn” or method previously set using rndseed (p. 770))	Type of random number generator for the simulation: improved Knuth generator (“kn”), improved Mersenne Twister (“mt”), Knuth’s (1997) lagged Fibonacci generator used in EVIEWS 4 (“kn4”) L’Ecuyer’s (1999) combined multiple recursive generator (“le”), Matsumoto and Nishimura’s (1998) Mersenne Twister used in EVIEWS 4 (“mt4”). For parallel analysis only (“n = parallel”).

Initial Communalities Options

<code>priors = arg</code>	Method for obtaining initial communalities: “smc” (squared multiple correlations), “max” (maximum absolute correlation), “pace” (noniterative partitioned covariance estimation), “frac” (fraction of the diagonals of the original matrix; specified using “priorfrac = ”), “random” (random fractions of the original diagonals), “user” (user-specified vector; specified using “priorunique”).
<code>priorfrac = number</code>	User-specified common fraction (between 0 and 1) to be used when “priors = frac”.
<code>priorunique = arg</code>	Vector of initial <i>uniqueness</i> estimates to be used when “priors = user”. By default, the values will be taken from the corresponding elements of the coefficient vector C.

Covariance Options

<code>cov = arg</code> (default = “cov”)	Covariance calculation method: ordinary (Pearson product moment) covariance (“cov”), ordinary correlation (“corr”), Spearman rank covariance (“rcov”), Spearman rank correlation (“rcorr”), Kendall’s tau-b (“taub”), Kendall’s tau-a (“taua”), uncentered ordinary covariance (“ucov”), uncentered ordinary correlation (“ucorr”). User-specified covariances are indicated by specifying a sym matrix object in place of a list of series or groups in the command.
<code>wgt = name</code> (optional)	Name of series containing weights.
<code>wgtmethod = arg</code> (default = “sstdev”)	Weighting method (when weights are specified using “weight = ”): frequency (“freq”), inverse of variances (“var”), inverse of standard deviation (“stdev”), scaled inverse of variances (“svar”), scaled inverse of standard deviations (“sstdev”). Only applicable for ordinary (Pearson) calculations. Weights specified by “wgt = ” are frequency weights for rank correlation and Kendall’s tau calculations.
<code>pairwise</code>	Compute using pairwise deletion of observations with missing cases (pairwise samples).
<code>df</code>	Compute covariances with a degree-of-freedom correction for the mean (for centered specifications), and any partial conditioning variables.

Examples

```
factor f1.ml group01
```

declares the factor object F1 then estimates the factor model using the correlation matrix for the series in GROUP01 by the method of maximum likelihood.

```
f1.ml group01 @partial ser1 ser2
```

estimates the same specification using the partial correlation for the series in GROUP01, conditional on the series SER1 and SER2.

```
f1.ml(n=parallel, priors=max) x y z
```

uses parallel analysis to determine the number of factors for a model estimates from the series X, Y, and Z, and uses the maximum absolute correlations to determine the initial uniqueness estimates.

```
f1.ml(n=scree) sym01 424
```

estimates the factor model using the observed matrix SYM01. The number of observations is 424, and the number of factors is determined using the standard error scree.

Cross-references

See [Chapter 40. “Factor Analysis,” on page 579](#) of the *User’s Guide II* for a general discussion of factor analysis. The various estimation methods are described in [“Estimation Methods” on page 612](#) of the *User’s Guide II*.

See also [Factor::gls \(p. 105\)](#), [Factor::ipf \(p. 109\)](#), [Factor::ml \(p. 117\)](#), [Factor::pace \(p. 122\)](#), [Factor::pf \(p. 125\)](#), [Factor::uls \(p. 138\)](#).

msa	Factor Views
-----	------------------------------

Display Kaiser’s Measure of Sampling Adequacy and matrix of partial correlations.

Syntax

```
factor_name.msa(options)
```

Options

p	Print the results.
---	--------------------

Examples

```
f1.msa(p)
```

displays and prints the results for the factor object F1.

Cross-references

See also [Factor::partcor \(p. 125\)](#) and [Factor::anticov \(p. 100\)](#).

observed	Factor Views
----------	------------------------------

Display observed covariance matrix, scaled observed covariance (correlation), or matrix of number of observations.

Syntax

`factor_name.observed(options)`

Options

scaled	Scale the observed matrix so that it has unit diagonals.
obs	Display the matrix containing number of observations for each covariance element.
p	Print the results.

Examples

```
factor f1.ml group01
f1.observed
```

estimates a common factor model for the series in GROUP01, then displays the observed covariance matrix.

```
f1.observed(obs, p)
```

displays and prints the matrix containing the number of observations.

```
f1.observed(scaled)
```

displays the corresponding correlation matrix.

Cross-references

See [“Observed Covariances” on page 591](#) of the *User’s Guide II*. See also [Factor::anticov](#) (p. 100), [Factor::partcor](#) (p. 125), and [Factor::smc](#) (p. 137).

output	Factor Views
--------	------------------------------

Display factor estimation output.

Syntax

`factor_name.output(options)`

Options

p	Print view.
---	-------------

Examples

```
f1.output
```

displays the estimation output for factor F1.

pace	Factor Methods
------	----------------

Non-iterative partitioned covariance estimation of the factor model

Syntax

```
factor_name.pace(options) x1 [x2 x3...] [@partial z1 z2 z3...]
factor_name.pace(options) matrix_name [[obs] [conditioning]] [@ name1 name2
name3...]
```

The first method computes the observed dispersion matrix from a set of series or group objects. Simply append a period and the `pace` keyword to the name of your object, followed by the names of your series and groups, You may optionally use the keyword `@partial` and append a list of conditioning series.

In the second method you will provide the name of the observed dispersion matrix, and optionally, the number of observations and the rank of the set of conditioning variables. If the latter is not provided, it will be set to 1 (representing the constant in the standard centered variance calculations). You may also provide names for the columns of the correlation matrix by entering the `@`-sign followed by a list of valid series names.

Options

Estimation Options

rescale	Rescale the uniqueness and loadings estimates so that they match the observed variances.
p	Print basic estimation results.

Number of Factors Options

n = arg (default = "map")	Number of factors: "kaiser" (Kaiser-Guttman greater than mean), "mineigen" (Minimum eigenvalue criterion; specified using "eiglimit"), "varfrac" (fraction of variance accounted for; specified using "varlimit"), "map" (Velicer's Minimum Average Partial method), "bstick" (comparison with broken stick distribution), "parallel" (parallel analysis: number of replications specified using "pnreps"; "pquant" indicates the quantile method value if employed), "scree" (standard error scree method), <i>integer</i> (user-specified integer value).
------------------------------	---

<code>eiglimit = number</code> (<i>default</i> = 1)	Limit value for retaining factors using the eigenvalue comparison (where “n = mineigen”).
<code>varlimit = number</code> (<i>default</i> = 0.5)	Fraction of total variance explained limit for retaining factors using the variance limit criterion (where “n = varlimit”).
<code>porig</code>	Use the unreduced matrix for parallel analysis (the default is to use the reduced matrix). For parallel analysis only (“n = parallel”).
<code>preps = integer</code> (<i>default</i> = 100)	Number of parallel analysis repetitions. For parallel analysis only (“n = parallel”).
<code>pquant = number</code>	Quantile value for parallel analysis comparison (if not specified, the mean value will be employed). For parallel analysis only (“n = parallel”).
<code>pseed = positive integer</code>	Seed the random number generator for parallel analysis. If not specified, EViews will seed the random number generator with a single integer draw from the default global random number generator. For parallel analysis only (“n = parallel”).
<code>prnd = arg</code> (<i>default</i> = “kn” or method previously set using rndseed (p. 770))	Type of random number generator for the simulation: improved Knuth generator (“kn”), improved Mersenne Twister (“mt”), Knuth’s (1997) lagged Fibonacci generator used in EViews 4 (“kn4”) L’Ecuyer’s (1999) combined multiple recursive generator (“le”), Matsumoto and Nishimura’s (1998) Mersenne Twister used in EViews 4 (“mt4”). For parallel analysis only (“n = parallel”).

Covariance Options

<code>cov = arg</code> (<i>default</i> = “cov”)	Covariance calculation method: ordinary (Pearson product moment) covariance (“cov”), ordinary correlation (“corr”), Spearman rank covariance (“rcov”), Spearman rank correlation (“rcorr”), Kendall’s tau-b (“taub”), Kendall’s tau-a (“taua”), uncentered ordinary covariance (“ucov”), uncentered ordinary correlation (“ucorr”). User-specified covariances are indicated by specifying a sym matrix object in place of a list of series or groups in the command.
<code>wgt = name</code> (optional)	Name of series containing weights.

<code>wgthmethod = arg</code> (default = "sstdev")	Weighting method (when weights are specified using "weight = "): frequency ("freq"), inverse of variances ("var"), inverse of standard deviation ("stdev"), scaled inverse of variances ("svar"), scaled inverse of standard deviations ("sstdev"). Only applicable for ordinary (Pearson) calculations. Weights specified by "wgt =" are frequency weights for rank correlation and Kendall's tau calculations.
<code>pairwise</code>	Compute using pairwise deletion of observations with missing cases (pairwise samples).
<code>df</code>	Compute covariances with a degree-of-freedom correction for the mean (for centered specifications), and any partial conditioning variables.

Examples

```
factor f1.pace(n=map, rescale) x y z
```

declares the factor object F1 and estimates the factors for the correlation matrix of X, Y, and Z, by the PACE method. The number of factors is determined by Velicer's MAP procedure and the result is rescaled to match the observed variances.

```
f1.pace(n=3) group01
```

estimates the three factor model for the series in GROUP01 by the PACE method.

```
f1.pace(n=3) group01 @partial ser1 ser2
```

estimates the same specification using the partial correlation for the series in GROUP01, conditional on the series SER1 and SER2.

```
f1.pace(n=scree) sym01 848
```

estimates the PACE factor model using the observed matrix SYM01. The number of observations is 848, and the number of factors is determined using the standard error scree.

Cross-references

See [Chapter 40. "Factor Analysis," on page 579](#) of the *User's Guide II* for a general discussion of factor analysis. The various estimation methods are described in ["Estimation Methods" on page 612](#) of the *User's Guide II*.

See also [Factor::gls \(p. 105\)](#), [Factor::ipf \(p. 109\)](#), [Factor::ml \(p. 117\)](#), [Factor::pf \(p. 125\)](#), [Factor::uls \(p. 138\)](#).

partcor[Factor Views](#)

Display the partial correlation matrix derived from the observed covariance matrix.

Syntax

```
factor_name.partcor(options)
```

The elements of the partial correlation matrix are the pairwise correlations conditional on the other variables.

The partial correlation matrix is computed by scaling the anti-image covariance to unit diagonal (or equivalently, by row and column scaling the inverse of the observed matrix by the square roots of its diagonals).

Options

p	Print the matrix.
---	-------------------

Examples

```
factor f1.ml group01
f1.partcor(p)
```

displays and prints the partial correlation matrix for the factor object F1.

Cross-references

See “[Observed Covariances](#)” on page 591 of the *User’s Guide II*. See also [Factor::anticov](#) (p. 100), [Factor::observed](#) (p. 121), and [Factor::smc](#) (p. 137).

pf[Factor Methods](#)

Principal factors estimation of the factor model.

Syntax

```
factor_name.pf(options) x1 [x2 x3...] [@partial z1 z2 z3...]
factor_name.pf(options) matrix_name [[obs] [conditioning]] [@ name1 name2
name3...]
```

The first method computes the observed dispersion matrix from a set of series or group objects. Simply append a period and the `pf` keyword to the name of your object, followed by the names of your series and groups. You may optionally use the keyword `@partial` and append a list of conditioning series.

In the second method you will provide the name of the observed dispersion matrix, and optionally, the number of observations and the rank of the set of conditioning variables. If the latter is not provided, it will be set to 1 (representing the constant in the standard centered variance calculations). You may also provide names for the columns of the correlation matrix by entering the @-sign followed by a list of valid series names.

Options

Estimation Options

p	Print basic estimation results.
---	---------------------------------

Number of Factors Options

n = <i>arg</i> (default = "map")	Number of factors: "kaiser" (Kaiser-Guttman greater than mean), "mineigen" (Minimum eigenvalue criterion; specified using "eiglimit"), "varfrac" (fraction of variance accounted for; specified using "varlimit"), "map" (Velicer's Minimum Average Partial method), "bstick" (comparison with broken stick distribution), "parallel" (parallel analysis: number of replications specified using "pnreps"; "pquant" indicates the quantile method value if employed), "scree" (standard error scree method), <i>integer</i> (user-specified integer value).
eiglimit = <i>number</i> (default = 1)	Limit value for retaining factors using the eigenvalue comparison (where "n = mineigen").
varlimit = <i>number</i> (default = 0.5)	Fraction of total variance explained limit for retaining factors using the variance limit criterion (where "n = varlimit").
porig	Use the unreduced matrix for parallel analysis (the default is to use the reduced matrix). For parallel analysis only ("n = parallel").
preps = <i>integer</i> (default = 100)	Number of parallel analysis repetitions. For parallel analysis only ("n = parallel").
pquant = <i>number</i>	Quantile value for parallel analysis comparison (if not specified, the mean value will be employed). For parallel analysis only ("n = parallel").
pseed = <i>positive integer</i>	Seed the random number generator for parallel analysis. If not specified, EViews will seed the random number generator with a single integer draw from the default global random number generator. For parallel analysis only ("n = parallel").

<code>prnd = arg</code> (default = “kn” or method previously set using rndseed (p. 770))	Type of random number generator for the simulation: improved Knuth generator (“kn”), improved Mersenne Twister (“mt”), Knuth’s (1997) lagged Fibonacci generator used in EViews 4 (“kn4”) L’Ecuyer’s (1999) combined mul- tiple recursive generator (“le”), Matsumoto and Nish- imura’s (1998) Mersenne Twister used in EViews 4 (“mt4”). For parallel analysis only (“n = parallel”).
--	---

Initial Communalities Options

<code>priors = arg</code>	Method for obtaining initial communalities: “smc” (squared multiple correlations), “max” (maximum abso- lute correlation), “pace” (noniterative partitioned covari- ance estimation), “frac” (fraction of the diagonals of the original matrix; specified using “priorfrac = ”), “random” (random fractions of the original diagonals), “user” (user- specified vector; specified using “priorunique”).
<code>priorfrac = number</code>	User-specified common fraction (between 0 and 1) to be used when “priors = frac”.
<code>priorunique = arg</code>	Vector of initial <i>uniqueness</i> estimates to be used when “priors = user”. By default, the values will be taken from the corresponding elements of the coefficient vector C.

Covariance Options

<code>cov = arg</code> (default = “cov”)	Covariance calculation method: ordinary (Pearson product moment) covariance (“cov”), ordinary correlation (“corr”), Spearman rank covariance (“rcov”), Spearman rank correla- tion (“rcorr”), Kendall’s tau-b (“taub”), Kendall’s tau-a (“taua”), uncentered ordinary covariance (“ucov”), uncen- tered ordinary correlation (“ucorr”). User-specified covariances are indicated by specifying a sym matrix object in place of a list of series or groups in the command.
<code>wgt = name</code> (optional)	Name of series containing weights.
<code>wgtmethod = arg</code> (default = “sst- dev”)	Weighting method (when weights are specified using “weight = ”): frequency (“freq”), inverse of variances (“var”), inverse of standard deviation (“stdev”), scaled inverse of variances (“svar”), scaled inverse of standard deviations (“sstdev”). Only applicable for ordinary (Pearson) calculations. Weights specified by “wgt = ” are frequency weights for rank correlation and Kendall’s tau calculations.

pairwise	Compute using pairwise deletion of observations with missing cases (pairwise samples).
df	Compute covariances with a degree-of-freedom correction for the mean (for centered specifications), and any partial conditioning variables.

Examples

```
factor f1.pf(n=map, priors=frac, priorfrac=1) x y z
```

declares the factor object F1 and extracts factors from the correlation matrix of the series X, Y, and Z, by the principal factor method. The original variances are used as the initial uniqueness estimates.

```
f1.pf(priors=pac) group01
```

extracts factors for the correlation of the series in GROUP01 by the principal factor method with initial uniqueness estimated by the PACE method.

```
f1.pf(priors=pac) group01 @partial ser1 ser2
```

estimates the same specification using the partial correlation for the series in GROUP01, conditional on the series SER1 and SER2.

Cross-references

See [Chapter 40. “Factor Analysis,” on page 579](#) of the *User’s Guide II* for a general discussion of factor analysis. The various estimation methods are described in [“Estimation Methods” on page 612](#) of the *User’s Guide II*.

See also [Factor::gls \(p. 105\)](#), [Factor::ipf \(p. 109\)](#), [Factor::ml \(p. 117\)](#), [Factor::pace \(p. 122\)](#), [Factor::uls \(p. 138\)](#).

reduced	Factor Views
---------	------------------------------

Display reduced covariance matrix for the estimated factor analysis object.

Syntax

```
factor_name.reduced(options)
```

By default, the reduced covariance is computed by subtracting the final uniqueness estimates from the observed covariance matrix. You may use the “initial” option to evaluate the reduced matrix using the initial uniqueness estimates.

Options

initial	Display the reduced matrix computed using the initial uniqueness estimates.
---------	---

p Print the matrix.

Examples

```
factor f1.pf x1 x2 x3 x4 x5 x6 x7 x8
f1.reduced
```

estimates a factor analysis model applied to the series X1 to X8 and displays the final reduced matrix (using final uniqueness estimates).

```
f1.reduced(initial)
```

displays the reduced matrix with the initial uniquenesses on the diagonal.

Cross-references

See [“Matrix Views” on page 591](#) of the *User’s Guide II*. See also `Factor::fitted` (p. 105).

resids	Factor Views
--------	------------------------------

Display residual covariance estimates for the factor analysis object.

Syntax

```
factor_name.resids(options)
```

By default, the residuals are computed by subtracting the estimate of the common variance and the final uniqueness estimates from the observed covariance matrix. You may use the “common” option to only subtract the common variance.

Options

common	Display the residuals computed using only the common fitted covariance.
p	Print the matrix.

Examples

```
factor f1.pfact x1 x2 x3 x4 x5 x6 x7 x8
f1.resids
```

estimates and displays the residuals for a factor analysis model applied to the series X1 to X8.

```
f1.resids(common)
```

displays the residuals computed without subtracting the uniqueness estimates.

Cross-references

See also [fit](#) (p. 720).

rotate	Factor Procs
--------	------------------------------

Perform an orthogonal or oblique factor rotation of the loadings of an estimated factor object.

Syntax

`factor_name.rotate(options)`

You may use the “type = ” and “method = ” options to select from a variety of rotations methods.

Method Options

The first five options control the basic rotation specification:

<code>type = arg</code> (<i>default</i> = “orthog”)	Orthogonal (“orthog”) or oblique (“oblique”) rotation (ignored if method is not supported, <i>e.g.</i> , “orthogonal Harris-Kaiser” or “oblique Entropy Ratio”).
<code>method = arg</code> (<i>default</i> = “ <i>varimax</i> ”)	Method (objective) for the rotation. See keywords below
<code>param = arg</code>	Rotation parameter, if applicable (see description below).
<code>preparam = arg</code> (<i>default</i> = 1, <i>Varimax</i>)	Orthomax pre-rotation parameter (for “method = <i>hk</i> ” and “method = <i>promax</i> ”).

The following rotation methods are supported:

Method	Keyword	Orthogonal	Oblique
Biquartimax	biquartimax	•	•
Crawford-Ferguson	cf	•	•
Entropy	entropy	•	
Entropy Ratio	entratio	•	
Equamax	equamax	•	•
Factor Parsimony	parsimony	•	•
Generalized Crawford-Ferguson	gcf	•	•
Geomin	geomin	•	•
Harris-Kaiser (case II)	hk		•

Infomax	infomax	•	•
Oblimax	oblimax		•
Oblimin	oblimin		•
Orthomax	orthomax	•	•
Parsimax	parsimax	•	•
Pattern Simplicity	pattern	•	•
Promax	promax		•
Quartimax/Quartimin	quartimax	•	•
Simplimax	simplimax	•	•
Tandem I	tandemi	•	
Tandem II	tandemii	•	
Target	target	•	•
Varimax	varimax	•	•

In selecting a rotation method you should bear in mind the following:

- EViews employs the Crawford-Ferguson variants of the Biquartimax, Equamax, Factor Parsimony, Orthomax, Parsimax, Quartimax, and Varimax objective functions. These objective functions yield the same results as the standard versions in the orthogonal case, but are better behaved (*e.g.*, do not permit factor collapse) under direct oblique rotation (see Browne 2001, p. 118-119). Note that oblique Crawford-Ferguson Quartimax is equivalent to Quartimin.
- The EViews Orthomax objective for parameter γ is evaluated using the Crawford-Ferguson objective with factor complexity weight $\kappa = \gamma/p$ (see “Types of Rotation,” on page 618 of the *User’s Guide II*).

Some special cases of Orthomax are Quartimax ($\gamma = 0$), Varimax ($\gamma = 1$), Equamax ($\gamma = m/2$), Parsimax ($\gamma = p(m-1)/(p+m-2)$) and Factor Parsimony ($\gamma = p$)

- The two orthoblique methods, Promax and Harris-Kaiser both perform an initial orthogonal rotation, followed by a oblique adjustment. For both of these methods, EViews provides some flexibility in the choice of initial rotation. By default, EViews will perform an initial orthogonal Orthomax rotation with the default parameter set to 1 (Varimax). To perform initial rotation with Quartimax, you should set the Orthomax parameter to 0.

Some of the rotation criteria have user-specified parameters that may be specified using the “param =” and (for Harris-Kaiser and Promax) the “preparam =” options. The parameters and their default values are given by:

Method	n	Parameter Description
Crawford-Ferguson	1	Factor complexity weight. The variable complexity weight is 1 minus the factor complexity weight. (<i>default</i> = 0, Quartimax)
Generalized Crawford-Ferguson	4	Vector of weights for (in order): total squares, variable complexity, factor complexity, diagonal quartics. (<i>no default</i>)
Geomin	1	Epsilon offset. (<i>default</i> = 0.01)
Harris-Kaiser (case II)	2	Power parameter (<i>default</i> = 0, independent cluster solution), Orthomax pre-rotation parameter. (<i>default</i> = 1, Varimax).
Oblimin	1	Deviation from orthogonality. (<i>default</i> = 0, Quartimin)
Orthomax	1	Factor complexity weight. (<i>default</i> = 1, Varimax)
Promax	2	Power parameter (<i>default</i> = 3), Orthomax pre-rotation parameter (<i>default</i> = 1, Varimax).
Simplimax	1	Fraction of near-zero loadings. (<i>default</i> = 0.75)
Target	1	Name of $p \times m$ matrix of target loadings. Missing values correspond to unrestricted elements. (<i>no default</i>)

where p is the number of variables and m is the number of factors. The remaining options modify the properties of the specified rotation method:

Options

`wgts = arg`
(*default* = “none”)

Row weighting for loadings: none (“none”), kaiser (“kaiser”), Cureton-Mulaik (“cureton”).

`prior = arg` (*default* = “unrotated”)
= “unrotated”)

Initial rotation matrix: unrotated (“unrotated”), randomly generated (“random”), previous rotation (“previous”), user-specified (“user”).

`ptype = arg`
(*default* = “orthog”)

Type of prior random rotation: orthogonal (“orthog”) or oblique (“oblique”).
Only relevant if “prior = random” and the main rotation method is oblique. If the main rotation method is orthogonal, random prior rotations will be orthogonalized.

<code>preps = integer</code> (<i>default</i> = 25)	Number of random prior rotations to evaluate (maximum 10000).
<code>pname = arg</code>	Name of matrix containing prior rotation.
<code>pseed = positive integer</code>	Seed the random number generator for the prior random rotations. If not specified, EViews will seed the random number generator with a single integer draw from the default global random number generator.
<code>prnd = arg</code> (<i>default</i> = “kn” or method previously set using rndseed (p. 770))	Type of random number generator for the random prior rotation: improved Knuth generator (“kn”), improved Mersenne Twister (“mt”), Knuth’s (1997) lagged Fibonacci generator used in EViews 4 (“kn4”) L’Ecuyer’s (1999) combined multiple recursive generator (“le”), Matsumoto and Nishimura’s (1998) Mersenne Twister used in EViews 4 (“mt4”).
<code>m = integer</code>	Maximum number of iterations.
<code>c = scalar</code>	Set convergence criterion. The criterion is based upon the norm of the gradients scaled by the objective function. The criterion will be set to the nearest value between 1e-24 and 0.2.
<code>showopts / -showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the rotation output.
<code>p</code>	Print rotation results.

Examples

```
f1.rotate(type=orthog, method=equamax)
```

performs an orthogonal rotation with the equamax objective function.

```
f1.rotate(type=oblique, method=hk, param=.4)
```

performs a Harris-Kaiser oblique rotation with parameter 0.4

```
f1.rotate(type=oblique, method=promax, param=.7)
```

performs a Promax rotation with parameter 0.7

Cross-references

See [“Rotating Factors” on page 586](#) of the *User’s Guide II* for a discussion of factor rotation. See also [Factor::rotateout](#) (p. 134) and [Factor::rotateclear](#) (p. 134).

rotateclear	Factor Views
-------------	------------------------------

Clear existing rotation.

Clears any existing factor rotations.

Syntax

factor_name.rotateclear

Examples

fact1.rotateclear

Cross-references

See [“Rotating Factors” on page 586](#) of the *User’s Guide II* for a discussion of factor rotation.
See also [Factor::rotate \(p. 130\)](#) and [Factor::rotateout \(p. 134\)](#).

rotateout	Factor Views
-----------	------------------------------

Display rotated factors and other results of factor rotation estimation.

Syntax

factor_name.rotateout

Options

p	Print the table of results.
---	-----------------------------

Examples

fl.rotate
fl.output
fl.rotateout(p)

performs factor rotation, switches to the main estimation output view, then displays and prints the rotation results.

Cross-references

See [“Rotating Factors” on page 586](#) of the *User’s Guide II* for a discussion of factor rotation.
See also [Factor::rotate \(p. 130\)](#) and [Factor::rotateclear \(p. 134\)](#).

scores	Factor Views
--------	--------------

Compute factor score coefficients and scores and display results in table, sheet, or graph form.

Syntax

There are two forms of the `scores` command. The first form of the command, which applies when displaying table results or spreadsheet displays of scores is given by:

```
factor_name.scores(options) [observed_list]
```

The optional *observed_list* of observed input variables will be multiplied by the score coefficients to compute the scores.

The second form of the command applies when plotting scores. In this case, the syntax is:

```
factor_name.scores(options) [graph_list] [@ observed_list]
```

where the *[graph_list]* is an optional list of integers and/or vectors containing integers identifying the factors to plot. If *graph_list* is not provided, EViews will construct graphs using all of the retained factors.

Multiple pairs are handled using the method specified in the “mult = ” option. Note that the order of elements in the list matters; reversing the order of two indices reverses the axis on which each factor is displayed.

You should also bear in mind that:

- Specification of the *observed_list* is required only for actually computing score values—it is not required for computing score coefficient summaries and diagnostics (“out = table”).
- If *observed_list* is not provided, EViews will use the observed variables from the factor estimation specification. For factor models specified using a symmetric matrix, you must provide a *observed_list* if you wish to obtain score values.
- Scores values will be computed for observations in the current workfile sample that do not have missing values for the observed inputs.

Options

out = *arg*
(default = “table”)

Output format: coefficient summary and diagnostics (“table”), spreadsheet table of scores (“sheet”), graph of scores (“graph”), graph of scores with loadings axes (“biplot”).

unrotated

Use unrotated loadings in computations (the default is to use the rotated loadings, if available).

<code>type = arg</code> (<i>default</i> = "exact")	Exact coefficient ("exact"), coarse adjusted factor coefficients ("coefs"), coarse adjusted factor loadings ("loadings").
<code>coef = arg</code> (<i>default</i> = "reg")	Method for computing the factor score coefficient matrix: Thurstone regression ("reg"), Ideal Variables ("ideal"), Bartlett weighted least squares ("wls"), generalized Anderson-Rubin-McDonald ("anderson"), Green ("green"). For "type = exact" and "type = coefs" specifications.
<code>cutoff = number</code> (<i>default</i> = 0.3)	Cutoff value for coarse score coefficient calculation (Grice, 1991a). For "type = coefs" specifications, the cutoff value represents the fraction of the largest absolute coefficient weight per factor against which the absolute exact score coefficients should be compared. For "type = loadings" specifications, the cutoff is the value against which the absolute loadings or structure coefficients should be compared.
<code>moment = arg</code> (<i>default</i> = "est"; if feasible)	Standardize the observables data using means and variances from: original estimation ("est"), the computed moments from specified observable variables ("obs"). The "moment = est" option is only available for factor models estimated using Pearson or uncentered Pearson correlation and covariances since the remaining models involve unobserved or non-comparable moments.
<code>df</code>	Degrees-of-freedom correct the observables variances computed when "moment = obs" (divide sums-of-squares by $n - 1$ instead of n).
<code>coefout</code>	(Optional) Name of matrix in which to save factor score coefficient matrix.
<code>p</code>	Print results.

Graph Options

<code>mult = arg</code> (<i>default</i> = "first")	Multiple series handling for graphs: plot first against remainder ("first"), plot as x-y pairs ("pair"), lower-triangular plot ("lt")
<code>nocenter</code>	Do not center graphs around the origin.
<code>labels = arg</code> , (<i>default</i> = "outlier")	Observation labels for scores: outliers only ("outlier"), all points ("all"), none ("none").

<code>labelprob = number</code>	Probability value for determining whether a point is an outlier according to the chi-square tests based on the squared Mahalanbois distance between the observation and the sample means (when using the “labels = outlier” option).
<code>userscale = arg</code>	User-scale factor to be applied to the unscaled loadings (setting this option overrides the automatic scaling).
<code>autoscale = arg</code> <i>(default = 1)</i>	User-scale factor to be applied to the automatic loadings scale (when displaying both loadings and scores).

Examples

```
f1.scores(out=table)
```

computes factor score coefficients and displays a table of coefficient summaries and diagnostics.

```
f1.scores(coef=anderson, out=biplot, mult=first) 1 3 4
```

displays a biplot graph of the factor scores. The graph plots the first factor against the third, and the first factor against the fourth. The scores are computed using the observed variables from the original factor estimation specification and generalized Anderson-Rubin-McDonald factor score coefficients.

Cross-references

See “Estimating Scores,” beginning on page 587 and “Scoring,” on page 620 of the *User’s Guide II*. See also `Factor::makescores` (p. 114).

smc	Factor Views
-----	------------------------------

Display the squared multiple correlations for the observed covariance matrix.

Syntax

```
factor_name.smc(options)
```

The SMCS are equal to 1 minus the diagonal elements of the anti-image covariance.

Options

<code>p</code>	Print the matrix.
----------------	-------------------

Examples

```
factor f1.ml group01
f1.smc(p)
```

displays and prints the squared multiple correlations for the observed matrix attached to F1.

Cross-references

See also [Factor::observed](#) (p. 121), [Factor::anticov](#) (p. 100), and [Factor::maxcor](#) (p. 116).

structure	Factor Views
-----------	------------------------------

Display the factor structure matrix.

Shows the factor structure matrix containing the correlations between the variables and factors implied by an estimated factor model. For orthogonal factors, the structure matrix is equal to the loadings matrix.

Syntax

`factor_name.structure(options)`

Options

p	Print the matrix.
---	-------------------

Examples

```
factor f1.ml group01
f1.structure(p)
```

displays and prints the factor structure matrix for the estimated factor object F1.

Cross-references

See “[Factor Structure Matrix](#)” on page 592 of the *User’s Guide II* for details. See [Factor::rotate](#) (p. 130) and [Factor::loadings](#) (p. 113).

uls	Factor Methods
-----	--------------------------------

Unweighted least squares estimation of the factor model.

Syntax

```
factor_name.uls(options) x1 [x2 x3...] [@partial z1 z2 z3...]
factor_name.uls(options) matrix_name [[obs] [conditioning]] [@ name1 name2
name3...]
```

The first method computes the observed dispersion matrix from a set of series or group objects. Simply append a period and the `uls` keyword to the name of your object, followed

by the names of your series and groups. You may optionally use the keyword `@partial` and append a list of conditioning series.

In the second method you will provide the name of the observed dispersion matrix, and optionally, the number of observations and the rank of the set of conditioning variables. If the latter is not provided, it will be set to 1 (representing the constant in the standard centered variance calculations). You may also provide names for the columns of the correlation matrix by entering the `@`-sign followed by a list of valid series names.

Options

Estimation Options

<code>rescale</code>	Rescale the uniqueness and loadings estimates so that they match the observed variances.
<code>maxit = integer</code>	Maximum number of iterations.
<code>conv = scalar</code>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled estimates. The criterion will be set to the nearest value between 1e-24 and 0.2.
<code>showopts / -showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the rotation output.
<code>p</code>	Print basic estimation results.

Number of Factors Options

<code>n = arg</code> (<i>default</i> = “map”)	Number of factors: “kaiser” (Kaiser-Guttman greater than mean), “mineigen” (Minimum eigenvalue criterion; specified using “eiglimit”), “varfrac” (fraction of variance accounted for; specified using “varlimit”), “map” (Velicer’s Minimum Average Partial method), “bstick” (comparison with broken stick distribution), “parallel” (parallel analysis: number of replications specified using “pnreps”; “pquant” indicates the quantile method value if employed), “scree” (standard error scree method), <i>integer</i> (user-specified integer value).
<code>eiglimit = number</code> (<i>default</i> = 1)	Limit value for retaining factors using the eigenvalue comparison (where “n = mineigen”).
<code>varlimit = number</code> (<i>default</i> = 0.5)	Fraction of total variance explained limit for retaining factors using the variance limit criterion (where “n = varlimit”).
<code>porig</code>	Use the unreduced matrix for parallel analysis (the default is to use the reduced matrix). For parallel analysis only (“n = parallel”).

<code>preps = integer</code> (<i>default</i> = 100)	Number of parallel analysis repetitions. For parallel analysis only (“n = parallel”).
<code>pquant = number</code>	Quantile value for parallel analysis comparison (if not specified, the mean value will be employed). For parallel analysis only (“n = parallel”).
<code>pseed = positive integer</code>	Seed the random number generator for parallel analysis. If not specified, EViews will seed the random number generator with a single integer draw from the default global random number generator. For parallel analysis only (“n = parallel”).
<code>prnd = arg</code> (<i>default</i> = “kn” or method previously set using rndseed (p. 770))	Type of random number generator for the simulation: improved Knuth generator (“kn”), improved Mersenne Twister (“mt”), Knuth’s (1997) lagged Fibonacci generator used in EViews 4 (“kn4”) L’Ecuyer’s (1999) combined multiple recursive generator (“le”), Matsumoto and Nishimura’s (1998) Mersenne Twister used in EViews 4 (“mt4”). For parallel analysis only (“n = parallel”).

Initial Communalities Options

<code>priors = arg</code>	Method for obtaining initial communalities: “smc” (squared multiple correlations), “max” (maximum absolute correlation), “pace” (noniterative partitioned covariance estimation), “frac” (fraction of the diagonals of the original matrix; specified using “priorfrac = ”), “random” (random fractions of the original diagonals), “user” (user-specified vector; specified using “priorunique”).
<code>priorfrac = number</code>	User-specified common fraction (between 0 and 1) to be used when “priors = frac”.
<code>priorunique = arg</code>	Vector of initial <i>uniqueness</i> estimates to be used when “priors = user”. By default, the values will be taken from the corresponding elements of the coefficient vector C.

Covariance Options

<code>cov = arg</code> (<i>default</i> = “cov”)	Covariance calculation method: ordinary (Pearson product moment) covariance (“cov”), ordinary correlation (“corr”), Spearman rank covariance (“rcov”), Spearman rank correlation (“rcorr”), Kendall’s tau-b (“taub”), Kendall’s tau-a (“taua”), uncentered ordinary covariance (“ucov”), uncentered ordinary correlation (“ucorr”). User-specified covariances are indicated by specifying a sym matrix object in place of a list of series or groups in the command.
<code>wgt = name</code> (optional)	Name of series containing weights.
<code>wgtmethod = arg</code> (<i>default</i> = “sstdev”)	Weighting method (when weights are specified using “weight = ”): frequency (“freq”), inverse of variances (“var”), inverse of standard deviation (“stdev”), scaled inverse of variances (“svar”), scaled inverse of standard deviations (“sstdev”). Only applicable for ordinary (Pearson) calculations. Weights specified by “wgt = ” are frequency weights for rank correlation and Kendall’s tau calculations.
<code>pairwise</code>	Compute using pairwise deletion of observations with missing cases (pairwise samples).
<code>df</code>	Compute covariances with a degree-of-freedom correction for the mean (for centered specifications), and any partial conditioning variables.

Examples

```
factor f1.uls(n=map, priors=frac, priorfrac=1) x y z
```

declares the factor object F1 and estimates the factors for the correlation matrix of the series X, Y, and Z, by the unweighted least squares method.

```
f1.uls(maxit=300, conv=1e-8) group01
```

estimates the factors by the unweighted least squares method for the series in GROUP01 with maximum iterations 300 and convergence criterion 1e-8.

```
f1.uls(maxit=300, conv=1e-8) group01 @partial ser1 ser2
```

estimates the same specification using the partial correlation for the series in GROUP01, conditional on the series SER1 and SER2.

```
f1.uls(n=4) sym01 747
```

estimates the four factor ULS factor model using the observed matrix SYM01. The number of observations is 747.

Cross-references

See [Chapter 40. “Factor Analysis,” on page 579](#) of the *User’s Guide II* for a general discussion of factor analysis. The various estimation methods are described in [“Estimation Methods” on page 612](#) of the *User’s Guide II*.

See also [Factor::gls \(p. 105\)](#), [Factor::ipf \(p. 109\)](#), [Factor::ml \(p. 117\)](#), [Factor::pace \(p. 122\)](#), [Factor::pf \(p. 125\)](#), [Factor::uls \(p. 138\)](#).

Graph

Graph object. Specialized object used to hold graphical output.

Graph Declaration

freeze freeze graphical view of object (p. 724).
graph create graph object using graph command or by merging existing graphs (p. 158).

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
group grp2 ser1 ser2
freeze(myscat) grp2.scat
freeze(myxy) grp1.xyline
```

which are equivalent to the declarations above.

Graph Type Commands

Graph creation types are discussed in detail in “Graph Creation Commands” on page 601.

area area graph (p. 603).
band area band graph (p. 606).
bar bar graph (p. 609).
boxplot boxplot graph (p. 613).
distplot distribution graph (p. 615).
dot dot plot graph (p. 622).
errbar error bar graph (p. 626).
hilo high-low (-open-close) graph (p. 628).
line line-symbol graph (p. 630).
pie pie chart (p. 633).
qqplot quantile-quantile graph (p. 636).
scat scatterplot (p. 640).
scatmat matrix of scatterplots (p. 644).
scatpair scatterplot pairs graph (p. 647).
seasplot seasonal line graph (p. 651).

spike spike graph (p. 652).
xyarea XY area graph (p. 656).
xybar XY bar graph (p. 659).
xyline XY line graph (p. 661).
xypair XY pairs graph (p. 664).

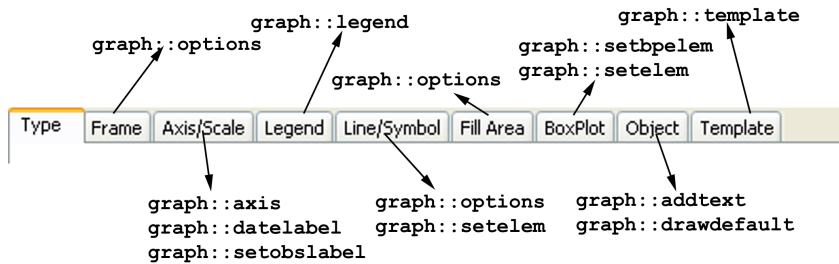
Graph View

label label information for the graph (p. 160).

Graph Procs

addtext place arbitrary text on the graph (p. 145).
align align the placement of multiple graphs (p. 148).
axis set the axis scaling and display characteristics for the graph (p. 149).
datelabel controls labeling of the bottom date/time axis in time plots (p. 152).
displayname set display name (p. 154).
draw draw lines and shaded areas on the graph (p. 154).
drawdefault set default settings for lines and shaded areas on the graph (p. 156).
legend control the appearance and placement of legends (p. 161).
merge merge graph objects (p. 163).
name change the series name for legends or axis labels (p. 163).
options change the option settings of the graph (p. 164).
save save graph to a graphics file (p. 168).
setbpelem set options for element of a boxplot graph (p. 170).
setelem set individual line, symbol, bar and legend options for each series in the graph (p. 171).
setobslabel set custom axis labels (p. 175).
sort sort the series in a graph (p. 177).
template use template graph (p. 178).
textdefault set default settings for text objects in the graph (p. 179).

The relationship between the tabs of the graph dialog and the associated graph procs is illustrated below:



Graph Examples

You can declare your graph:

```
graph abc.xyline(m) unemp gnp inf
graph bargraph.bar(d,l) unemp gnp
```

Alternately, you may freeze any graphical view:

```
freeze(mykernel) ser1.distplot kernel
```

You can change the graph type,

```
graph mygraph.line ser1
mygraph.hist
```

or combine multiple graphs:

```
graph xyz.merge graph1 graph2
```

Graph Entries

The following section provides an alphabetical listing of the commands associated with the “Graph” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

addtext	Graph Procs
---------	-----------------------------

Place text in graphs.

When adding text in one of the four predefined positions (left, right, top, bottom), EViews 6.0 now deletes any existing text that is in that position before adding the new text. Use the **keep** option to preserve the existing text.

Syntax

```
graph_name.addtext(options) "text"
```

Follow the `addtext` keyword with the *text* to be placed in the graph, enclosed in double quotes.

To include carriage returns in your text, use the control “\r” or “\n” to represent the return. Since the backslash “\” is a special character in the `addtext` command, use a double slash “\\” to include the literal backslash character.

Options

The following options may be provided to change the characteristics of the specified text object. *Any unspecified options will use the default text settings of the graph.*

<code>font([face], [pt], [+/- b], [+/- i], [+/- u], [+/- s])</code>	Set characteristics of text font. The font name (<i>face</i>), size (<i>pt</i>), and characteristics are all optional. <i>face</i> should be a valid font name, enclosed in double quotes. <i>pt</i> should be the font size in points. The remaining options specify whether to turn on/off boldface (b), italic (i), underline (u), and strikeout (s) styles.
<code>textcolor(arg)</code>	Sets the color of the text. <i>arg</i> may be one of the predefined color keywords, or it may be made up of <i>n1</i> , <i>n2</i> , <i>n3</i> , a set of three integers from 0 to 255, representing the RGB values of the color. For a description of the available color keywords (“blue”, “red”, “green”, “black”, “white”, “purple”, “orange”, “yellow”, “gray”, “ltgray”), see Table::setfillcolor (p. 516).
<code>fillcolor(arg)</code>	Sets the background fill color of the text box. <i>arg</i> may be one of the predefined color keywords, or it may be made up of <i>n1</i> , <i>n2</i> , <i>n3</i> , a set of three integers from 0 to 255, representing the RGB values of the color. For a description of the available color keywords (“blue”, “red”, “green”, “black”, “white”, “purple”, “orange”, “yellow”, “gray”, “ltgray”), see Table::setfillcolor (p. 516).
<code>framecolor(arg)</code>	Sets the color of the text box frame. <i>arg</i> may be one of the predefined color keywords, or it may be made up of <i>n1</i> , <i>n2</i> , <i>n3</i> , a set of three integers from 0 to 255, representing the RGB values of the color. For a description of the available color keywords (“blue”, “red”, “green”, “black”, “white”, “purple”, “orange”, “yellow”, “gray”, “ltgray”), see Table::setfillcolor (p. 516).
<code>keep</code>	When adding text to one of the predefined positions (left, right, top, bottom), any existing text in that position will be deleted and replaced with the new text. Use the keep option to preserve the existing text and place the second text object on top of the text in that position.

The following options control the position of the text:

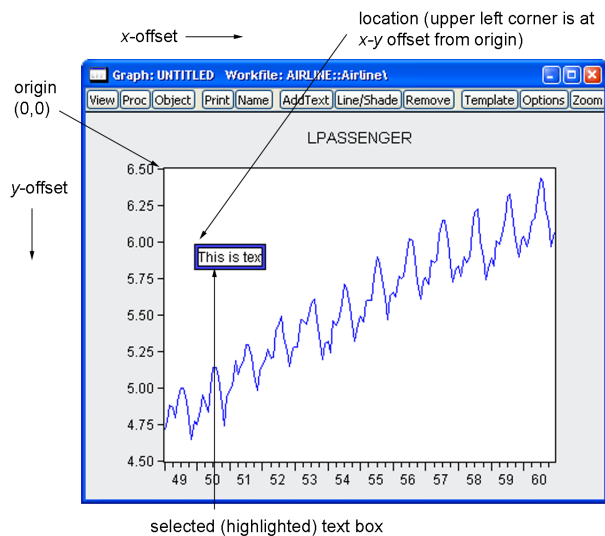
t	Top (above and centered over the graph).
l	Left rotated.
r	Right rotated.
b	Below and centered over the graph.
just(<i>arg</i>)	Set the justification of the text, where <i>arg</i> may be: “c” (center), “l” (left - default), “r” (right).
x	Enclose text in box.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

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 4×3 virtual inches (scatter diagrams are 3×3 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
```

```
g1.addtext(t) "Fig 1: Monthly GDP (78m1-95m12) "
```

places the text “Fig1: Monthly GDP (78m1-95m12)” centered above the graph G1.

```
g1.addtext(.2, .2, X) "Seasonally Adjusted"
```

places the text “Seasonally Adjusted” in a box within the graph, slightly indented from the upper left corner.

```
g1.addtext(t, x, textcolor(red), fillcolor(128,128,128), frame-  
color(black)) "Civilian\rUnemployment (First\\Last) "
```

adds the text “Civilian Unemployment (First\Last)” where there is a return between the “Civilian” and “Unemployment”. The text is colored red, and is enclosed in a gray box with a black frame.

Cross-references

See [Graph::legend](#) (p. 161) and [Graph::textdefault](#) (p. 179).

align	Graph Procs
--------------	-----------------------------

Align placement of multiple graphs.

Syntax

```
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

```
mygraph.align(3,1.5,1)
```

aligns MYGRAPH with graphs placed in three columns, horizontal spacing of 1.5 virtual inches, and vertical spacing of 1 virtual inch.

```
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 13. “Graphing Data,” beginning on page 415](#) of the *User’s Guide I*.

axis	Graph Procs
------	-----------------------------

Sets axis scaling and display characteristics for the graph.

By default, EViews optimally chooses the axis scaling to fit the graph data.

Syntax

```
graph_name.axis(axis_id) options_list
```

The *axis_id* parameter identifies which of the axes the command 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. 640) , xyarea (p. 656) , xybar (p. 659) , xyline (p. 661) , xypair (p. 664)).
top / t	Top axis for XY and scatter graphs (scat (p. 640) , xyarea (p. 656) , xybar (p. 659) , xyline (p. 661) , xypair (p. 664)).
all / a	All axes.

Options

The options list may include any of the following options:

Data scaling options

linear	Linear data scaling (<i>default</i>).
linearzero	Linear data scaling (include zero when auto range selection is employed).
log	Logarithmic scaling.
norm	Norm (standardize) the data prior to plotting.
range(<i>arg</i>)	Specifies the endpoints for the scale, where <i>arg</i> may be: “auto” (automatic choice), “minmax” (use the maximum and minimum values of the data), “ <i>n1</i> , <i>n2</i> ” (set minimum to <i>n1</i> and maximum to <i>n2</i> , <i>e.g.</i> “range(3, 9)”).

overlap / -overlap	[Overlap / Do not overlap] scales on dual scale graphs.
invert / -invert	[Invert / do not invert] scale.
units(<i>arg</i>)	Specifies the units of the data, where <i>arg</i> may be: “n” (native), “p” (percent), “k” (thousands), “m” (millions), “b” (billions), “t” (trillions).
format(<i>option1</i> [, <i>option2</i> , ...])	Sets data formatting, where you may provide one or more of the following options: “commadec” / “-commadec” ([Do / Do not] use comma as decimal, “ksep” / “-ksep” ([Do / Do not] include a thousands separator, “leadzero” / “-leadzero” ([Do / Do not] include leading zeros, “dec = <i>arg</i> ” (set number of decimal places, where <i>arg</i> may be an integer or “a” for auto), “prefix = <i>c</i> ” (add a prefix character, where <i>c</i> may be a single quoted character or “” to remove the prefix), “suffix = <i>c</i> ” (add a suffix character, where <i>c</i> may be a single quoted character or “” to remove the suffix).

Axis options

grid / -grid	[Draw / Do not draw] grid lines.
zeroline / -zeroline	[Draw / Do not 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.
ticksbtw	Draw tickmarks between observations.
tickson	Draw tickmarks on observations.
minor / -minor	[Allow / Do not allow] minor tick marks.
label / -label	[Place / Do not place] labels on the axes.
font([<i>face</i>], [<i>pt</i>], [<i>+/- b</i>], [<i>+/- i</i>], [<i>+/- u</i>], [<i>+/- s</i>])	Set characteristics of font. The font name (<i>face</i>), size (<i>pt</i>), and characteristics are all optional. <i>face</i> should be a valid font name, enclosed in double quotes. <i>pt</i> should be the font size in points. The remaining options specify whether to turn on/off boldface (<i>b</i>), italic (<i>i</i>), underline (<i>u</i>), and strikeout (<i>s</i>) styles.

<code>textcolor(arg)</code>	Sets the background color of the legend text. <i>arg</i> may be one of the predefined color keywords, or it may be made up of <i>n1</i> , <i>n2</i> , <i>n3</i> , a set of three integers from 0 to 255, representing the RGB values of the color. For a description of the available color keywords (“blue”, “red”, “green”, “black”, “white”, “purple”, “orange”, “yellow”, “gray”, “ltgray”), see Table::setfillcolor (p. 516).
<code>mirror / -mirror</code>	[Label / Do not label] both left and right axes with duplicate axes (single scale graphs only).
<code>angle(arg)</code>	Set label angle, where <i>arg</i> can be an integer between -90 and 90 degrees, measured in 15 degree increments, or “a” (auto) for automatically determined angling. The angle is measured from the horizontal axis.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

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:

```
graph1.axis(right) log range(10, 30)
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.

```
mygral.axis font("Times", 12, b, i) textcolor(blue)
```

sets the axis font to blue “Times” 12pt bold italic.

```
gral.axis(l) units(b) format(ksep, prefix="$", suffix="")
```

plots the data on the left axis in billions, using commas to separate thousands, adds a “\$” to the beginning of each data label and erases the suffix.

Cross-references

See [Chapter 15. “Graphs, Tables, Text, and Spools,”](#) on page 523 of the *User’s Guide I* for a discussion of graph options.

See also [Graph::datelabel](#) (p. 152), [Graph::options](#) (p. 164) and [Graph::setelem](#) (p. 171).

<code>bplabel</code>	Graph Procs
----------------------	-----------------------------

Specify labeling of a boxplot axis.

Note that `bplabel` is no longer supported. See instead, [Graph::setobslabel \(p. 175\)](#).

<code>datelabel</code>	Graph Procs
------------------------	-----------------------------

Control labeling of the bottom date/time axis in time plots.

`datelabel` sets options that are specific to the appearance of time/date labeling. Many of the options that also affect the appearance of the date axis are set by the [Graph::axis \(p. 149\)](#) command with the “bottom” option. These options include tick control, label and font options, and grid lines.

Syntax

`graph_name.datelabel option_list`

Options

<code>format("datestring")</code>	<i>datestring</i> should be one of the supported data formats describing how the date should appear. The <i>datestring</i> argument should be enclosed in double-quotes. For example, “yy:mm” specifies two-digit years followed by a colon delimited and then two-digit months. EViews provides considerable flexibility in formatting your dates. See “Date Formats” on page 707 of the <i>User’s Guide I</i> for a complete description.
<code>interval(step_size [,steps][,align_date])</code>	where <i>step_size</i> takes one of the following values: “auto” (<i>steps</i> and <i>align_date</i> are ignored), “ends” (only label endpoints; <i>steps</i> and <i>align_date</i> are ignored), “all” (label every point; the <i>steps</i> and <i>align_date</i> options are ignored), “obs” (steps are one observation), “year” (steps are one year), “m” (steps are one month), “q” (steps are one quarter). <i>steps</i> is a number (<i>default</i> = 1) indicating the number of steps between labels. <i>align_date</i> is a date specified to receive a label. Note, the <i>align_date</i> should be in the units of the data being graphed, but may lie outside the current sample or workfile range.
<code>span / -span</code>	[Allow/Do not allow] date labels to span an interval. Consider the case of a yearly label with monthly ticks. If <i>span</i> is on, the label is centered on the 12 monthly ticks. If the <i>span</i> option is off, year labels are put on the first quarter or month of the year.

Examples

```
graph1.datelabel format(yyyy:mm)
```

will display dates using four-digit years followed by the default delimiter “:” and a two-digit month (e.g. – “1974:04”).

```
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. – “74q04”)

```
graph1.datelabel interval(y, 2, 1951)
```

specifies labels every two years on odd numbered years.

Cross-references

See [Chapter 15. “Graphs, Tables, Text, and Spools,” on page 523](#) of the *User’s Guide I* for a discussion of graph options.

See also [Graph::axis](#) (p. 149), [Graph::options](#) (p. 164), and [Graph::setelem](#) (p. 171).

dates	Graph Procs
-------	-----------------------------

See the replacement command [Graph::datelabel](#) (p. 152).

displayname	Graph Procs
-------------	-----------------------------

Display name for a graph object.

Attaches a display name to a graph object which may be used to label output in place of the standard graph object name.

Syntax

`graph_name.displayname display_name`

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in graph object names.

Examples

```
gr1.displayname Hours Worked
gr1.label
```

The first line attaches a display name “Hours Worked” to the graph GR1, and the second line displays the label view of GR1, including its display name.

Cross-references

See “[Labeling Objects](#)” on page 72 of the *User’s Guide I* for a discussion of labels and display names.

See also [Graph::label](#) (p. 160) and [Graph::legend](#) (p. 161).

draw	Graph Procs
------	-----------------------------

Place horizontal or vertical lines and shaded areas on the graph.

Syntax

`graph_name.draw(draw_type, axis_id [options]) position1 [position2]`

where *draw_type* may be one of the following:

line / l	A line
shade	A shaded area

Note that the “dashline” option has been removed (though it is supported for backward compatibility). You should use the “pattern” option to specify whether the line is solid or patterned.

axis_id may take the values:

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

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 boundaries of the shaded region.

Line/Shade Options

The following options may be provided to change the characteristics of the specified line or shade. *Any unspecified options will use the default text settings of the graph.*

color(<i>arg</i>)	Specifies the color of the line or shade. The argument may be made up of <i>n1</i> , <i>n2</i> , and <i>n3</i> , a set of three integers from 0 to 255, representing the RGB values of the line or shade, or it may be one of the predefined color keywords (“blue”, “red”, “green”, “black”, “white”, “purple”, “orange”, “yellow”, “gray”, “ltgray”). For a full description of the keywords, see Table::setfillcolor (p. 516). The default is black for lines and gray for shades. RGB values may be examined by calling up the color palette in the Graph Options dialog.
pattern(<i>index</i>)	Sets the line pattern to the type specified by <i>index</i> . <i>index</i> can be an integer from 1 to 12 or one of the matching keywords (“solid”, “dash1” through “dash10”, “none”). See Graph::setelem (p. 171) for a description of the available patterns. The “none” keyword turns on solid lines.
width(<i>n1</i>)	Specify the width, where <i>n1</i> is the line width in points (used only if <i>object_type</i> is “line” or “dashline”). The default is 0.5 points.

Examples

```
graph1.draw(line, left, rgb(0,0,127)) 5.25
```

draws a horizontal blue line at the value “5.25” as measured on the left axis while:

```
graph1.draw(shade, right) 7.1 9.7
```

draws a shaded horizontal region bounded by the right axis values “7.1” and “9.7”. You may also draw vertical regions by using the “bottom” *axis_id*:

```
graph1.draw(shade, bottom) 1980:1 1990:2
```

draws a shaded vertical region bounded by the dates “1980:1” and “1990:2”.

```
graph1.draw(line, bottom, pattern(dash1)) 1985:1
```

draws a vertical dashed line at “1985:1”.

Cross-references

See [Chapter 15. “Graphs, Tables, Text, and Spools,” on page 523](#) of the *User’s Guide I* for a discussion of graph options.

See [Graph::drawdefault \(p. 156\)](#) for setting defaults.

drawdefault	Graph Procs
-------------	-----------------------------

Change default settings for lines and shaded areas in the graph.

This command specifies changes in the default settings which will be applied to line and shade objects added subsequently to the graph. If you include the “existing” option, *all* of the drawing default settings will also be applied to existing line and shade objects in the graph.

Syntax

```
graph_name.drawdefault draw_options
```

where *draw_options* may include one or more of the following:

<code>linecolor(<i>arg</i>)</code>	Sets the default color for lines. The <i>arg</i> 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 (“blue”, “red”, “green”, “black”, “white”, “purple”, “orange”, “yellow”, “gray”, “ltgray”). For a full description of the keywords, see Table::setfillcolor (p. 516).
<code>shadecolor(<i>arg</i>)</code>	Sets the default color for shades. <i>arg</i> may be one of the predefined color keywords, or it may be made up of <i>n1</i> , <i>n2</i> , <i>n3</i> , a set of three integers from 0 to 255, representing the RGB values of the color. For a description of the available color keywords (“blue”, “red”, “green”, “black”, “white”, “purple”, “orange”, “yellow”, “gray”, “ltgray”), see Table::setfillcolor (p. 516).
<code>width(<i>n1</i>)</code>	Specify the width, where <i>n1</i> is the line width in points (used only if <code>object_type</code> is “line” or “dashline”). The default is 0.5 points.
<code>pattern(<i>index</i>)</code>	Sets the default line pattern to the type specified by <i>index</i> . <i>index</i> can be an integer from 1 to 12 or one of the matching keywords (“solid”, “dash1” through “dash10”, “none”). See Graph::setelem (p. 171) for a description of the available patterns. The “none” keyword turns on solid lines.
<code>existing</code>	Apply the default settings to all existing line/shade objects in the graph.

Examples

```
graph1.drawdefault linecolor(blue) width(.25) existing
```

changes the default setting for new line/shade objects. New lines added to the graph will now be drawn in blue, with a width of 0.25 points. In addition, all existing line and shade objects will be updated with the graph default settings. Note that in addition to the line color and width settings specified in the command, the existing default line pattern and shade colors will be applied to the line and shade objects in graph.

```
graph1.drawdefault existing
```

updates all line and shade objects in the graph with the currently specified default draw object settings.

Cross-references

See [Chapter 15. “Graphs, Tables, Text, and Spools,”](#) on page 523 of the *User’s Guide I* for a discussion of graph options.

See `Graph::draw` (p. 154).

graph	Graph Declaration
--------------	-----------------------------------

Create named graph object containing the results of a graph command, or created when merging multiple graphs into a single graph.

Syntax

```
graph graph_name.graph_command(options) arg1 [arg2 arg3 ...]
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 forms 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,

```
graph_command(options) arg1 [arg2 arg3 ...]
```

should follow the form of one of the standard EViews graph commands:

area	Area graph (area (p. 603)).
band	Area band graph (band (p. 606)).
bar	Bar graph (bar (p. 609)).
boxplot	Boxplot graph (boxplot (p. 613)).
distplot	Distribution graph (distplot (p. 615)).
dot	Dot plot graph (dot (p. 622)).
errbar	Error bar graph (errbar (p. 626)).
hilo	High-low(-open-close) graph (hilo (p. 628)).
line	Line graph (line (p. 630)).
pie	Pie graph (pie (p. 633)).
qqplot	Quantile-Quantile graph (qqplot (p. 636)).
scat	Scatterplot—same as XY, but lines are initially turned off, symbols turned on, and a 3 × 3 frame is used (scat (p. 640)).
scatmat	Matrix of scatterplots (scatmat (p. 644)).
scatpair	Scatterplot pairs graph (scatpair (p. 647)).
seasplot	Seasonal line graph (seasplot (p. 651)).
spike	Spike graph (spike (p. 652)).

xyarea	XY line-symbol graph with one X plotted against one or more Y's using existing line-symbol settings (xyarea (p. 656)).
xybar	XY line-symbol graph with one X plotted against one or more Y's using existing line-symbol settings (xybar (p. 659)).
xyline	Same as XY, but symbols are initially turned off, lines turned on, and a 4×3 frame is used (xyline (p. 661)).
xypair	Same as XY but sets XY settings to display pairs of X and Y plotted against each other (xypair (p. 664)).

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 name.

Options

reset	Resets all graph options to the global defaults. May be used to remove existing customization of the graph.
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 (for example, see [bar \(p. 609\)](#) for details on bar graphs).

Examples

```
graph gra1.line(s, p) gdp m1 inf
```

creates and prints a stacked line graph object named GRA1. This command is equivalent to running the command:

```
line(s, p) gdp m1 inf
```

freezing the view, and naming the graph GRA1.

```
graph mygra.merge gr_line gr_scatt gr_pie
```

creates a multiple graph object named MYGRA that merges three graph objects named GR_LINE, GR_SCAT, and GR_PIE.

Cross-references

See [Chapter 15. “Graphs, Tables, Text, and Spools,” on page 523](#) of the *User's Guide I* for a general discussion of graphs.

See also [freeze \(p. 724\)](#) and [Graph::merge \(p. 163\)](#).

label	Graph View Graph Procs
-------	--

Display or change the label view of a graph object, including the last modified date and display name (if any).

As a procedure, `label` changes the fields in the graph label.

Syntax

```
graph_name.label
graph_name.label(options) [text]
```

Options

The first version of the command displays the label view of the graph. The second version may be used to modify the label. 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 <i>text</i> .
s	Sets the source field to <i>text</i> .
u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of GRA1 with “Data from CPS 1988 March File”:

```
gra1.label(r)
gra1.label(r) Data from CPS 1988 March File
```

To append additional remarks to GRA1, and then to print the label view:

```
gra1.label(r) Log of hourly wage
gra1.label(p)
```

To clear and then set the units field, use:

```
gra1.label(u) Millions of bushels
```

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels.

See also [Graph::displayname \(p. 154\)](#).

legend	Graph Procs
--------	-----------------------------

Set legend appearance and placement in graphs.

When `legend` is used with a multiple graph, the legend settings apply to all graphs. See [Graph::setelem \(p. 171\)](#) for setting legends for individual graphs in a multiple graph.

Syntax

`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

<code>columns(arg)</code> (<i>default</i> = “auto”)	Columns for legend: “auto” (automatically choose number of columns), <i>int</i> (put legend in specified number of columns).
<code>display/-display</code>	Display/do not display the legend.
<code>inbox/-inbox</code>	Put legend in box/remove box around legend.
<code>position(arg)</code>	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), “botright” or “br” (place right-justified legend below graph), “(<i>h</i> , <i>v</i>)” (the first number <i>h</i> specifies the number of virtual inches to offset to the right from the origin. The second number <i>v</i> specifies the virtual inch offset below the origin. The origin is the upper left hand corner of the graph).
<code>font([face], [pt], [+/- b], [+/- i], [+/- u], [+/- s])</code>	Set characteristics of legend font. The font name (<i>face</i>), size (<i>pt</i>), and characteristics are all optional. <i>face</i> should be a valid font name, enclosed in double quotes. <i>pt</i> . should be the font size in points. The remaining options specify whether to turn on/off boldface (b), italic (i), underline (u), and strikeout (s) styles.

<code>textcolor(arg)</code>	Sets the color of the legend text. <i>arg</i> may be one of the predefined color keywords, or it may be made up of <i>n1</i> , <i>n2</i> , <i>n3</i> , a set of three integers from 0 to 255, representing the RGB values of the color. For a description of the available color keywords (“blue”, “red”, “green”, “black”, “white”, “purple”, “orange”, “yellow”, “gray”, “ltgray”), see Table::setfillcolor (p. 516) .
<code>fillcolor(arg)</code>	Sets the background fill color of the legend box. <i>arg</i> may be one of the predefined color keywords, or it may be made up of <i>n1</i> , <i>n2</i> , <i>n3</i> , a set of three integers from 0 to 255, representing the RGB values of the color. For a description of the available color keywords (“blue”, “red”, “green”, “black”, “white”, “purple”, “orange”, “yellow”, “gray”, “ltgray”), see Table::setfillcolor (p. 516) .
<code>framecolor(arg)</code>	Sets the color of the legend box frame. <i>arg</i> may be one of the predefined color keywords, or it may be made up of <i>n1</i> , <i>n2</i> , <i>n3</i> , a set of three integers from 0 to 255, representing the RGB values of the color. For a description of the available color keywords (“blue”, “red”, “green”, “black”, “white”, “purple”, “orange”, “yellow”, “gray”, “ltgray”), see Table::setfillcolor (p. 516) .

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

The default settings are taken from the global defaults.

Examples

```
mygral.legend display position(l) inbox
```

places the legend of MYGRA1 in a box to the left of the graph.

```
mygral.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.

```
mygral.legend font("Times", 12, b, i) textcolor(red) fill-  
color(blue) framecolor(blue)
```

sets the legend font to red “Times” 12pt bold italic, and changes both the legend fill and frame colors to blue.

Cross-references

See [Chapter 15. “Graphs, Tables, Text, and Spools,” on page 523](#) of the *User’s Guide I* for a discussion of graph objects in EViews.

See [Graph::addtext](#) (p. 145) and [Graph::textdefault](#) (p. 179). See [Graph::setelem](#) (p. 171) for changing legend text and other graph options.

merge	Graph Procs
-------	-----------------------------

Merge graph objects.

`merge` combines graph objects into a single graph object. The graph objects to merge must exist in the current workfile.

Syntax

```
graph_name.merge graph1 graph2 [graph3 ...]
```

Follow the keyword with a list of existing graph object names to merge.

Examples

```
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

See [Chapter 15. “Graphs, Tables, Text, and Spools,”](#) on page 523 of the *User’s Guide I* for a discussion of graphs.

metafile	Graph Procs
----------	-----------------------------

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 [Graph::save](#) (p. 168), which allows for saving graphs in metafile or postscript files, with additional options for controlling the output.

name	Graph Procs
------	-----------------------------

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_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 current legend will be removed from the legend/axis label.

Examples

```
graph 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.

```
graph g2.scatt id w h
g2.name(1)
g2.name(2) weight
g2.name(3) height
g2.legend(1)
```

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 15. “Graphs, Tables, Text, and Spools,” on page 523](#) of the *User’s Guide I* for a discussion of working with graphs.

See also [Graph::displayname \(p. 154\)](#).

options	Graph Procs
---------	-----------------------------

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_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

Basic Graph Options

<code>size(w, h)</code>	Specifies the size of the plotting frame in virtual inches (w = width, h = height).
<code>lineauto</code>	Use solid lines when drawing in color and use patterns and grayscale when drawing in black and white.
<code>linesolid</code>	Always use solid lines.
<code>linepat</code>	Always use line patterns.
<code>color / -color</code>	Specifies that lines/filled areas [use / do not use] color. Note that if the “lineauto” 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.
<code>barlabelabove / -barlabelabove</code>	[Place / Do not place] text value of data above bar in bar graph.
<code>barlabelinside / -barlabelinside</code>	[Place / Do not place] text value of data inside bar in bar graph.
<code>outlinebars / -outlinebars</code>	[Outline / Do not outline] bars in a bar graph.
<code>outlinearea / -outlinearea</code>	[Outline / Do not outline] areas in an area graph.
<code>outlineband / -outlineband</code>	[Outline / Do not outline] bands in an area band graph.
<code>barspace / -barspace</code>	[Put / Do not put] space between bars in bar graph.
<code>pielabel / -pielabel</code>	[Place / Do not place] text value of data in pie chart.
<code>barfade(arg)</code>	Sets the fill fade of the bars in a bar graph. <i>arg</i> may be: “none” (solid fill - default), “3d” (3D rounded fill), “lzero” (light at zero), “dzero” (dark at zero).

Graph Grid Options

<code>gridl / -gridl</code>	[Turn on / Turn off] grid lines on the left scale.
<code>gridr / -gridr</code>	[Turn on / Turn off] grid lines on the right scale.
<code>gridb / -gridb</code>	[Turn on / Turn off] grid lines on the bottom scale.

<code>gridt / -gridt</code>	[Turn on / Turn off] grid lines on the top scale.
<code>gridcolor(arg)</code>	Sets the grid line color. <i>arg</i> may be one of the predefined color keywords, or it may be made up of <i>n1</i> , <i>n2</i> , <i>n3</i> , a set of three integers from 0 to 255, representing the RGB values of the color. For a description of the available color keywords (“blue”, “red”, “green”, “black”, “white”, “purple”, “orange”, “yellow”, “gray”, “ltgray”), see Table::setfillcolor (p. 516).
<code>gridwidth(n)</code>	Sets the width of the grid lines in points. <i>n</i> should be a number between 0.25 and 5.
<code>gridpat(index)</code>	Sets the line pattern for grid lines to the type specified by <i>index</i> . <i>index</i> can be an integer from 1 to 12 or one of the matching keywords (“solid”, “dash1” through “dash10”, “none”). See Graph::setelem (p. 171) for a description of the available patterns. The “none” keyword turns on solid lines.

Background and Frame Options

<code>fillcolor(arg)</code>	Sets the fill color of the graph frame. <i>arg</i> may be one of the predefined color keywords, or it may be made up of <i>n1</i> , <i>n2</i> , <i>n3</i> , a set of three integers from 0 to 255, representing the RGB values of the color. For a description of the available color keywords (“blue”, “red”, “green”, “black”, “white”, “purple”, “orange”, “yellow”, “gray”, “ltgray”), see Table::setfillcolor (p. 516).
<code>backcolor(arg)</code>	Sets the background color of the graph. <i>arg</i> may be one of the predefined color keywords, or it may be made up of <i>n1</i> , <i>n2</i> , <i>n3</i> , a set of three integers from 0 to 255, representing the RGB values of the color. For a description of the available color keywords (“blue”, “red”, “green”, “black”, “white”, “purple”, “orange”, “yellow”, “gray”, “ltgray”), see Table::setfillcolor (p. 516).
<code>framecolor(arg)</code>	Sets the background color of the graph frame. <i>arg</i> may be one of the predefined color keywords, or it may be made up of <i>n1</i> , <i>n2</i> , <i>n3</i> , a set of three integers from 0 to 255, representing the RGB values of the color. For a description of the available color keywords (“blue”, “red”, “green”, “black”, “white”, “purple”, “orange”, “yellow”, “gray”, “ltgray”), see Table::setfillcolor (p. 516).
<code>fillfade(arg)</code>	Sets the fill fade of the graph frame. <i>arg</i> may be: “none” (solid frame fill - default), “ltop” (light at top), “dtop” (dark at top).

<code>backfade(<i>arg</i>)</code>	Sets the background fade of the graph. <i>arg</i> may be: “none” (solid background - default), “ltop” (light at top), “dtop” (dark at top).
<code>framewidth(<i>n</i>)</code>	Sets the width of the graph frame in points. <i>n</i> should be a number between 0.25 and 5.
<code>frameaxes(<i>arg</i>)</code>	Specifies which frame axes to display. <i>arg</i> may be one of the keywords: “all”, “none”, or “labeled” (all axes that have labels), or any combination of letters “l” (left), “r” (right), “t” (top), and “b” (bottom), e.g. “lrt” for left, right and top.
<code>indenth(<i>n</i>)</code>	Sets the horizontal indentation of the graph from the graph frame in virtual inches. <i>n</i> should be a number between 0 and 0.75.
<code>indentv(<i>n</i>)</code>	Sets the vertical indentation of the graph from the graph frame in virtual inches. <i>n</i> should be a number between 0 and 0.75.
<code>background / -background</code>	[Include / Do not include] the background color when exporting or printing the graph.

The options which support the “-” may be preceded 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.options size(4,4) +inbox color
```

sets GRAPH1 to use a 4×4 frame enclosed in a box. The graph will use color.

```
graph1.options linepat -color size(2,8) -inbox
```

sets GRAPH1 to use a 2×8 frame with no box. The graph does not use color, with the lines instead being displayed using patterns.

```
graph1.options fillcolor(gray) backcolor(192, 192, 192)
framecolor(blue)
```

sets the fill color of the graph frame to gray, the background color of the graph to the RGB values 192, 192, and 192, and the graph frame color to blue.

```
graph1.options gridpat(3) gridl -gridb
```

display left scale grid lines using line pattern 3 (“dash2”) and turn off display of vertical grid lines from the bottom axis.

```
graph1.options indenth(.5) frameaxes(lb) framewidth(.5)
      gridwidth(.25)
```

indents the graph .5 virtual inches from the frame, displays left and bottom frame axes of width .5 points, and sets the gridline width to .25 points.

Cross-references

See [Chapter 15. “Graphs, Tables, Text, and Spools,” on page 523](#) of the *User’s Guide I* for a discussion of graph options in EViews.

See also [Graph::axis \(p. 149\)](#), [Graph::datelabel \(p. 152\)](#), and [Graph::setelem \(p. 171\)](#).

save	Graph Procs
------	-----------------------------

Save a graph object to disk as a Windows metafile (.EMF or .WMF), PostScript (.EPS), bitmap (.BMP), Graphics Interchange Format (.GIF), Joint Photographic Experts Exchange (.JPEG), or Portable Network Graphics (.PNG) file.

Syntax

```
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. A graph may be saved with an .EMF, .WMF, .EPS, .BMP, .GIF, .JPEG, or .PNG extension.

If an explicit path is not specified, the file will be stored in the default directory, as set in the **File Locations** global options.

General Graph Options

t = <i>file_type</i>	Specifies the file type, where <i>file_type</i> may be one of: Enhanced Windows metafile (“emf” or “meta”), ordinary Windows metafile (“wmf”), Encapsulated PostScript (“eps” or “ps”), Bitmap file (“bmp”), Graphics Interchange Format (“gif”), Joint Photographic Experts Exchange (“jpeg” or “jpg”), or Portable Network Graphics (“png”). Files will be saved with the “emf”, “wmf”, “eps”, “bmp”, “gif”, “jpeg”, and “png” extensions, respectively.
u = <i>units</i>	Specify units of measurement, where <i>units</i> is one of: “in” (inches), “cm” (centimeters), “pt” (points), “pica” (picas), “pixels” (pixels). Note: pixels are only applicable to bmp, gif, jpeg, and png files. Default is inches otherwise.
w = <i>width</i>	Set width of the graphic in the selected units.

<code>h = height</code>	Set height of the graphic in the selected units.
<code>c / -c</code>	[Save / Do not save] the graph in color.
<code>d = dpi</code>	Specify the number of dots per inch. Only applicable to bmp, gif, jpeg, and png files when units has not been set to pixels. In the case units = “pixels”, it is ignored.

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. (Note that the aspect ratio for an ordinary Windows Metafile (.WMF) cannot be unlocked, so only a height or width should be specified in this case.) EViews will default to the current graph dimensions if size is unspecified.

All defaults with exception to dots per inch are taken from the global graph export settings (**Options/Graphics Defaults.../Exporting**). The default dots per inch for bmp, gif, jpeg, and png file types is equal to the number of pixels per logical inch along the screen width of your system. Values may therefore differ from system to system.

Postscript specific Graph Options

<code>box / -box</code>	[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.
<code>land</code>	Save the graph in landscape orientation. The default uses portrait mode.

Examples

```
graph1.save(t=ps, -box, land) c:\data\MyGra1
```

saves GRAPH1 as a PostScript file MYGRA1.EPS. The graph is saved in landscape orientation without a bounding box.

```
graph2.save(t=emf, u=pts, w=300, h=300) MyGra2
```

saves GRAPH2 in the default directory as an Enhanced Windows metafile MYGRA2.EMF. The image will be scaled to 300 × 300 points.

```
graph3.save(t=png, u=in, w=5, d=300) MyGra3
```

saves GRAPH3 in the default directory as a PNG file MYGRA3.PNG. The image will be 5 inches wide at 300 dpi.

Cross-references

See [Chapter 15. “Graphs, Tables, Text, and Spools,”](#) beginning on page 523 of the *User’s Guide I* for a discussion of graphs.

scale	Graph Procs
-------	-----------------------------

The **scale** command is supported for backward compatibility, but has been replaced in EViews 6 by the [Graph::axis](#) (p. 149) command, which handles all axis and scaling options.

setbpelem	Graph Procs
-----------	-----------------------------

Enable/disable individual boxplot elements.

Syntax

`graph_name.setbpelem element_list`

The *element_list* may contain one or more of the following:

median, med / - median, -med	[Show / Do not show] the medians.
mean / -mean	[Show / Do not show] the means.
whiskers, w / -whiskers, -w	[Show / Do not show] the whiskers (lines from the box to the staples).
staples, s / -staples, -s	[Show / Do not show] the staples (lines drawn at the last data point within the inner fences).
near / -near	[Show / Do not show] the near outliers (values between the inner and outer fences).
far / -far	[Show / Do not show] the far outliers (values beyond the outer fences).
width(<i>arg</i>) (<i>default</i> = "fixed")	Set the width settings for the boxplots, where <i>arg</i> is one of: "fixed" (uniform width), "n" (proportional to sample size), "rootn" (proportional to the square root of sample size).
ci = <i>arg</i> (<i>default</i> = "shade")	Set the display method for the confidence intervals, where <i>arg</i> is one of: "none" (do not display), "shade" (shaded intervals), "notch" (notched intervals).
ci = <i>arg</i> (<i>default</i> = "shade")	

Examples

`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 [“Boxplot” on page 477](#) of the *User’s Guide I* for a description of boxplots.

See [Graph::setelem \(p. 171\)](#) to modify line and symbol attributes. See also [Graph::options \(p. 164\)](#) and [Graph::axis \(p. 149\)](#).

setelem	Graph Procs
---------	-----------------------------

Set individual line, bar and legend options for each series in the graph.

Syntax

```
graph_name.setelem(graph_elem) argument_list
```

where *graph_elem* is the identifier for the graph element whose options you wish to modify:

<i>integer</i>	Index for graph element (for non-boxplot graphs). For example, if you provide the integer “2”, EViews will modify the second line in the graph.
<i>box_elem</i>	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:

<code>symbol(arg)</code>	<p>Sets the drawing symbol: <i>arg</i> can be an integer from 1–13, or one of the matching keywords. “obslabel” and “dotobslabel” use the observation label as the symbol.</p> <p>Selecting a symbol automatically turns on symbol use. The “none” option turns off symbol use.</p>	<table><tr><td>(1) circle</td><td></td></tr><tr><td>(2) filledcircle</td><td></td></tr><tr><td>(3) transcircle</td><td></td></tr><tr><td>(4) star</td><td></td></tr><tr><td>(5) diagcross</td><td></td></tr><tr><td>(6) cross</td><td></td></tr><tr><td>(7) filledsquare</td><td></td></tr><tr><td>(8) square</td><td></td></tr><tr><td>(9) filledtriup</td><td></td></tr><tr><td>(10) triup</td><td></td></tr><tr><td>(11) filledtridown</td><td></td></tr><tr><td>(12) tridown</td><td></td></tr><tr><td>(13) obslabel</td><td>Obs. Label</td></tr><tr><td>(14) dotobslabel</td><td> Obs. Label</td></tr><tr><td>(15) none</td><td></td></tr></table>	(1) circle		(2) filledcircle		(3) transcircle		(4) star		(5) diagcross		(6) cross		(7) filledsquare		(8) square		(9) filledtriup		(10) triup		(11) filledtridown		(12) tridown		(13) obslabel	Obs. Label	(14) dotobslabel	Obs. Label	(15) none	
(1) circle																																
(2) filledcircle																																
(3) transcircle																																
(4) star																																
(5) diagcross																																
(6) cross																																
(7) filledsquare																																
(8) square																																
(9) filledtriup																																
(10) triup																																
(11) filledtridown																																
(12) tridown																																
(13) obslabel	Obs. Label																															
(14) dotobslabel	Obs. Label																															
(15) none																																
<code>symbolsize(arg),</code> <code>symsize(arg)</code>	<p>Sets the symbol size. <i>arg</i> may be an integer between 1-8, where 1 is the smallest symbol and 8 is the largest, or one of the keywords: “XS” (X-Small), “S” (Small), “M” (Medium), “L” (Large), “XL” (X-Large), “2XL” (2X-Large), “3XL” (3X-Large), “4XL” (4X-Large).</p>																															
<code>linecolor(args),</code> <code>lcolor(args)</code>	<p>Sets the line and symbol color. The <i>args</i> 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 (“blue”, “red”, “green”, “black”, “white”, “purple”, “orange”, “yellow”, “gray”, “ltgray”). For a full description of the keywords, see Table::setfillcolor (p. 516).</p>																															
<code>linewidth(n1),</code> <code>lwidth(n1)</code>	<p>Sets the line and symbol width: <i>n1</i> should be a number between “.25” and “5”, indicating the width in points.</p>																															

linepattern(*arg*),
lpat(*arg*)

Sets the line pattern to the type specified by *arg*. *arg* can be an integer from 1–12 or one of the matching keywords.

Note that the option interacts with the graph options for “color”, “lineauto”, “linesolid”, “linepat” (see [Graph::options](#) (p. 164), for details). You may need to set the graph option for “linepat” to enable the display of line patterns. See [Graph::options](#) (p. 164).

Note also that the patterns with index values 7–11 have been modified since version 5.0. In particular, the “none” option has been moved to position 12.

The “none” option turns off lines and uses only symbols.
















(1) solid	—————
(2) dash1	- - - - -
(3) dash2	- - - - -
(4) dash3	- - - - -
(5) dash4	- - - - -
(6) dash5	- - - - -
(7) dash6	- - - - -
(8) dash7	- - - - -
(9) dash8	- - - - -
(10) dash9	- - - - -
(11) dash10	- - - - -
(12) none	

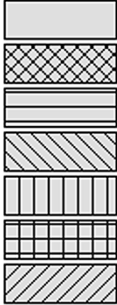
fillcolor(*arg*),
fcolor(*arg*)

Sets the fill color for symbols, bars, and pies. The *args* value may set by using of the color keywords (“blue”, “red”, “green”, “black”, “white”, “purple”, “orange”, “yellow”, “gray”, “ltgray”) or by using the RGB values (*e.g.*, “@RGB(255, 255, 0)”). For a full description of the keywords, see [Table::setfillcolor](#) (p. 516)

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).

1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

<code>fillhatch(arg), hatch(arg)</code>	Sets the hatch characteristics for bars and pies: <i>arg</i> can be an integer from 1–7, or one of the matching keywords.	(1) none (2) diagcross (3) horizontal (4) fdiagonal (5) vertical (6) cross (7) bdiagonal	
<code>preset(n1)</code>	Sets line and fill characteristics to the specified EViews preset values, where <i>n1</i> is an integer from 1–30. Simultaneously sets “linecolor”, “linepattern”, “linewidth”, “symbol”, “fillcolor”, “fillgray”, and “fillhatch” to the EViews predefined definitions for graph element <i>n1</i> . When applied to boxplots, the line color of the specified element will be applied to the box, whiskers, and staples.		
<code>default(n1)</code>	Sets line and fill characteristics to the specified user-defined default settings where <i>n1</i> is an integer from 1–30. Simultaneously sets “linecolor”, “linepattern”, “linewidth”, “symbol”, “fillcolor”, “fillgray”, and “fillhatch” to the values in the user-defined global defaults for graph element <i>n1</i> . When applied to boxplots, the line color of the specified settings will be applied to the box, whiskers, and staples.		
<code>axis(arg), axis scale(arg)</code>	Assigns the element to an axis: left (“l”), right (“r”), bottom (“b”), top (“t”). The latter two options are only applicable for XY and scatter graphs (scat (p. 640), xyarea (p. 656), xybar (p. 659), xyline (p. 661), xypair (p. 664)).		
<code>legend(str)</code>	Assigns legend text for the element. <i>str</i> will be used in the legend to label the element.		

Examples

```
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.

graph1.setelem(1) lcolor(blue)
graph1.setelem(1) linecolor(0, 0, 255)
are equivalent methods of setting the linecolor to blue.
```

```
graph1.setelem(1) fillgray(6)
```

sets the gray-scale color for the first graph element.

The lines:

```
graph1.setelem(1) scale(1)
graph1.setelem(2) scale(1)
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.

```
graph1.setelem(2) legend("gross domestic product")
```

sets the legend for the second graph element.

Cross-references

See [Chapter 15. “Graphs, Tables, Text, and Spools,”](#) on page 523 of the *User’s Guide I* for a discussion of graph options in EViews.

See also [Graph::axis](#) (p. 149), [Graph::datelabel](#) (p. 152) and [Graph::options](#) (p. 164).

setobslabel	Graph Procs
-------------	-----------------------------

Sets custom axis labels for the observation scale of a graph.

Syntax

```
graph_name.setobslabel([step_options,] init_options) label1 label2 ...
```

Follow the keyword with a list of axis labels. To preserve case, enclose the label in quotation marks. To hide a label, use “”. If the number of labels provided is less than the number of existing labels, the remaining labels will not be affected.

Options

Step options

<i>start</i> [, <i>step</i>]	<i>start</i> should be the observation number of the first label to modify. <i>step</i> defines the number of observations to skip between applying labels.
-------------------------------	---

Init options

<i>init_opt</i> (default = "blank")	<i>init_opt</i> sets the initialization options for the labels: "current" (keeps current labels, or initializes the labels with dates/observation numbers if custom labels do not currently exist, then adds the labels provided), "blank" (sets all labels to empty strings, then adds the labels provided), "clear" (delete custom labels if they exist and return to automatic labeling, and ignore labels provided).
---	--

Examples

To create editable labels on the time axis of GRA1 and change the first label to "CA," use the command:

```
gra1.setobslabel(current) "CA"
```

Note that all but the first label remain unchanged.

To keep the first label as "CA" and set the second label to "OR", you could enter:

```
gra1.setobslabel(current) "CA" "OR"
```

Alternatively, an equivalent command would be

```
gra1.setobslabel(2,current) "OR"
```

which starts applying labels at the second observation.

To set the first, third, and fifth observation labels and leave all others unchanged:

```
graph2.setobslabel(1,2,current) "first" "third" "fifth"
```

This instructs EViews to begin modifying at the first label and step two observations between new labels.

```
graph2.setobslabel(1,2,blank) "first" "third" "fifth"
```

performs the same operation as the previous command, while also clearing out all other labels.

```
graph2.setobslabel(clear)
```

deletes all custom labels and returns to EViews automatic labeling.

Cross-references

See [Chapter 15. "Graphs, Tables, Text, and Spools," on page 523](#) of the *User's Guide I* for a discussion of graph options.

See also [Graph::datelabel \(p. 152\)](#), [Graph::axis \(p. 149\)](#), [Graph::options \(p. 164\)](#) and [Graph::setelem \(p. 171\)](#).

sort	Graph Procs
------	-----------------------------

Sort the series in a graph.

The `sort` command sorts *all* series in the graph on the basis of the values of up to three 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.

Note that sorting cannot be undone. You may wish to freeze or copy the graph before applying the sort.

Syntax

```
graph_name.sort(series1[, series2, series3])
```

Follow the keyword with a list of the series by which you wish to sort the graph. 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.

The series may be specified using the series display name or the index of the series in the graph. For example, if you provide the integer “2”, EViews will use the second series. To sort by observation labels, use the integer “0” or the keyword “Obs label”.

To sort in descending order, precede the series name with a minus sign (“-”).

Note that a graph with more than 500 observations cannot be sorted.

Examples

```
gral.sort(x,y)
```

sorts graph GRA1 first by the series X. Any ties in X will be resolved by the series Y.

If X is the first series in graph GRA1 and Y is the second series,

```
gral.sort(1,-2)
```

sorts first in ascending order by X and then in descending order by Y.

```
gral.sort(0)
```

sorts GRA1 by its observation labels.

template	Graph Procs
----------	-----------------------------

Apply a template to a graph object.

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_name.template(options) template
```

Follow the name of the graph to which you want to apply the template options with a period, the keyword `template`, and the name of a graph template. *template* may be one of the predefined template keywords: “default” (current global defaults), “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”, or a named graph in the workfile.

Options

t	Replace text and line/shade objects with those of the template graph, when <i>template</i> is the name of a graph in the workfile.
e	Apply template settings to existing text and line/shade options.
b / -b	[Apply / Remove] bold modifiers of the specified <i>pre-defined</i> template style.
w / -w	[Apply / Remove] wide modifiers of the specified <i>pre-defined</i> template style.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Examples

```
gra_cs.template gra_gdp
```

applies the option settings in the graph object GRA_GDP to the graph GRA_CS. Text and line shading options from GRA_GDP will be applied to GRA_CS, but the characteristics of existing text and line/shade objects in GRA_CS will not be modified. Text and shading objects include those added with the [Graph::addtext](#) (p. 145) or [Graph::draw](#) (p. 154) commands.

```
g1.template(t) mygraph1
```

applies the option settings of MYGRAPH1, and all text and shadings in the template graph, to the graph G1. Note that the “t” option overwrites any existing text and shading objects in the target graph.

```
graph1.template(e) modern
```

applies the predefined template “modern” to GRAPH1, also changing the settings of existing text and line/shade objects in the graph.

```
graph1.template(e, b, w) reverse
```

applies the predefined template “reverse” to GRAPH1, with the *bold* and *wide* modifiers. Any existing text and line/shade objects in GRAPH1 are also modified to use the object settings of the monochrome template.

```
graph1.template(-w) monochrome
```

applies the monochrome settings to GRAPH1, removing the wide modifier.

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 [“Templates” on page 536](#) of the *User’s Guide I* for additional discussion.

textdefault	Graph Procs
-------------	-----------------------------

Change default settings for text objects in the graph.

This command specifies changes in the default settings which will be applied to text objects added subsequently to the graph. If you include the “existing” option, *all* of the text default settings will also be applied to existing text objects in the graph.

Syntax

```
graph_name.textdefault text_options
```

where *text_options* include one or more of one of the following:

<code>font([<i>face</i>], [<i>pt</i>], [+/− b], [+/− i], [+/− u], [+/− s])</code>	Set characteristics of default text font. The font name (<i>face</i>), size (<i>pt</i>), and characteristics are all optional. <i>face</i> should be a valid font name, enclosed in double quotes. <i>pt</i> should be the font size in points. The remaining options specify whether to turn on/off boldface (b), italic (i), underline (u), and strikeout (s) styles.
<code>textcolor(<i>arg</i>)</code>	Sets the default color of the text. <i>arg</i> may be one of the predefined color keywords, or it may be made up of <i>n1</i> , <i>n2</i> , <i>n3</i> , a set of three integers from 0 to 255, representing the RGB values of the color. For a description of the available color keywords (“blue”, “red”, “green”, “black”, “white”, “purple”, “orange”, “yellow”, “gray”, “ltgray”), see Table::setfillcolor (p. 516) .
<code>fillcolor(<i>arg</i>)</code>	Sets the default background fill color of the text box. <i>arg</i> may be one of the predefined color keywords, or it may be made up of <i>n1</i> , <i>n2</i> , <i>n3</i> , a set of three integers from 0 to 255, representing the RGB values of the color. For a description of the available color keywords (“blue”, “red”, “green”, “black”, “white”, “purple”, “orange”, “yellow”, “gray”, “ltgray”), see Table::setfillcolor (p. 516) .
<code>framecolor(<i>arg</i>)</code>	Sets the default color of the text box frame. <i>arg</i> may be one of the predefined color keywords, or it may be made up of <i>n1</i> , <i>n2</i> , <i>n3</i> , a set of three integers from 0 to 255, representing the RGB values of the color. For a description of the available color keywords (“blue”, “red”, “green”, “black”, “white”, “purple”, “orange”, “yellow”, “gray”, “ltgray”), see Table::setfillcolor (p. 516) .
<code>existing</code>	Apply the default settings to all existing text objects in the graph.

The options which support the “−” may be preceded by a “+” or “−” indicating whether to turn on or off the option. The “+” is optional.

Examples

```
graph1.textdefault font("Arial", b) fillcolor(gray) existing
```

changes the default text settings for new text objects so that new text is in Arial bold, using the current default font size and color. Should the new text be enclosed in a box, the box will have a gray fill. Additionally, the “existing” keyword specifies that existing text objects in the graph will be updated with the current text settings. Note that in addition to the font type and fill color specified in the command, all text default settings will be applied to the existing text.

```
graph1.textdefault existing
```


updates the text objects in GRAPH1 with the current text default settings.

Cross-references

See [Chapter 15. “Graphs, Tables, Text, and Spools,” on page 523](#) of the *User’s Guide I* for a discussion of graph options.

See [Graph::addtext \(p. 145\)](#) and [Graph::legend \(p. 161\)](#).

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. 202](#)).

To declare a group, enter the keyword `group`, followed by a name, and optionally, a list of series or expressions:

```
group salesvrs
group nipa cons(-1) log(inv) g x
```

Additionally, a number of object procedures will automatically create a group.

Group Views

[cause](#)pairwise Granger causality tests ([p. 186](#)).
[coint](#)Johansen cointegration test ([p. 187](#)).
[cor](#)correlation matrix between series ([p. 190](#)).
[correl](#)correlogram of the first series in the group ([p. 193](#)).
[cov](#)covariance matrix between series ([p. 194](#)).
[cross](#)cross correlogram of the first two series ([p. 197](#)).
[dtable](#)dated data table ([p. 199](#)).
[freq](#)frequency table *n*-way contingency table ([p. 200](#)).
[label](#)label information for the group ([p. 203](#)).
[pcomp](#)principal components analysis ([p. 206](#)).
[sheet](#)spreadsheet view of the series in the group ([p. 214](#)).
[stats](#)descriptive statistics ([p. 216](#)).
[testbtw](#)tests of equality for mean, median, or variance, between series in group ([p. 218](#)).
[uroot](#)unit root test on the series in the group ([p. 219](#)).

Group Graph Views

Graph creation types are discussed in detail in “[Graph Creation Commands](#)” on [page 601](#).

[area](#)area graph of the series in the group ([p. 603](#)).
[band](#)area band graph ([p. 606](#)).
[bar](#)single or multiple bar graph view of all series ([p. 609](#)).
[boxplot](#)boxplot of each series in the group ([p. 613](#)).
[distplot](#)distribution graph ([p. 615](#)).
[dot](#)dot plot graph ([p. 622](#)).
[errbar](#)error bar graph view ([p. 626](#)).

hilo high-low(-open-close) chart (p. 628).
line single or multiple line graph view of all series (p. 630).
pie pie chart view (p. 633).
qqplot quantile-quantile plots (p. 636).
scat scatterplot (p. 640).
scatmat matrix of all pairwise scatter plots (p. 644).
scatpair scatterplot pairs graph (p. 647).
seasplot seasonal line graph (p. 651).
spike spike graph (p. 652).
xyarea XY area graph (p. 656).
xybar XY bar graph (p. 659).
xyline XY line graph (p. 661).
xypair XY pairs graph (p. 664).

Group Procs

add add one or more series to the group (p. 185).
displayname set display name (p. 198).
drop drop one or more series from the group (p. 199).
makecomp save the scores from a principal components analysis of the series in a group (p. 204).
resample resample from rows of group (p. 207).
setformat set the display format in the group spreadsheet for the specified series (p. 209).
setindent set the indentation in the group spreadsheet for the specified series (p. 212).
setjust set the justification in the group spreadsheet for the specified series (p. 213).
setWidth set the column width in the group spreadsheet for the specified series (p. 214).
sort change display order for group spreadsheet (p. 215).
stom convert the group to a vector or matrix (p. 216).
stomna convert the group to a vector or matrix without dropping NAs (p. 217).

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.

@minobsnumber of non-missing observations in the current sample for the shortest series in the group.

@maxobsnumber 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 may enter:

```
group g1 gdp income
```

To change the contents of an existing group, you can repeat the declaration, or use the `add` and `drop` commands:

```
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:

```
g1.freq
```

```
g1.cov
```

```
g1.testbtw(var,c)
```

You can index selected series in the group:

```
show g1(2).line
```

```
series sum=g1(1)+g1(2)
```

To create a scalar containing the number of series in the group, use the command:

```
scalar nseries=g1.@count
```

Group Entries

The following section provides an alphabetical listing of the commands associated with the “[Group](#)” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

add	Group Procs
------------	-----------------------------

Add series to a group.

Syntax

```
group_name.add arg1 [arg2 arg3 ...]
```

List the names of series or a group of series to add to the group.

Examples

```
dummy.add d11 d12
```

Adds the two series D11 and D12 to the group DUMMY.

Cross-references

See [“Groups” on page 84](#) of the *User’s Guide I* for additional discussion of groups. [“Cross-section Identifiers” on page 461](#) of the *User’s Guide II* discusses pool identifiers.

See also `Group::drop` (p. 199).

cause	Group Views
-------	-----------------------------

Granger causality test.

Performs pairwise Granger causality tests between (all possible) pairs of the group of series.

Syntax

```
group_name.cause(n, options)
```

Options

You must specify the number of lags *n* to use for the test by providing an integer in parentheses after the 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

To compute Granger causality tests of whether GDP Granger causes M1 and whether M1 Granger causes GDP, you may enter the commands:

```
group g1 gdp m1
g1.cause(4)
```

The regressors of each test are a constant and four lags of GDP and M1.

The commands:

```
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 410 of the *User’s Guide I* for a discussion of Granger’s approach to testing hypotheses about causality.

See also `var::var` (p. 572).

cdfplot	Group Views
----------------	-----------------------------

Empirical distribution plot.

The `cdfplot` command is no longer supported. See [distplot](#) (p. 615).

coint	Group Views
--------------	-----------------------------

Johansen’s cointegration test.

Syntax

```
group_name.coint(test_option,n,option)
group_name.coint(option)
```

The `coint` command tests for cointegration among the series in the group. The second form of the command should be used with panel structured workfiles.

As of EViews 5, the output for cointegration tests now displays p -values for the rank test statistics. These p -values are computed using the response surface coefficients as estimated in MacKinnon, et. al. (1999). The 0.05 critical values are now based on the response surface coefficients from MacKinnon-Haug-Michelis. *Note: the reported critical values assume no exogenous variables other than an intercept and trend.*

Test Options

You must specify the test option followed by the number of lags n . You must choose one of the following six test options:

- | | |
|---|--|
| a | No deterministic trend in the data, and no intercept or trend in the cointegrating equation. |
| b | No deterministic trend in the data, and an intercept but no trend in the cointegrating equation. |
| c | Linear trend in the data, and an intercept but no trend in the cointegrating equation. |

d	Linear trend in the data, and both an intercept and a trend in the cointegrating equation.
e	Quadratic trend in the data, and both an intercept and a trend in the cointegrating equation.
s	Summarize the results of all 5 options (a-e).

Options

Non-panel options

restrict	Impose restrictions as specified by the <code>append (coint)</code> proc.
m = integer	Maximum number of iterations for restricted estimation (only valid if you choose the restrict option).
c = scalar	Convergence criterion for restricted estimation. (only valid if you choose the restrict option).
save = mat_name	Stores test statistics as a named matrix object. The save = option stores a $(k + 1) \times 4$ matrix, where k is the number of endogenous variables in the VAR. The first column contains the eigenvalues, the second column contains the maximum eigenvalue statistics, the third column contains the trace statistics, and the fourth column contains the log likelihood values. The i -th row of columns 2 and 3 are the test statistics for rank $i - 1$. The last row is filled with NAs, except the last column which contains the log likelihood value of the unrestricted (full rank) model.
cvtype = ol	Display 0.05 and 0.01 critical values from Osterwald-Lenum (1992). This option reproduces the output from version 4. The default is to display critical values based on the response surface coefficients from MacKinnon-Haug-Michelis (1999). Note that the argument on the right side of the equals sign are letters, not numbers 0-1).
cvsize = arg (default = 0.05)	Specify the size of MacKinnon-Haug-Michelis (1999) critical values to be displayed. The size must be between 0.0001 and 0.9999; values outside this range will be reset to the default value of 0.05. This option is ignored if you set “cvtype = ol”.
p	Print output of the test.

Panel options

For panel cointegration tests, you may specify the type using one of the following keywords:

Pedroni (default)	Pedroni (1994 and 2004).
Kao	Kao (1999)
Fisher	Fisher - pooled Johansen

Depending on the type selected above, the following may be used to indicate deterministic trends:

const (default)	Include a constant in the test equation. Applicable to Pedroni and Kao tests.
trend	Include a constant and a linear time trend in the test equation. Applicable to Pedroni tests.
none	Do not include a constant or time trend. Applicable to Pedroni tests.
a, b, c, d, or e	Indicate deterministic trends using the “a”, “b”, “c”, “d”, and “e” keywords, as detailed above in “Test Options” . Applicable to Fisher tests.

Additional options:

<code>ac = arg</code> (<i>default</i> = “bt”)	Method of estimating the frequency zero spectrum: “bt” (Bartlett kernel), “pr” (Parzen kernel), “qs” (Quadratic Spectral kernel). Applicable to Pedroni and Kao tests.
<code>band = arg</code> (<i>default</i> = “nw”)	Method of selecting the bandwidth, where <i>arg</i> may be “nw” (Newey-West automatic variable bandwidth selection), or a number indicating a user-specified common bandwidth. Applicable to Pedroni and Kao tests.
<code>lag = arg</code>	For Pedroni and Kao tests, the method of selecting lag length (number of first difference terms) to be included in the residual regression. For Fisher tests, a pair of numbers indicating lag.

<code>info = arg</code> <code>(default = "sic")</code>	Information criterion to use when computing automatic lag length selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn). Applicable to Pedroni and Kao tests.
<code>maxlag = int</code>	Maximum lag length to consider when performing automatic lag length selection, where <i>int</i> is an integer. Default = $\text{floor}[\min(12, T_i/3)(T_i/100)^{0.25}]$ where T_i is the length of the cross-section. Applicable to Pedroni and Kao tests.
<code>disp = arg</code> <code>(default = 500)</code>	Maximum number of individual results to be displayed.

Examples

`gr1.coint(s,4)`

summarizes the results of the Johansen cointegration test for the series in the group GR1 for all five specifications of trend. The test equation uses lags of up to order four.

For a panel structured workfile,

`grp1.coint(pedroni,maxlag=3,info=sic)`

performs Pedroni’s residual-based panel cointegration test with automatic lag selection with a maximum lag limit of 3. Automatic selection based on Schwarz criterion.

Cross-references

See [“Cointegration Testing” on page 363](#) of the *User’s Guide II* for details on the Johansen test.

cor	Group Views
------------	-----------------------------

Compute measure of association for the series in a group. You may compute measures related to Pearson product-moment (ordinary) correlations, rank correlations, or Kendall’s tau.

Syntax

`group_name.cor(options) [keywords [@partial z1 z2 z3...]]`

You should specify keywords indicating the statistics you wish to display from the list below, optionally followed by the keyword `@partial` and a list of conditioning series or groups (for the group view), or the name of a conditioning matrix (for the matrix view). In the matrix view setting, the columns of the matrix should contain the conditioning information, and the number or rows should match the original matrix.

You may specify keywords from one of the four sets (Pearson correlation, Spearman correlation, Kendall's tau, Uncentered Pearson) corresponding the computational method you wish to employ. (You may not select keywords from more than one set.)

If you do not specify *keywords*, EViews will assume “corr” and compute the Pearson correlation matrix. Note that `cor` is equivalent to the `cov` (p. 194) command with a different default setting.

Note that this view has been expanded considerably from EViews 5.1, but the syntax for the earlier version of the routine is fully compatible with the current version.

Pearson Correlation

<code>cov</code>	Product moment covariance.
<code>corr</code>	Product moment correlation.
<code>sscp</code>	Sums-of-squared cross-products.
<code>stat</code>	Test statistic (<i>t</i> -statistic) for evaluating whether the correlation is zero.
<code>prob</code>	Probability under the null for the test statistic.
<code>cases</code>	Number of cases.
<code>obs</code>	Number of observations.
<code>wgts</code>	Sum of the weights.

Spearman Rank Correlation

<code>rcov</code>	Spearman's rank covariance.
<code>rcorr</code>	Spearman's rank correlation.
<code>rsscp</code>	Sums-of-squared cross-products.
<code>rstat</code>	Test statistic (<i>t</i> -statistic) for evaluating whether the correlation is zero.
<code>rprob</code>	Probability under the null for the test statistic.
<code>cases</code>	Number of cases.
<code>obs</code>	Number of observations.
<code>wgts</code>	Sum of the weights.

Kendall's tau

<code>taub</code>	Kendall's tau-b.
<code>taua</code>	Kendall's tau-a.
<code>taucd</code>	Kendall's concordances and discordances.

taustat	Kendall's score statistic for evaluating whether the Kendall's tau-b measure is zero.
tauprob	Probability under the null for the score statistic.
cases	Number of cases.
obs	Number of observations.
wgts	Sum of the weights.

Uncentered Pearson

ucov	Product moment covariance.
ucorr	Product moment correlation.
usscp	Sums-of-squared cross-products.
ustat	Test statistic (<i>t</i> -statistic) for evaluating whether the correlation is zero.
uprob	Probability under the null for the test statistic.
cases	Number of cases.
obs	Number of observations.
wgts	Sum of the weights.

Note that `cases`, `obs`, and `wgts` are available for each of the methods.

Options

<code>wgt = name</code> (optional)	Name of series containing weights.
<code>wgtmethod = arg</code> (default = "sstdev")	<p>Weighting method (when weights are specified using "weight ="): frequency ("freq"), inverse of variances ("var"), inverse of standard deviation ("stdev"), scaled inverse of variances ("svar"), scaled inverse of standard deviations ("sstdev").</p> <p>Only applicable for ordinary (Pearson) calculations. Weights specified by "wgt =" are frequency weights for rank correlation and Kendall's tau calculations.</p>
<code>pairwise</code>	Compute using pairwise deletion of observations with missing cases (pairwise samples).
<code>df</code>	Compute covariances with a degree-of-freedom correction for the mean (for centered specifications), and any partial conditioning variables.

<code>multi = arg</code> (default = "none")	Adjustment to p -values for multiple comparisons: none ("none"), Bonferroni ("bonferroni"), Dunn-Sidak ("dunn").
<code>outfmt = arg</code> (default = "single")	Output format: single table ("single"), multiple table ("mult"), list ("list"), spreadsheet ("sheet"). Note that "outfmt = sheet" is only applicable if you specify a single statistic keyword.
<code>out = name</code>	Basename for saving output. All results will be saved in Sym matrices named using keys ("COV", "CORR", "SSCP", "TAUA", "TAUB", "CONC" (Kendall's concurrences), "DISC" (Kendall's discordances), "CASES", "OBS", "WGTS") appended to the basename (e.g., the covariance specified by "out = my" is saved in the Sym matrix "MYCOV").

`p` Print the result.

Examples

```
group grp1 height weight age
grp1.cor
```

displays a 3×3 Pearson correlation matrix for the three series in GRP1.

```
grp1.cor corr stat prob
```

displays a table containing the Pearson correlation, t -statistic for testing for zero correlation, and associated p -value, for the series in GRP1.

```
grp1.cor(pairwise) taub taustat tauprob
```

computes the Kendall's tau-b, score statistic, and p -value for the score statistic, using samples with pairwise missing value exclusion.

```
grp1.cor(out=aa) cov @partial gender
```

computes the Pearson covariance for the series in GRP1 conditional on GENDER and saves the results in the symmetric matrix object AACOV.

Cross-references

See also [cov](#) (p. 194), [@cor](#) (p. 847), and [@cov](#) (p. 847).

correl	Group Views
--------	-----------------------------

Display autocorrelation and partial correlations.

Displays the autocorrelation and partial correlation functions of the first series in the group, together with the Q -statistics and p -values associated with each lag.

Syntax

```
group_name.correl(n, options)
```

You must specify the largest lag *n* to use when computing the autocorrelations.

Options

p	Print the correlograms.
---	-------------------------

Examples

```
gr1.correl(24)
```

Displays the correlograms of group GR1 for up to 24 lags.

Cross-references

See [“Autocorrelations \(AC\)” on page 325](#) and [“Partial Autocorrelations \(PAC\)” on page 326](#) of the *User’s Guide I* for a discussion of autocorrelation and partial correlation functions, respectively.

COV	Group Views
-----	-----------------------------

Compute measure of association for the series in a group. You may compute measures related to Pearson product-moment (ordinary) covariances, rank covariances, or Kendall’s tau.

Syntax

```
group_name.cov(options) [keywords [@partial z1 z2 z3...]]
```

You should specify keywords indicating the statistics you wish to display from the list below, optionally followed by the keyword `@partial` and a list of conditioning series or groups (for the group view), or the name of a conditioning matrix (for the matrix view). In the matrix view setting, the columns of the matrix should contain the conditioning information, and the number or rows should match the original matrix.

You may specify keywords from one of the four sets (Pearson correlation, Spearman correlation, Kendall’s tau, Uncentered Pearson) corresponding the computational method you wish to employ. (You may not select keywords from more than one set.)

If you do not specify *keywords*, EViews will assume “cov” and compute the Pearson covariance matrix. Note that `cov` is equivalent to the `cor` (p. 190) command with a different default setting.

Note that this view has been expanded considerably from EViews 5.1, but the syntax for the earlier version of the routine is fully compatible with the current version.

Pearson Correlation

cov	Product moment covariance.
corr	Product moment correlation.
sscp	Sums-of-squared cross-products.
stat	Test statistic (t -statistic) for evaluating whether the correlation is zero.
prob	Probability under the null for the test statistic.
cases	Number of cases.
obs	Number of observations.
wgts	Sum of the weights.

Spearman Rank Correlation

rcov	Spearman's rank covariance.
rcorr	Spearman's rank correlation.
rsscp	Sums-of-squared cross-products.
rstat	Test statistic (t -statistic) for evaluating whether the correlation is zero.
rprob	Probability under the null for the test statistic.
cases	Number of cases.
obs	Number of observations.
wgts	Sum of the weights.

Kendall's tau

taub	Kendall's tau-b.
taua	Kendall's tau-a.
taucd	Kendall's concordances and discordances.
taustat	Kendall's score statistic for evaluating whether the Kendall's tau-b measure is zero.
tauprob	Probability under the null for the score statistic.
cases	Number of cases.
obs	Number of observations.
wgts	Sum of the weights.

Uncentered Pearson

<code>ucov</code>	Product moment covariance.
<code>ucorr</code>	Product moment correlation.
<code>usscp</code>	Sums-of-squared cross-products.
<code>ustat</code>	Test statistic (<i>t</i> -statistic) for evaluating whether the correlation is zero.
<code>uprob</code>	Probability under the null for the test statistic.
<code>cases</code>	Number of cases.
<code>obs</code>	Number of observations.
<code>wgts</code>	Sum of the weights.

Note that `cases`, `obs`, and `wgts` are available for each of the methods.

Options

<code>wgt = name</code> (optional)	Name of series containing weights.
<code>wgtmethod = arg</code> (default = “sstdev”)	Weighting method (when weights are specified using “weight = ”): frequency (“freq”), inverse of variances (“var”), inverse of standard deviation (“stdev”), scaled inverse of variances (“svar”), scaled inverse of standard deviations (“sstdev”). Only applicable for ordinary (Pearson) calculations. Weights specified by “wgt = ” are frequency weights for rank correlation and Kendall’s tau calculations.
<code>pairwise</code>	Compute using pairwise deletion of observations with missing cases (pairwise samples).
<code>df</code>	Compute covariances with a degree-of-freedom correction for the mean (for centered specifications), and any partial conditioning variables.
<code>multi = arg</code> (default = “none”)	Adjustment to <i>p</i> -values for multiple comparisons: none (“none”), Bonferroni (“bonferroni”), Dunn-Sidak (“dunn”).
<code>outfmt = arg</code> (default = “single”)	Output format: single table (“single”), multiple table (“mult”), list (“list”), spreadsheet (“sheet”). Note that “outfmt = sheet” is only applicable if you specify a single statistic keyword.

`out = name` Basename for saving output. All results will be saved in Sym matrices named using keys (“COV”, “CORR”, “SSCP”, “TAUA”, “TAUB”, “CONC” (Kendall’s concurrences), “DISC” (Kendall’s discordances), “CASES”, “OBS”, “WGTS”) appended to the basename (e.g., the covariance specified by “out = my” is saved in the Sym matrix “MYCOV”).

`p` Print the result.

Examples

```
group grp1 height weight age
grp1.cov
```

displays a 3×3 covariance matrix for the three series in GRP1.

```
grp1.cov corr stat prob
```

displays a table containing the Pearson correlation, t -statistic for testing for zero correlation, and associated p -value, for the series in GRP1.

```
grp1.cov(pairwise) taub taustat tauprob
```

computes the Kendall’s tau-b, score statistic, and p -value for the score statistic, using samples with pairwise missing value exclusion.

```
grp1.cov(out=aa) cor @partial gender
```

computes the Pearson correlation for the series in GRP1 conditional on GENDER and saves the results in the symmetric matrix object AACORR.

Cross-references

See also [cor \(p. 190\)](#), [@cor \(p. 847\)](#), and [@cov \(p. 847\)](#).

cross	Group Views
--------------	-----------------------------

Displays cross correlations (correlograms) for a pair of series.

Syntax

```
group_name.cross(n,options)
```

You must specify the number of lags n to use in computing the cross correlations as the first option. Cross correlations will be computed for the first two series in the group.

Options

The following options may be specified inside the parentheses after the number of lags:

```
p          Print the cross correlogram.
```

Examples

```
group grp1 log(m1) dlog(cpi)
grp1.cross(36)
```

displays the cross correlogram between the log of M1 and the first difference of the log of CPI, using up to 36 leads and lags.

```
equation eq1.arch sp500 c
eq1.makeresids(s) res_std
group g1 res_std^2 res_std
g1.cross(24)
```

The first line estimates a GARCH(1,1) model and the second line retrieves the standardized residuals. The third line creates a group and the fourth line plots the cross correlogram squared standardized residual and the standardized residual, up to 24 leads and lags. This correlogram provides a rough check of asymmetry in the ARCH effect.

Cross-references

See [“Cross Correlations and Correlograms” on page 409](#) of the *User’s Guide I* for discussion.

displayname	Group Procs
-------------	-----------------------------

Display name for the group object.

Attaches a display name to a group object which may be used to label output in tables and graphs in place of the standard group object name.

Syntax

```
group_name.displayname display_name
```

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in group object names.

Examples

```
grp1.displayname Hours Worked
grp1.label
```

The first line attaches a display name “Hours Worked” to the group object GRP1, and the second line displays the label view of GRP1, including its display name.

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels and display names.

See also [Group::label](#) (p. 203).

drop	Group Procs
------	-----------------------------

Drops series from a group.

Syntax

```
group_name.drop ser1 [ser2 ser3 ...]
```

List the series to be dropped from the group object.

Examples

```
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.

Cross-references

See [“Groups” on page 84](#) of the *User’s Guide I* for additional discussion of groups.

See also [Group::add](#) (p. 185).

dtable	Group Views
--------	-----------------------------

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_name.dtable(options)
```

Options

p	Print the report table.
---	-------------------------

Examples

```
freeze(report) group1.dtable
```

freezes the dated 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 371 of the *User’s Guide I* 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.

freq	Group Views
------	-----------------------------

Compute frequency tables.

When used with a group containing a single series, `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, `freq` produces an *N*-way frequency tabulation for all of the series in the group.

Syntax

```
group_name.freq(options)
```

Options

Options common to both one-way and N-way frequency tables

dropna (default) / keepna	[Drop/Keep] NA as a category.
v = integer (default = 100)	Make bins if the number of distinct values or categories exceeds the specified number.
nov	Do not make bins on the basis of number of distinct values; ignored if you set “v = integer.”
a = number (default = 2)	Make bins if average count per distinct value is less than the specified number.
noa	Do not make bins on the basis of average count; ignored if you set “a = number.”
b = integer (default = 5)	Maximum number of categories to bin into.

n, obs, count (<i>default</i>)	Display frequency counts.
nocount	Do not display frequency counts.
p	Print the table.

Options for one-way tables

total (<i>default</i>) / nototal	[Display / Do not display] totals.
pct (<i>default</i>) / nopct	[Display / Do not display] percent frequencies.
cum (<i>default</i>) / nocum	(Display/Do not) display cumulative frequency counts/percentages.

Options for N-way tables

table (<i>default</i>)	Display in table mode.
list	Display in list mode.
rowm (<i>default</i>) / norowm	[Display / Do not display] row marginals.
colm (<i>default</i>) / nocolm	[Display / Do not display] column marginals.
tabm (<i>default</i>) / notabm	[Display / Do not display] table marginals—only for more than two series.
subm (<i>default</i>) / nosubm	[Display / Do not display] sub marginals—only for “l” option with more than two series.
full (<i>default</i>) / sparse	(Full/Sparse) tabulation in list display.
totpct / nototpct (<i>default</i>)	[Display / Do not display] percentages of total observations.
tabpct / notabpct (<i>default</i>)	[Display / Do not display] percentages of table observations—only for more than two series.
rowpct / norowpct (<i>default</i>)	[Display / Do not display] percentages of row total.
colpct / nocolpct (<i>default</i>)	[Display / Do not display] percentages of column total.

<code>exp / noexp</code> <i>(default)</i>	[Display / Do not display] expected counts under full independence.
<code>tabexp / notabexp</code> <i>(default)</i>	[Display / Do not display] expected counts under table independence—only for more than two series.
<code>test (default) / notest</code>	[Display / Do not display] tests of independence.

Examples

```
group g1 hrs
g1.freq(nov,noa)
```

tabulates each value (no binning) of HRS in ascending order with counts, percentages, and cumulatives.

```
group g2 inc
g2.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 [“One-Way Tabulation” on page 323](#) and [“N-Way Tabulation” on page 392](#) of the *User’s Guide I* for a discussion of frequency tables.

group	Group Declaration
--------------	-----------------------------------

Declare a group object containing a group of series.

Syntax

```
group group_name ser1 ser2 [ser3 ...]
```

Follow the group name with a list of series to be included in the group.

Examples

```
group g1 gdp cpi inv
group g1 tb3 m1 gov
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 [Group::add \(p. 185\)](#).

```
group rhs d1 d2 d3 d4 gdp(0 to -4)
ls cons rhs
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 [Chapter 12. “Groups,” on page 367](#) of the *User’s Guide I* for additional discussion.

See also [Group::add \(p. 185\)](#) and [Group::drop \(p. 199\)](#).

kerfit	Group Views
--------	-----------------------------

Scatterplot with bivariate kernel regression fit.

The `kerfit` command is no longer supported. See [scat \(p. 640\)](#).

label	Group Views Group Procs
-------	---

Display or change the label view of a group, including the last modified date and display name (if any).

As a procedure, `label` changes the fields in the group label.

Syntax

```
group_name.label
group_name.label(options) [text]
```

Options

The first version of the command displays the label view of the group. The second version may be used to modify the label. 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 <i>text</i> .
s	Sets the source field to <i>text</i> .

u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of G1 with “Data from CPS 1988 March File”:

```
g1.label(r)
g1.label(r) Data from CPS 1988 March File
```

To append additional remarks to G1, and then to print the label view:

```
g1.label(r) Log of hourly wage
g1.label(p)
```

To clear and then set the units field, use:

```
g1.label(u) Millions of bushels
```

Cross-references

See “Labeling Objects” on page 72 of the *User’s Guide I* for a discussion of labels.

See also `Group::displayname` (p. 198).

linefit	Group Views
---------	-----------------------------

Scatterplot with bivariate fit.

The `linefit` command is no longer supported. See `scat` (p. 640).

makecomp	Group Procs
----------	-----------------------------

Save the scores from a principal components analysis of the series in a group.

Syntax

```
group_name.makecomp(options) output_list
```

where the *output_list* is a list of names identifying the components to save. EViews will save the first *k* components corresponding to the *k* elements in *output_list*, up to the total number of series in the group.

Options

<code>scale = arg,</code> (<i>default</i> = “normload”)	Diagonal matrix scaling of the loadings and the scores: normalize loadings (“normload”), normalize scores (“normscores”), symmetric weighting (“symmetric”), user-specified (<i>arg</i> = <i>number</i>).
<code>cpnorm</code>	Compute the normalization for the score so that cross-products match the target (by default, EVIEWS chooses a normalization scale so that the moments of the scores match the target).
<code>eigval = vec_name</code>	Specify name of vector to hold the saved the eigenvalues in workfile.
<code>eigvec = mat_name</code>	Specify name of matrix to hold the save the eigenvectors in workfile.

Covariance Options

<code>cov = arg</code> (<i>default</i> = “cov”)	Covariance calculation method: ordinary (Pearson product moment) covariance (“cov”), ordinary correlation (“corr”), Spearman rank covariance (“rcov”), Spearman rank correlation (“rcorr”), uncentered ordinary correlation (“ucorr”). Note that Kendall’s tau measures are not valid methods.
<code>wgt = name</code> (optional)	Name of series containing weights.
<code>wgtmethod = arg</code> (<i>default</i> = “sstdev”)	Weighting method: frequency (“freq”), inverse of variances (“var”), inverse of standard deviation (“stdev”), scaled inverse of variances (“svar”), scaled inverse of standard deviations (“sstdev”). Only applicable for ordinary (Pearson) calculations where “weights = ” is specified. Weights for rank correlation and Kendall’s tau calculations are always frequency weights.
<code>pairwise</code>	Compute using pairwise deletion of observations with missing cases (pairwise samples).
<code>df</code>	Compute covariances with a degree-of-freedom correction accounting for the mean (for centered specifications) and any partial conditioning variables (for the default “cov = cov”, and the “cov = ucov” specifications). The default behavior in these cases is to perform no adjustment (e.g. – compute sample covariance dividing by n rather than $n - k$).

Examples

```
grp1.makecomp comp1 comp2 comp3
```

saves the first three principal components (in normalized loadings form) to the workfile. The components will have variances that are proportional to the eigenvalues.

```
grp1.makepcomp(scale=normscore) comp1 comp2 comp3
```

normalizes the scores so that the resulting series have variances that are equal to 1.

You may change the scaling for the normalized components so that the cross-products equal 1, using the `cpnorm` option:

```
grp1.makepcomp(scale=normscore, cpnorm) comp1 comp2 comp3
```

Cross-references

See “[Saving Component Scores,](#)” [beginning on page 403](#) of the *User’s Guide I* for further discussion.

nnfit	Group Views
--------------	-----------------------------

Scatterplot with bivariate nearest neighbor fit.

The `nnfit` command is no longer supported. See [scat](#) (p. 640).

pcomp	Group Views
--------------	-----------------------------

Principal components analysis.

Syntax

```
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

<code>cor</code> (<i>default</i>)	Use sample correlation matrix.
<code>cov</code>	Use sample covariance matrix.
<code>dof</code>	Degrees of freedom adjustment if “cov” option used. Default is no adjustment (compute sample covariance dividing by n rather than $n - 1$).

<code>eigval = vec_name</code>	Specify name of vector to hold the saved the eigenvalues in workfile.
<code>eigvec = mat_name</code>	Specify name of matrix to hold the save the eigenvectors in workfile.
<code>p</code>	Print results.

Examples

```
group g1 x1 x2 x3 x4
freeze(tab1) g1.pcomp(cor, eigval=v1, eigvec=m1) pc1 pc2
```

The first line creates a group named G1 containing the four series X1, X2, X3, X4. The second line stores the first two principal components of the sample correlation matrix in series named PC1 and PC2. 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 397](#) of the *User’s Guide I* for further discussion.

resample	Group Procs
-----------------	-----------------------------

Resample from observations in a group.

Syntax

```
group_name.resample(options) [output_spec]
```

You should follow the `resample` keyword and options with a list of names or a 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

<code>outsmpl = smpl_spec</code>	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.
<code>name = group_name</code>	Name of group to hold created series.
<code>permute</code>	Draw from rows without replacement. Default is to draw with replacement.
<code>weight = series_name</code>	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.

<code>block = integer</code>	Block length for each draw. Must be a positive integer. The default block length is 1.
<code>withna (default)</code>	[Draw / Do not draw] from all rows in the current sample, including those with NAs.
<code>dropna</code>	Do not draw from rows that contain missing values in the current workfile sample.
<code>fixna</code>	Excludes NAs from draws but copies rows containing missing values to the output series.

- Since we append a suffix for the new name, you may not use groups that contain auto-series. For example, resampling from a group containing the series `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 group name you provide already exists and is a group object, the group object will be overwritten. If the object 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 rows containing missing values and the output sample is adjusted so as 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

```
group g1 x y
g1.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 over-

written 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.resample(weight=wt,suffix=_2) g2
```

will append “_2” to the names for the new series, and will create a group objected named G2 containing these series. 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 “Resample” on page 337 of the *User’s Guide I* for a discussion of the resampling procedure. For additional discussion of wildcards, see Appendix C. “Wildcards,” on page 775 of the *User’s Guide I*.

See also @resample (p. 860) and @permute (p. 859) for sampling from matrices.

setformat	Group Procs
-----------	-----------------------------

Set the display format for cells in a group spreadsheet view.

Syntax

```
group_name.setformat(col_range) 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 *col_range* option is used to describe the columns to be updated in groups. It 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[:] <i>last_col</i>	Colon delimited range of columns (from low to high, e.g., “3:5”). Apply to all series corresponding to the column range.
first_series[:] <i>last_ser</i> <i>ies</i>	Colon delimited range of columns (from low to high, e.g., “series01:series05”) specified by the series names. Apply to all series corresponding to the column range.

To format numeric values, you should use one of the following format specifications:

g[.precision]	significant digits
f[.precision]	fixed decimal places

<i>c[.precision]</i>	fixed characters
<i>e[.precision]</i>	scientific/float
<i>p[.precision]</i>	percentage
<i>r[.precision]</i>	fraction

In order to set a format that groups digits into thousands, place a “t” after a specified format. For example, to obtain a fixed number of decimal places, with commas used to separate thousands, use “ft[.precision]”. To obtain a fixed number of characters with a period used to separate thousands, use “ct[.precision]”.

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 [“Date Formats” on page 707](#) of the *User’s Guide I*). 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 given by:

WF	(uses current EViews workfile date display format)
YYYY	“2002”
YYYY-Mon	“2002-Jan”
YYYYMon	“2002 Jan”
YYYY[M]MM	“2002[M]01”
YYYY:MM	“2002:01”
YYYY[Q]Q	“2002[Q]1”
YYYY:Q	“2002:Q
YYYY[S]S	“2002[S]1” (semi-annual)
YYYY:S	“2002:1”
YYYY-MM-DD	“2002-01-07”
YYYY Mon dd	“2002 Jan 7”
YYYY Month dd	“2002 January 7”
YYYY-MM-DD HH:MI	“2002-01-07 22:40”
YYYY-MM-DD HH:MI:SS	“2002-01-07 22:40:30”
YYYY-MM-DD HH:MI:SS.SSS	“2002-01-07 22:40:30.125”
Mon-YYYY	“Jan-2002”
Mon dd YYYY	“Jan 7 2002”
Mon dd, YYYY	“Jan 7, 2002”

Month dd YYYY	“January 7 2002”
Month dd, YYYY	“January 7, 2002”
MM/DD/YYYY	“01/07/2002”
mm/DD/YYYY	“1/07/2002”
mm/DD/YYYY HH:MI	“1/07/2002 22:40”
mm/DD/YYYY HH:MI:SS	“1/07/2002 22:40:30”
mm/DD/YYYY HH:MI:SS.SSS	“1/07/2002 22:40:30.125”
mm/dd/YYYY	“1/7/2002”
mm/dd/YYYY HH:MI	“1/7/2002 22:40”
mm/dd/YYYY HH:MI:SS	“1/7/2002 22:40:30”
mm/dd/YYYY HH:MI:SS.SSS	“1/7/2002 22:40:30.125”
dd/MM/YYYY	“7/01/2002”
dd/mm/YYYY	“7/1/2002”
DD/MM/YYYY	“07/01/2002”
dd Mon YYYY	“7 Jan 2002”
dd Mon, YYYY	“7 Jan, 2002”
dd Month YYYY	“7 January 2002”
dd Month, YYYY	“7 January, 2002”
dd/MM/YYYY HH:MI	“7/01/2002 22:40”
dd/MM/YYYY HH:MI:SS	“7/01/2002 22:40:30”
dd/MM/YYYY HH:MI:SS.SSS	“7/01/2002 22:40:30.125”
dd/mm/YYYY hh:MI	“7/1/2002 22:40”
dd/mm/YYYY hh:MI:SS	“7/1/2002 22:40:30”
dd/mm/YYYY hh:MI:SS.SSS	“7/1/2002 22:40:30.125”
hm:MI am	“10:40 pm”
hm:MI:SS am	“10:40:30 pm”
hm:MI:SS.SSS am	“10:40:30.125 pm”
HH:MI	“22:40”
HH:MI:SS	“22:40:30”
HH:MI:SS.SSS	“22:40:30.125”
hh:MI	“22:40”
hh:MI:SS	“22:40:30”
hh:MI:SS.SSS	“22:40:30.125”

Note that the “hh” formats display 24-hour time without leading zeros. In our examples above, there is no difference between the “HH” and “hh” formats for 10 p.m.

Also note that all of the “YYYY” formats above may be displayed using two-digit year “YY” format.

Examples

To set the format for a series in a group, provide the column identifier and format:

```
group1.setformat(1) f.5
```

sets the first series in GROUP1 to fixed 5-digit precision.

```
group1.setformat(2) f(.7)
group1.setformat(c) e.5
```

sets the formats for the second and third series in the group.

You may use any of the date formats given above:

```
group1.setformat(2) YYYYMon
group1.setformat(d) "YYYY-MM-DD HH:MI:SS.SSS"
```

The column identifier may be the series names. Assuming we have a group which contains the series A1, C1, B2, A5, and H2, in that order,

```
group1.setformat(c1:a5) p.3
```

sets the formats of the series C1, B2, and A5.

Cross-references

See [Group::setwidth \(p. 214\)](#), [Group::setindent \(p. 212\)](#) and [Group::setjust \(p. 213\)](#) for details on setting spreadsheet widths, indentation and justification.

setindent	Group Procs
-----------	-----------------------------

Set the display indentation for cells in a group object spreadsheet view.

Syntax

```
group_name.setindent(col_range) 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 spreadsheet (the EViews spreadsheet default font at the time the spreadsheet 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 spreadsheet view is created.

The *col_range* option is used to describe the columns to be updated. See [Group::setformat \(p. 209\)](#) for the syntax for *col_range* specifications.

Examples

To set the justification, 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.

Cross-references

See [Group::setWidth \(p. 214\)](#) and [Group::setjust \(p. 213\)](#) for details on setting spreadsheet widths and justification.

setjust	Group Procs
---------	-----------------------------

Set the display justification for cells in a group object spreadsheet view.

Syntax

```
group_name.setjust(col_range) 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. See [Group::setformat \(p. 209\)](#) for the syntax for *col_range* specifications.

The *format_arg* may be formed using the following:

top / middle / bottom]	Vertical justification setting.
auto / left / cen- ter / 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 taken from the Global Defaults for spreadsheet views.

Examples

To set the justification, 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.

Cross-references

See [Group::setwidth](#) (p. 214) and [Group::setindent](#) (p. 212) for details on setting spreadsheet widths and indentation.

setwidth	Group Procs
----------	-----------------------------

Set the column width for selected columns in a group spreadsheet.

Syntax

```
group_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 spreadsheet (the EViews spreadsheet default font at the time the spreadsheet 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

```
gr1.setwidth(2) 12
```

sets the width of column 2 to 12 width units.

```
gr1.setwidth(2:10) 20
```

sets the widths for columns 2 through 10 to 20 width units.

Cross-references

See [Group::setindent](#) (p. 212) and [Group::setjust](#) (p. 213) for details on setting spreadsheet indentation and justification.

sheet	Group Views
-------	-----------------------------

Spreadsheet view of a group object.

Syntax

```
group_name.sheet(options)
```

Options

p	Print the spreadsheet view.
---	-----------------------------

Examples

```
g1.sheet(p)
```

displays and prints the spreadsheet view of the group G1.

Cross-references

See [Chapter 5. “Basic Data Handling,” on page 77](#) of the *User’s Guide I* for a discussion of the spreadsheet view of series and groups.

sort	Group Procs
------	-----------------------------

Change display order for group spreadsheet.

The `sort` command changes the sort order settings for spreadsheet display of the group.

Syntax

```
group_name.sort(series1[, series2, series3])
```

Follow the keyword with a list of the series you wish to use to determine display order. You may specify up to three series for sorting. If you list two or more series, `sort` uses the values of the second series to resolve ties in the first series, and values of the third series to resolve ties in the first and second. By default, EViews will sort in ascending order. For purposes of sorting, NAs are considered to be smaller than any other value.

The series may be specified using the name or index of a series in the group. For example, if you provide the integer “2”, EViews will use the second series. To sort by the original work-file observation order, use the integer “0”, or the keyword “obs”.

To sort in descending order, precede the series name or index with a minus sign (“-”).

Examples

```
gr1.sort(x,y)
```

change the display order for group GR1, sorting by the series X and Y, with ties in X resolved using Y.

If X is the first series in group GR1 and Y is the second series,

```
gr1.sort(1,-2)
```

sorts first in ascending order by X and then in descending order by Y.

```
gr1.sort(obs)
```

returns the display order for group GR1 to the original (by observation).

Cross-references

See [“Spreadsheet” on page 368](#) of the *User’s Guide II* for additional discussion.

stats	Group Views
-------	-----------------------------

Descriptive statistics.

Computes and displays a table of means, medians, maximum and minimum values, standard deviations, and other descriptive statistics of a group of series.

Syntax

```
group_name.stats(options)
```

Options

i	Individual sample for each series. By default, EViews computes the statistics using a common sample.
p	Print the stats table.

Examples

```
group group1 wage hrs edu
group1.stats(i)
```

displays the descriptive statistics view of GROUP1 for the individual samples.

Cross-references

See [“Descriptive Statistics” on page 379](#) of the *User’s Guide I* for a discussion of the descriptive statistics views of a group.

See also [boxplot \(p. 613\)](#).

stom	Group Procs
------	-----------------------------

Convert a group to a matrix.

Fills a matrix with the data from a group.

Syntax

```
stom(group_name, matrix[, sample])
```

Include the group name in parentheses followed by a comma and the matrix name. By default, the series values in the current workfile sample are used to fill the 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 matrix (for alternative behavior, see [Group::stomna \(p. 217\)](#)).
- If the matrix already exists in the workfile, EViews automatically resizes the matrix to fit the group.

Examples

```
group g1 x y z
stom(g1,mat1)
```

converts the group G1 into a matrix named MAT1 using the current workfile sample. Any NAs in G1 will be dropped from MAT1.

```
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 [Chapter 18. “Matrix Language,” on page 627](#) of the *User’s Guide I* for further discussion and examples of matrices.

See also [Group::stomna \(p. 217\)](#) and [mtos \(p. 857\)](#).

stomna	Group Procs
--------	-----------------------------

Convert a group to a matrix without dropping NAs.

Fills a matrix with the data from a group without dropping observations with missing values.

Works in identical fashion to [Group::stom \(p. 216\)](#), but does not drop observations containing NAs.

Syntax

```
stomna(group, matrix[, sample])
```

Include the group name in parentheses, followed by a comma and the matrix name. By default, the series values in the current workfile sample are used to fill the matrix. You may optionally provide an alternative sample.

Examples

```
group g1 x y z
stomna(g1,mat1)
```

converts the group G1 into a matrix named MAT1 using the current workfile sample. Any NAs in G1 will be placed in MAT1.

```
group rhs x1 x2 x3
sample s1 1951 1990
stomna(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 always have 40 rows and 3 columns.

Cross-references

See [Chapter 18. “Matrix Language,” on page 627](#) of the *User’s Guide I* for further discussion and examples of matrices.

See also [Group::stom](#) (p. 216) and [mtos](#) (p. 857).

testbtw	Group Views
---------	-----------------------------

Test equality of the mean, median or variance between (among) series in a group.

Syntax

```
group_name.testbtw(options)
```

Specify the type of test as an option.

Options

mean (default)	Test equality of mean.
med	Test equality of median.
var	Test equality of variance.
c	Use common sample.
i (default)	Use individual sample.
p	Print the test results.

Examples

```
group g1 wage_m wage_f
g1.testbtw
g1.testbtw(var,c)
```

tests the equality of means between the two series WAGE_M and WAGE_F.

Cross-references

See “[Tests of Equality](#)” on page 395 of the *User's Guide I* for further discussion of these tests.

See also [Series::testby](#) (p. 391), [Series::teststat](#) (p. 392).

uroot	Group Views
-------	-----------------------------

Carries out (panel) unit root tests on a group of series.

When used on a group of series, the procedure will perform panel unit root testing. 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.

Note that simulation evidence suggests that in various settings (for example, small T), Hadri's panel unit root test experiences significant size distortion in the presence of autocorrelation when there is no unit root. In particular, the Hadri test appears to over-reject the null of stationarity, and may yield results that directly contradict those obtained using alternative test statistics (see Hlouskova and Wagner (2006) for discussion and details).

Syntax

```
group_name.uroot(options)
```

Options

Basic Specification Options

You should specify the exogenous variables and order of dependent variable differencing in the test equation using the following options:

const (<i>default</i>)	Include a constant in the test equation.
trend	Include a constant and a linear time trend in the test equation.
none	Do not include a constant or time trend (only available for the ADF and PP tests).
dif = <i>integer</i> (<i>default</i> = 0)	Order of differencing of the series prior to running the test. Valid values are {0, 1, 2}.

You may use one of the following keywords to specify the test:

sum (<i>default</i>)	Summary of the first five panel unit root tests (where applicable).
llc	Levin, Lin, and Chu.
breit	Breitung.
ips	Im, Pesaran, and Shin.
adf	Fisher - ADF.
pp	Fisher - PP.
hadri	Hadri.

Panel Specification Options

The following additional panel specific options are available:

balance	Use balanced (across cross-sections or series) data when performing test.
hac = <i>arg</i> (<i>default</i> = “bt”)	Method of estimating the frequency zero spectrum: “bt” (Bartlett kernel), “pr” (Parzen kernel), “qs” (Quadratic Spectral kernel). Applicable to “Summary”, LLC, Fisher-PP, and Hadri tests.
band = <i>arg</i> , b = <i>arg</i> (<i>default</i> = “nw”)	Method of selecting the bandwidth: “nw” (Newey-West automatic variable bandwidth selection), “a” (Andrews automatic selection), <i>number</i> (user-specified common bandwidth), <i>vector_name</i> (user-specified individual bandwidth). Applicable to “Summary”, LLC, Fisher-PP, and Hadri tests.

`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,

$$default = \begin{cases} 1 & \text{if } (T_{\min} \leq 60) \\ 2 & \text{if } (60 < T_{\min} \leq 100) \\ 4 & \text{if } (T_{\min} > 100) \end{cases}$$

where T_{\min} is the length of the shortest cross-section or series, otherwise *default* = “a”.

Applicable to “Summary”, LLC, Breitung, IPS, and Fisher-ADF tests.

`info = arg`
(*default* = “sic”) Information criterion to use when computing automatic lag length selection: “aic” (Akaike), “sic” (Schwarz), “hqc” (Hannan-Quinn).
Applicable to “Summary”, LLC, Breitung, IPS, and Fisher-ADF tests.

`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)

$$default = \text{int}(\min_i(12, T_i/3) \cdot (T_i/100)^{0.25})$$

where T_i is the length of the cross-section or series.

Other options

`p` Print output from the test.

Examples

The command:

```
Grp1.root(llc,trend)
```

performs the LLC panel unit root test with exogenous individual trends and individual effects on series in GRP1.

```
Gp2.uroot(IPS,const,maxlag=4,info=AIC)
```

performs the IPS panel unit root test on series in group GP2. The test includes individual effects, lag will be chosen by AIC from maximum lag of three.

```
Gp3.uroot(sum,const,lag=3,hac=pr,b=2.3)
```

performs a summary of the panel unit root tests on the series in group GP3. The test equation includes a constant term and three lagged first-difference terms. The frequency zero spectrum is estimated using kernel methods (with a Parzen kernel), and a bandwidth of 2.3.

Cross-references

See [“Unit Root Tests” on page 88](#) of the *User’s Guide II* for discussion of standard unit root tests performed on a single series, and [“Panel Unit Root Tests” on page 100](#) of the *User’s Guide II* for discussion of unit roots tests performed on panel structured workfiles, groups of series, or pooled data.

References

- MacKinnon, James G., Alfred A. Haug, and Leo Michelis (1999), “Numerical Distribution Functions of Likelihood Ratio Tests For Cointegration,” *Journal of Applied Econometrics*, 14, 563-577.
- Osterwald-Lenum, Michael (1992). “A Note with Quantiles of the Asymptotic Distribution of the Maximum Likelihood Cointegration Rank Test Statistics,” *Oxford Bulletin of Economics and Statistics*, 54, 461-472.

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. 355) or “Alpha” (p. 5) objects.

Link Declaration

[link](#).....link object declaration (p. 225).

To declare a link object, enter the keyword `link`, followed by a name:

```
link newser
```

and an optional link specification:

```
link altser.linkto(c=obs,nacat) indiv::x @src ind1 ind2 @dest ind1  
ind2
```

Link Views

[label](#)label information for the link (p. 224).

Link Procs

[displayname](#).....set display name (p. 223).

[linkto](#).....specify link object definition (p. 226).

Link Entries

The following section provides an alphabetical listing of the commands associated with the “Link” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

displayname	Link Procs
-------------	----------------------------

Display names for a link object.

Attaches a display name to a link object which may be used to label output in tables and graphs in place of the standard link object name.

Syntax

```
link_name.displayname display_name
```

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in link object names.

Examples

```
hrs.displayname Hours Worked
```

```
hrs.label
```

The first line attaches a display name “Hours Worked” to the link 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 link object GDP. The line graph view of GDP from the second line will use the display name as the legend.

Cross-references

See “[Labeling Objects](#)” on page 72 of the *User’s Guide I* for a discussion of labels and display names.

See also [Link::label](#) (p. 224) and [Graph::legend](#) (p. 161).

label	Link Views Link Procs
-------	---

Display or change the label view of the link object, including the last modified date and display name (if any).

As a procedure, `label` changes the fields in the link object label.

Syntax

```
link_name.label
link_name.label(options) [text]
```

Options

The first version of the command displays the label view of the link. The second version may be used to modify the label. 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 <i>text</i> .
s	Sets the source field to <i>text</i> .
u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of the link object LWAGE with “Data from CPS 1988 March File”:

```
lwage.label(r)
lwage.label(r) Data from CPS 1988 March File
```

To append additional remarks to LWAGE, and then to print the label view:

```
lwage.label(r) Log of hourly wage
lwage.label(p)
```

To clear and then set the units field, use:

```
lwage.label(u) Millions of bushels
```

Cross-references

See “[Labeling Objects](#)” on page 72 of the *User’s Guide I* for a discussion of labels.

See also [Link::displayname](#) (p. 223).

link	Link Declaration
-------------	----------------------------------

Create a series link object.

Declares a link object which may be used to refer to data in a series contained in a different workfile page. Links are used to create automatically updating match merges using identifier series or using dates (frequency conversion).

Syntax

```
link link_name
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 [Link::linkto](#) (p. 226) 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 ind2
```

combines the link declaration with the link specification step.

Cross-references

For a discussion of linking, see [Chapter 8. “Series Links,” on page 173](#) of the *User’s Guide I*.

See also [Link::linkto](#) (p. 226) and [unlink](#) (p. 796).

linkto	Link Procs
--------	----------------------------

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

```
link_name.linkto(options) source_page\series_name [src_id dest_id]
link_name.linkto(options) source_page\series_name [@src src_ids @dest dest_ids]
```

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 date structured 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 uses 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.
- Frequency conversion linking uses special handling for panel structured pages. Links involving panel pages first perform a mean contraction in the source page, if necessary, then a frequency conversion to the destination page, then an expansion in the destination, if necessary.

Options

General Match Merge Link Options

The following options are available when linking with general match merging:

<code>smp1 = smp1_spec</code>	Sample to be used when computing contractions in a link by match merge. Either provide the sample range in double quotes or specify a named sample object. By default, EViews will use the entire workfile sample “@ALL”.
<code>c = arg</code>	<p>Set the match merge contraction or the frequency conversion method.</p> <p>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).</p> <p>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”.</p> <p>If you are linking by frequency conversion, you may use this argument to specify the up- or down-conversion method using the options found in fetch (p. 717). The default frequency conversion methods are taken from the series defaults.</p>
<code>quant = number</code>	Quantile value to be used when contracting using the “c = quant” option (e.g., “quant = .3”).
<code>nacat</code>	Treat “NA” values as a category when performing link by general match merge operations.

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.

Frequency Conversion Link Options

If the `linkto` command does not specify identifier series, EViews will link series data using frequency conversion where appropriate.

The following options control the frequency conversion method when creating a frequency conversion link, converting from *low* to *high* frequency:

<code>c = arg</code>	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 creating a frequency conversion link, converting from *high* to *low* frequency:

<code>c = arg</code>	<p><i>High to low conversion methods removing NAs:</i> “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).</p> <p><i>High to low conversion methods propagating NAs:</i> “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).</p>
----------------------	--

Note that if no conversion method is specified, the series specific default conversion method or the global settings will be employed.

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 (BIRTH-STATE).

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:

```
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.

```
link l1.linkto(c=obs,nacat) indiv\x @src ind1 ind2 @dest ind1 ind2
```

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:

```
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:

```
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 matching 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:

```
link gdp
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:

```
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 detailed discussion of linking, see [Chapter 8. “Series Links,”](#) on [page 173](#) of the *User’s Guide I*.

See also [Link::link](#) (p. 225), [unlink](#) (p. 796), and [copy](#) (p. 696).

Logl

Likelihood object. Used for performing maximum likelihood estimation of user-specified likelihood functions.

Logl Declaration

logllikelihood object declaration (p. 240).

To declare a logl object, use the `logl` keyword, followed by a name to be given to the object.

Logl Method

mlmaximum likelihood estimation (p. 242).

Logl Views

appendadd line to the specification (p. 235).

cellipseConfidence ellipses for coefficient restrictions (p. 235).

checkderivscompare user supplied and numeric derivatives (p. 236).

coefcovcoefficient covariance matrix (p. 237).

gradsexamine the gradients of the log likelihood (p. 238).

labellabel view of likelihood object (p. 239).

outputtable of estimation results (p. 243).

resultsestimation results (p. 243).

speclikelihood specification (p. 244).

waldWald coefficient restriction test (p. 245).

Logl Procs

displaynameset display name (p. 238).

makegradsmake group containing gradients of the log likelihood (p. 241).

makemodelmake model (p. 241).

updatecoefsupdate coefficient vector(s) from likelihood (p. 244).

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 32. “The Log Likelihood \(LogL\) Object,” on page 283](#) of the *User’s Guide II* for further discussion.

@byeqnevaluate specification by equation.

@byobsevaluate specification by observation (default).

@derivspecify an analytic derivative series.

@derivstepset parameters to control step size.

@loglspecify the likelihood contribution series.

@paramset starting values.

`@temp` remove temporary working series.

Logl Data Members

Scalar Values (system data)

`@aic` Akaike information criterion.
`@cofcov(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

`@cofcov` 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.@cofcov
```

Logl Entries

The following section provides an alphabetical listing of the commands associated with the “Logl” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

append	Logl Procs
---------------	----------------------------

Append a specification line to a logl.

Syntax

`logl_name.append text`

Type the text to be added after the `append` keyword.

Examples

```
logl ll1
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
```

declares a logl object called LL1, and then appends a specification that estimates an ordinary least squares model.

cellipse	Logl Views
-----------------	----------------------------

Confidence ellipses for coefficient restrictions.

The `cellipse` view displays confidence ellipses for pairs of coefficient restrictions for an estimation object.

Syntax

`logl_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

<code>ind = arg</code>	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.
<code>size = number</code> (<i>default</i> = 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.
<code>dist = arg</code>	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 χ^2 distribution.
<code>p</code>	Print the graph.

Examples

The two commands:

```
log1.ellipse c(1), c(2), c(3)
log1.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).

```
log1.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 [“Confidence Ellipses” on page 142](#) of the *User’s Guide II* for discussion.

See also [Log1:wald \(p. 245\)](#).

checkderivs	Log1 Views
-------------	----------------------------

Check derivatives of likelihood object.

Displays a table containing information on numeric derivatives and, if available, the user-supplied analytic derivatives.

Syntax

```
logl_name.checkderiv(options)
```

Options

p	Print the table of results.
---	-----------------------------

Examples

```
ll1.checkderiv
```

displays a table that evaluates the numeric derivatives of the logl object LL1.

Cross-references

See [Chapter 32. “The Log Likelihood \(LogL\) Object,” on page 283](#) of the *User’s Guide II* for a general discussion of the likelihood object and the “@deriv” statement.

See also [Logl::grads \(p. 238\)](#) and [Logl::makegrads \(p. 241\)](#).

coefcov	Logl Views
---------	----------------------------

Coefficient covariance matrix.

Displays the covariances of the coefficient estimates for an estimated likelihood object.

Syntax

```
logl_name.coefcov(options)
```

Options

p	Print the coefficient covariance matrix.
---	--

Examples

```
ll2.coefcov
```

displays the coefficient covariance matrix for the likelihood object LL2 in a window. To store the coefficient covariance matrix as a sym object, use “@coefcov”:

```
sym eqcov = ll2.@coefcov
```

Cross-references

See also [Coef::coef \(p. 16\)](#) and [Logl::spec \(p. 244\)](#).

displayname	Logl Procs
-------------	----------------------------

Display names for likelihood objects.

Attaches a display name to a likelihood object which may be used to label output in place of the standard object name.

Syntax

```
logl_name.displayname display_name
```

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in likelihood object names.

Examples

```
logl.displayname Hours Worked
logl.label
```

The first line attaches a display name “Hours Worked” to the likelihood object LG1, and the second line displays the label view of LG1, including its display name.

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels and display names.

See also [Logl::label \(p. 239\)](#).

grads	Logl Views
-------	----------------------------

Gradients of the objective function.

Displays the gradients of the objective function (where available) for an estimated likelihood object.

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. Evaluating the gradients at current coefficient values allows you to examine 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

```
logl_name.grads(options)
```

Options

<code>g</code>	Display multiple graph showing the gradients of the objective function with respect to the coefficients evaluated at each observation.
<code>t</code> (<i>default</i>)	Display spreadsheet view of the values of the gradients of the objective function with respect to the coefficients evaluated at each observation.
<code>p</code>	Print results.

Examples

To show a summary view of the gradients:

```
l12.grads
```

To display and print the table view:

```
l12.grads(t, p)
```

Cross-references

See also [Logl::makegrads](#) (p. 241).

label	Logl Views Logl Procs
--------------	---

Display or change the label view of likelihood object, including the last modified date and display name (if any).

As a procedure, `label` changes the fields in the likelihood object label.

Syntax

```
logl_name.label
```

```
logl_name.label(options) [text]
```

Options

The first version of the command displays the label view of the likelihood object. The second version may be used to modify the label. Specify one of the following options along with optional text. If there is no text provided, the specified field will be cleared.

<code>c</code>	Clears all text fields in the label.
<code>d</code>	Sets the description field to <i>text</i> .
<code>s</code>	Sets the source field to <i>text</i> .

u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of the logl object L2 with “Data from CPS 1988 March File”:

```
l2.label(r)
l2.label(r) Data from CPS 1988 March File
```

To append additional remarks to L2, and then to print the label view:

```
l2.label(r) Log of hourly wage
l2.label(p)
```

To clear and then set the units field, use:

```
l2.label(u) Millions of bushels
```

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels.

See also [Logl::displayname \(p. 238\)](#).

logl	Logl Declaration
------	----------------------------------

Declare likelihood object.

Syntax

```
logl logl_name
```

Examples

```
logl ll1
```

declares a likelihood object named LL1.

```
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 32. “The Log Likelihood \(LogL\) Object,” on page 283](#) of the *User’s Guide II* for further examples of the use of the likelihood object.

See also [Logl::append \(p. 235\)](#) for adding specification lines to an existing likelihood object, and [Logl::ml \(p. 242\)](#) for estimation.

makegrads	Logl Procs
------------------	----------------------------

Make a group containing individual series which hold the gradients of the objective function.

Syntax

```
logl_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

<code>n = arg</code>	Name of group object to contain the series.
----------------------	---

Examples

```
l12.grads(n=out)
```

creates a group named OUT containing series named GRAD01, GRAD02, and GRAD03.

```
l12.grads(n=out) g1 g2 g3
```

creates the same group, but names the series G1, G2 and G3.

Cross-references

See also [Logl::grads \(p. 238\)](#).

makemodel	Logl Procs
------------------	----------------------------

Make a model from a likelihood object.

Syntax

```
logl_name.makemodel(name) assign_statement
```

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

```
l13.makemodel(logmod) @prefix s_
```

makes a model named LOGMOD from the estimated logl object. LOGMOD includes an assignment statement “ASSIGN @PREFIX S_”. Use the command “show logmod” or “logmod.spec” to open the LOGMOD window.

Cross-references

See [Chapter 36. “Models,” on page 407](#) of the *User’s Guide II* for a discussion of specifying and solving models in EViews.

See also [Logl::append \(p. 235\)](#), [Model::merge \(p. 283\)](#) and [Model::solve \(p. 287\)](#).

ml	Logl Method
----	-----------------------------

Maximum likelihood estimation of logl models.

Syntax

```
logl_name.ml(options)
```

Options

b	Use 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 logl object BVAR by maximum likelihood.

Cross-references

See [Chapter 32. “The Log Likelihood \(LogL\) Object,” on page 283](#) of the *User’s Guide II* for a discussion of user specified likelihood models.

output	Logl Views
---------------	----------------------------

Display estimation output.

`output` changes the default object view to display the estimation output (equivalent to using [Logl::results \(p. 243\)](#)).

Syntax

`logl_name.output`

Options

<code>p</code>	Print estimation output for estimation object
----------------	---

Examples

The `output` keyword may be used to change the default view of an estimation object. Entering the command:

```
log2.output
```

displays the estimation output for likelihood object LOG2.

Cross-references

See [Logl::results \(p. 243\)](#).

results	Logl Views
----------------	----------------------------

Displays the results view of an estimated likelihood object.

Syntax

`logl_name.results(options)`

Options

<code>p</code>	Print the view.
----------------	-----------------

Examples

```
l11.results(p)
```

prints the estimation results from the estimated logl, LL1.

spec	Logl Views
-------------	----------------------------

Display the text specification view for logl objects.

Syntax

`logl_name.spec(options)`

Options

<code>p</code>	Print the specification text.
----------------	-------------------------------

Examples

```
lg1.spec
```

displays the specification of the logl object LG1.

Cross-references

See also [Logl::append](#) (p. 235).

updatecoefs	Logl Procs
--------------------	----------------------------

Update coefficient object values from likelihood object.

Copies coefficients from the likelihood object into the appropriate coefficient vector or vectors.

Syntax

`logl_name.updatecoefs`

Follow the name of the likelihood object by a period and the keyword `updatecoefs`.

Examples

```
ll1.updatecoefs
```

places the coefficients from LL1 in the default coefficient vector C.

Cross-references

See also [Coef::coef](#) (p. 16).

wald	Logl Views
------	----------------------------

Wald coefficient restriction test.

Syntax

`logl_name.wald restrictions`

Enter the likelihood object name, followed by a period, and the keyword. You must provide a list of the coefficient restrictions, with joint (multiple) coefficient restrictions separated by commas.

Options

p	Print the test results.
---	-------------------------

Examples

```
ll1.wald c(2)=0, c(3)=0
```

tests the null hypothesis that the second and third coefficients in LL1 are jointly zero.

Cross-references

See [“Wald Test \(Coefficient Restrictions\)” on page 145](#) of the *User’s Guide II* for a discussion of Wald tests.

See also [Logl::cellipse \(p. 235\)](#), [testdrop \(p. 790\)](#), [testadd \(p. 789\)](#).

Matrix

Matrix (two-dimensional array).

Matrix Declaration

[matrix](#).....declare matrix object (p. 258).

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

[cor](#).....correlation matrix by columns (p. 249).
[cov](#).....covariance matrix by columns (p. 252).
[label](#).....label information for the matrix (p. 257).
[pcomp](#).....Principal components analysis of the columns in a matrix (p. 259).
[sheet](#).....spreadsheet view of the matrix (p. 267).
[stats](#).....descriptive statistics by column (p. 267).

Matrix Graph Views

Graph creation types are discussed in detail in “Graph Creation Commands” on page 601.

[area](#).....area graph of the columns in the matrix (p. 603).
[band](#).....area band graph (p. 606).
[bar](#).....bar graph of each column (p. 609).
[boxplot](#).....boxplot of each column (p. 613).
[distplot](#).....distribution graph (p. 615).
[dot](#).....dot plot graph (p. 622).
[errbar](#).....error bar graph view (p. 626).
[hilo](#).....high-low(-open-close) chart (p. 628).
[line](#).....line graph of each column (p. 630).
[pie](#).....pie chart view (p. 633).
[qqplot](#).....quantile-quantile graph (p. 636).
[scat](#).....scatter diagrams of the columns of the matrix (p. 640).
[scatmat](#).....matrix of all pairwise scatter plots (p. 644).
[scatpair](#).....scatterplot pairs graph (p. 647).
[seasplot](#).....seasonal line graph of the columns of the matrix (p. 651).

spike spike graph (p. 652).
xyarea XY area graph (p. 656).
xybar XY bar graph (p. 659).
xyline XY line graph (p. 661).
xypair XY pairs graph (p. 664).

Matrix Procs

displayname set display name (p. 255).
fill fill the elements of the matrix (p. 256).
read import data from disk (p. 262).
setformat set the display format for the matrix spreadsheet (p. 264).
setindent set the indentation for the matrix spreadsheet (p. 265).
setjust set the justification for the matrix spreadsheet (p. 266).
setwidth set the column width in the matrix spreadsheet (p. 266).
write export data to disk (p. 268).

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,

```
matrix copymat=results
matrix covmat1=eql.@coefcov
matrix(5,2) count
count.fill 1,2,3,4,5,6,7,8,9,10
```

as does the equation procedure:

```
eql.makecoefcov covmat2
```

You can declare and initialize a matrix in one command:

```
matrix(10,30) results=3
matrix(5,5) other=results1
```

Graphs and covariances may be generated for the columns of the matrix,

```
copymat.line
copymat.cov
```

and statistics computed for the rows of a matrix:

```
matrix rowmat=@transpose(copymat)
rowmat.stats
```

You can use explicit indices to refer to matrix elements:

```
scalar diagsum=cov1(1,1)+cov1(2,2)+cov(3,3)
```

Matrix Entries

The following section provides an alphabetical listing of the commands associated with the “Matrix” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

cor	Matrix Views
------------	------------------------------

Compute measure of association for the columns in a matrix. You may compute measures related to Pearson product-moment (ordinary) correlations, rank correlations, or Kendall’s tau.

Syntax

```
matrix_name.cor(options) [keywords [@partial z1 z2 z3...]]
```

You should specify keywords indicating the statistics you wish to display from the list below, optionally followed by the keyword `@partial` and a list of conditioning series or groups (for the group view), or the name of a conditioning matrix (for the matrix view). In the matrix view setting, the columns of the matrix should contain the conditioning information, and the number of rows should match the original matrix.

You may specify keywords from one of the four sets (Pearson correlation, Spearman correlation, Kendall’s tau, Uncentered Pearson) corresponding the computational method you wish to employ. (You may not select keywords from more than one set.)

If you do not specify *keywords*, EViews will assume “corr” and compute the Pearson correlation matrix. Note that `cor` is equivalent to the `cov` (p. 252) command with a different default setting.

Note that this view has been expanded considerably from EViews 5.1, but the syntax for the earlier version of the routine is fully compatible with the current version.

Pearson Correlation

cov	Product moment covariance.
corr	Product moment correlation.
sscp	Sums-of-squared cross-products.
stat	Test statistic (<i>t</i> -statistic) for evaluating whether the correlation is zero.
prob	Probability under the null for the test statistic.

cases	Number of cases.
obs	Number of observations.
wgts	Sum of the weights.

Spearman Rank Correlation

rcov	Spearman's rank covariance.
rcorr	Spearman's rank correlation.
rsscp	Sums-of-squared cross-products.
rstat	Test statistic (t -statistic) for evaluating whether the correlation is zero.
rprob	Probability under the null for the test statistic.
cases	Number of cases.
obs	Number of observations.
wgts	Sum of the weights.

Kendall's tau

taub	Kendall's tau-b.
taua	Kendall's tau-a.
taucd	Kendall's concordances and discordances.
taustat	Kendall's score statistic for evaluating whether the Kendall's tau-b measure is zero.
tauprob	Probability under the null for the score statistic.
cases	Number of cases.
obs	Number of observations.
wgts	Sum of the weights.

Uncentered Pearson

ucov	Product moment covariance.
ucorr	Product moment correlation.
usscp	Sums-of-squared cross-products.
ustat	Test statistic (t -statistic) for evaluating whether the correlation is zero.
uprob	Probability under the null for the test statistic.

<code>cases</code>	Number of cases.
<code>obs</code>	Number of observations.
<code>wgts</code>	Sum of the weights.

Note that `cases`, `obs`, and `wgts` are available for each of the methods.

Options

<code>wgt = name</code> (optional)	Name of series containing weights.
<code>wgtmethod = arg</code> (default = “sstdev”)	Weighting method (when weights are specified using “weight = ”): frequency (“freq”), inverse of variances (“var”), inverse of standard deviation (“stdev”), scaled inverse of variances (“svar”), scaled inverse of standard deviations (“sstdev”). Only applicable for ordinary (Pearson) calculations. Weights specified by “wgt = ” are frequency weights for rank correlation and Kendall’s tau calculations.
<code>pairwise</code>	Compute using pairwise deletion of observations with missing cases (pairwise samples).
<code>df</code>	Compute covariances with a degree-of-freedom correction for the mean (for centered specifications), and any partial conditioning variables.
<code>multi = arg</code> (default = “none”)	Adjustment to p -values for multiple comparisons: none (“none”), Bonferroni (“bonferroni”), Dunn-Sidak (“dunn”).
<code>outfmt = arg</code> (default = “single”)	Output format: single table (“single”), multiple table (“mult”), list (“list”), spreadsheet (“sheet”). Note that “outfmt = sheet” is only applicable if you specify a single statistic keyword.
<code>out = name</code>	Baseline for saving output. All results will be saved in Sym matrices named using keys (“COV”, “CORR”, “SSCP”, “TAUA”, “TAUB”, “CONC” (Kendall’s concurrences), “DISC” (Kendall’s discordances), “CASES”, “OBS”, “WGTS”) appended to the basename (e.g., the covariance specified by “out = my” is saved in the Sym matrix “MYCOV”).
<code>p</code>	Print the result.

Examples

```
mat1.cor
```

displays a 3×3 Pearson correlation matrix for the columns series in MAT1.

```
mat1.cor corr stat prob
```

displays a table containing the Pearson correlation, t -statistic for testing for zero correlation, and associated p -value, for the columns in MAT1.

```
mat1.cor(pairwise) taub taustat tauprob
```

computes the Kendall's tau-b, score statistic, and p -value for the score statistic, using samples with pairwise missing value exclusion.

```
grp1.cor(out=aa) cov
```

computes the Pearson covariance for the columns in MAT1 and saves the results in the symmetric matrix object AACO.

Cross-references

See also [cov](#) (p. 252), [@cor](#) (p. 847), and [@cov](#) (p. 847).

COV	Matrix Views
------------	------------------------------

Compute measure of association for the columns in a matrix. You may compute measures related to Pearson product-moment (ordinary) covariances, rank covariances, or Kendall's tau.

Syntax

```
matrix_name.cov(options) [keywords [@partial z1 z2 z3...]]
```

You should specify keywords indicating the statistics you wish to display from the list below, optionally followed by the keyword `@partial` and a list of conditioning series or groups (for the group view), or the name of a conditioning matrix (for the matrix view). In the matrix view setting, the columns of the matrix should contain the conditioning information, and the number of rows should match the original matrix.

You may specify keywords from one of the four sets (Pearson correlation, Spearman correlation, Kendall's tau, Uncentered Pearson) corresponding the computational method you wish to employ. (You may not select keywords from more than one set.)

If you do not specify *keywords*, EViews will assume “cov” and compute the Pearson covariance matrix. Note that `cov` is equivalent to the [cor](#) (p. 249) command with a different default setting.

Note that this view has been expanded considerably from EViews 5.1, but the syntax for the earlier version of the routine is fully compatible with the current version.

Pearson Correlation

cov	Product moment covariance.
corr	Product moment correlation.
sscp	Sums-of-squared cross-products.
stat	Test statistic (t -statistic) for evaluating whether the correlation is zero.
prob	Probability under the null for the test statistic.
cases	Number of cases.
obs	Number of observations.
wgts	Sum of the weights.

Spearman Rank Correlation

rcov	Spearman's rank covariance.
rcorr	Spearman's rank correlation.
rsscp	Sums-of-squared cross-products.
rstat	Test statistic (t -statistic) for evaluating whether the correlation is zero.
rprob	Probability under the null for the test statistic.
cases	Number of cases.
obs	Number of observations.
wgts	Sum of the weights.

Kendall's tau

taub	Kendall's tau-b.
taua	Kendall's tau-a.
taucd	Kendall's concordances and discordances.
taustat	Kendall's score statistic for evaluating whether the Kendall's tau-b measure is zero.
tauprob	Probability under the null for the score statistic.
cases	Number of cases.
obs	Number of observations.
wgts	Sum of the weights.

Uncentered Pearson

<code>ucov</code>	Product moment covariance.
<code>ucorr</code>	Product moment correlation.
<code>usscp</code>	Sums-of-squared cross-products.
<code>ustat</code>	Test statistic (<i>t</i> -statistic) for evaluating whether the correlation is zero.
<code>uprob</code>	Probability under the null for the test statistic.
<code>cases</code>	Number of cases.
<code>obs</code>	Number of observations.
<code>wgts</code>	Sum of the weights.

Note that `cases`, `obs`, and `wgts` are available for each of the methods.

Options

<code>wgt = name</code> (optional)	Name of series containing weights.
<code>wgtmethod = arg</code> (default = “sstdev”)	Weighting method (when weights are specified using “weight = ”): frequency (“freq”), inverse of variances (“var”), inverse of standard deviation (“stdev”), scaled inverse of variances (“svar”), scaled inverse of standard deviations (“sstdev”). Only applicable for ordinary (Pearson) calculations. Weights specified by “wgt = ” are frequency weights for rank correlation and Kendall’s tau calculations.
<code>pairwise</code>	Compute using pairwise deletion of observations with missing cases (pairwise samples).
<code>df</code>	Compute covariances with a degree-of-freedom correction for the mean (for centered specifications), and any partial conditioning variables.
<code>multi = arg</code> (default = “none”)	Adjustment to <i>p</i> -values for multiple comparisons: none (“none”), Bonferroni (“bonferroni”), Dunn-Sidak (“dunn”).
<code>outfmt = arg</code> (default = “single”)	Output format: single table (“single”), multiple table (“mult”), list (“list”), spreadsheet (“sheet”). Note that “outfmt = sheet” is only applicable if you specify a single statistic keyword.

out = name

Basename for saving output. All results will be saved in Sym matrices named using keys (“COV”, “CORR”, “SSCP”, “TAUA”, “TAUB”, “CONC” (Kendall’s concurrences), “DISC” (Kendall’s discordances), “CASES”, “OBS”, “WGTS”) appended to the basename (e.g., the covariance specified by “out = my” is saved in the Sym matrix “MYCOV”).

p

Print the result.

Examples

```
mat1.cov
```

displays a 3×3 Pearson covariance matrix for the columns series in MAT1.

```
mat1.cov corr stat prob
```

displays a table containing the Pearson covariance, t -statistic for testing for zero correlation, and associated p -value, for the columns in MAT1.

```
mat1.cov(pairwise) taub taustat tauprob
```

computes the Kendall’s tau-b, score statistic, and p -value for the score statistic, using samples with pairwise missing value exclusion.

```
mat1.cov(out=aa) cov
```

computes the Pearson covariance for the columns in MAT1 and saves the results in the symmetric matrix object AACO.

Cross-references

See also [cor](#) (p. 249), [@cor](#) (p. 847), and [@cov](#) (p. 847).

displayname	Matrix Procs
-------------	------------------------------

Display names for matrix objects.

Attaches a display name to a matrix object which may be used to label output in place of the standard matrix object name.

Syntax

```
matrix_name.displayname display_name
```

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in matrix object names.

Examples

```
m1.displayname Hours Worked
```

```
m1.label
```

The first line attaches a display name “Hours Worked” to the matrix object M1, and the second line displays the label view of M1, including its display name.

Cross-references

See “[Labeling Objects](#)” on page 72 of the *User’s Guide I* for a discussion of labels and display names.

See also `Matrix::label` (p. 257).

fill	Matrix Procs
------	------------------------------

Fill a matrix object with specified values.

Syntax

```
matrix_name.fill(options) n1[, n2, n3 ...]
```

Follow the keyword with a list of values to place in the matrix object. *Each value should be separated by a comma.*

Running out of values before the object is completely filled is not an error; the remaining cells or observations will be unaffected, unless the “l” option is specified. If, however, you list more values than the object can hold, EViews will not modify any observations and will return an error message.

Options

l	Loop repeatedly over the list of values as many times as it takes to fill the object.
o = integer (default = 1)	Fill the object from the specified element. Default is the first element.
b = arg (default = “c”)	Matrix fill order: “c” (fill the matrix by column), “r” (fill the matrix by row).

Examples

The commands,

```
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:

$$m1 = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}, \quad m2 = \begin{bmatrix} 1 & 0 \\ 1 & 2 \end{bmatrix} \tag{1.1}$$

Cross-references

See [Chapter 18. “Matrix Language,” on page 627](#) of the *User’s Guide I* for a detailed discussion of vector and matrix manipulation in EViews.

label	Matrix Views Matrix Procs
-------	---

Display or change the label view of a matrix, including the last modified date and display name (if any).

As a procedure, `label` changes the fields in the matrix label.

Syntax

```
matrix_name.label
matrix_name.label(options) [text]
```

Options

The first version of the command displays the label view of the matrix. The second version may be used to modify the label. 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 <i>text</i> .
s	Sets the source field to <i>text</i> .
u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of M1 with “Data from CPS 1988 March File”:

```
m1.label(r)
m1.label(r) Data from CPS 1988 March File
```

To append additional remarks to M1, and then to print the label view:

```
m1.label(r) Log of hourly wage
m1.label(p)
```

To clear and then set the units field, use:

```
ml.label(u) Millions of bushels
```

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels.

See also [Matrix::displayname](#) (p. 255).

matrix	Matrix Declaration
--------	------------------------------------

Declare and optionally initializes a matrix object.

Syntax

```
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. Once declared, matrices may be resized by repeating the `matrix` command using the original name.

You may combine matrix declaration and assignment. If there is no assignment statement, the matrix will initially be filled with zeros.

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 [Chapter 18. “Matrix Language,” beginning on page 627](#) of the *User’s Guide I* for further discussion.

See [“Rowvector” \(p. 337\)](#) and [“Vector” \(p. 575\)](#) and [“Sym” \(p. 453\)](#) for full descriptions of the various matrix objects.

pcomp	Matrix Views
-------	------------------------------

Principal components analysis of the columns in a matrix.

Syntax

There are two forms of the `pcomp` command. The first form, which applies when displaying eigenvalue table output or graphs of the ordered eigenvalues, has only options and no command argument.

```
matrix_name.pcomp(options)
```

The second form, which applies to the graphs of component loadings, component scores, and biplots, uses the optional argument to determine which components to plot. In this form:

```
matrix_name.pcomp(options) [graph_list]
```

where the `[graph_list]` is an optional list of integers and/or vectors containing integers identifying the components to plot. Multiple pairs are handled using the method specified in the “mult = ” option.

If the list of component indices omitted, EViews will plot only first and second components. Note that the order of elements in the list matters; reversing the order of two indices reverses the axis on which each component is displayed.

Options

<code>out = arg</code> (<i>default</i> = “table”)	Output: table of eigenvalue and eigenvector results (“table”), graphs of ordered eigenvalues (“graph”), graph of the eigenvectors (“loadings”), graph of the component scores (“scores”), biplot of the loadings and scores (“biplot”). Note: when specifying the eigenvalue graph (“out = graph”), the option keywords “scree” (scree graph), “diff” (difference in successive eigenvalues), and “cproport” (cumulative proportion of total variance) may be included to control the output. By default, EViews will display the scree graph. If you may one or more the three keywords, EViews will construct the graph using only the specified types.
---	--

<code>n = integer</code>	<p>Maximum number of components to retain when presenting table (“out = table”) or eigenvalue graph (“out = graph”) results.</p> <p>The default is to set <i>n</i> to the number of variables.</p> <p>EViews will retain the minimum number satisfying any of: “n =”, “mineig =” or “cproport =”.</p>
<code>mineig = arg</code> (<i>default</i> = 0)	<p>Minimum eigenvalue threshold value: we retain components with eigenvalues that are greater than or equal to the threshold.</p> <p>EViews will retain the minimum number satisfying any of: “n =”, “mineig =” or “cproport =”.</p>
<code>cproport = arg</code> (<i>default</i> = 1)	<p>Cumulative proportion threshold value: we retain <i>k</i>, the number of components required for the sum of the first <i>k</i> eigenvalues exceeds the specified value for the cumulative variance explained proportion.</p> <p>EViews will retain the minimum number satisfying any of: “n =”, “mineig =” or “cproport =”.</p>
<code>eigval = vec_name</code>	<p>Specify name of vector to hold the saved the eigenvalues in workfile.</p>
<code>eigvec = mat_name</code>	<p>Specify name of matrix to hold the save the eigenvectors in workfile.</p>
<code>p</code>	<p>Print results.</p>

Covariance Options

<code>cov = arg</code> (<i>default</i> = “cov”)	<p>Covariance calculation method: ordinary (Pearson product moment) covariance (“cov”), ordinary correlation (“corr”), Spearman rank covariance (“rcov”), Spearman rank correlation (“rcorr”), Kendall’s tau-b (“taub”), Kendall’s tau-a (“taua”), uncentered ordinary covariance (“ucov”), uncentered ordinary correlation (“ucorr”).</p>
<code>wgt = name</code> (optional)	<p>Name of series containing weights.</p>
<code>wgtmethod = arg</code> (<i>default</i> = “sstdev”)	<p>Weighting method: frequency (“freq”), inverse of variances (“var”), inverse of standard deviation (“stdev”), scaled inverse of variances (“svar”), scaled inverse of standard deviations (“sstdev”).</p> <p>Only applicable for ordinary (Pearson) calculations where “weights =” is specified. Weights for rank correlation and Kendall’s tau calculations are always frequency weights.</p>
<code>pairwise</code>	<p>Compute using pairwise deletion of observations with missing cases (pairwise samples).</p>

<code>df</code>	<p>Compute covariances with a degree-of-freedom correction accounting for the mean (for centered specifications) and any partial conditioning variables (for the default “<code>cov = cov</code>”, and the “<code>cov = ucov</code>” specifications).</p> <p>The default behavior in these cases is to perform no adjustment (e.g. – compute sample covariance dividing by n rather than $n - k$).</p>
-----------------	--

Graph Options

<code>scale = arg,</code> (default = “ <i>normload</i> ”)	Diagonal matrix scaling of the loadings and the scores: normalize loadings (“ <i>normload</i> ”), normalize scores (“ <i>normscores</i> ”), symmetric weighting (“ <i>symmetric</i> ”), user-specified (<i>arg = number</i>).
<code>mult = arg</code> (default = “ <i>first</i> ”)	Multiple series handling: plot first against remainder (“ <i>first</i> ”), plot as x-y pairs (“ <i>pair</i> ”), lower-triangular plot (“ <i>lt</i> ”).
<code>nocenter</code>	Do not center graphs around the origin. By default, EViews centers biplots around (0, 0).
<code>labels = arg,</code> (default = “ <i>outlier</i> ”)	Observation labels for the scores: outliers only (“ <i>outlier</i> ”), all points (“ <i>all</i> ”), none (“ <i>none</i> ”).
<code>labelprob = number</code>	Probability value for determining whether a point is an outlier according to the chi-square tests based on the squared Mahalanbois distance between the observation and the sample means (when using the “ <i>labels = outlier</i> ” option).
<code>autoscale = arg</code>	Scale factor applied to the automatically specified loadings when displaying both loadings and scores). The default is to let EViews auto-choose a scale or to specify “ <i>user-scale =</i> ” to scale the original loadings.
<code>userscale = arg</code>	Scale factor applied to the original loadings when displaying both loadings and scores). The default is to let EViews auto-choose a scale, or to specify “ <i>autoscale =</i> ” to scale the automatically scaled loadings.
<code>cpnorm</code>	Compute the normalization for the score so that cross-products match the target (by default, EViews chooses a normalization scale so that the moments of the scores match the target).

Examples

```
freeze(tab1) mat1.pcomp(method=corr, eigval=v1, eigvec=m1)
```

stores the table view of the eigenvalues and eigenvectors of MAT1 in a table object named TAB1, the eigenvalues in a vector named V1, and the eigenvectors in a matrix named M1.

```
mat1.pcomp(method=cov, out=graph)
```

displays the scree plot of the ordered eigenvalues computed from the covariance matrix.

```
mat1.pcomp(method=rcorr, out=biplot, scale=normscores)
```

displays a biplot where the scores are normalized to have variances that equal the eigenvalues of the Spearman correlation matrix computed for the series in MAT1.

Cross-references

See “Principal Components” on page 397 of the *User’s Guide* for further discussion. See also “Covariance Analysis,” beginning on page 380 of the *User’s Guide* for discussion of the preliminary computation.

Note that this view analyzes the eigenvalues and eigenvectors of a covariance (or other association) matrix computed from the series in a group or the columns of a matrix. You may use [eigen \(p. 462\)](#) to examine the eigenvalues of a symmetric matrix.

read	Matrix Procs
------	------------------------------

Import data from a foreign disk file into a matrix.

May be used to import data into an existing workfile from a text, Excel, or Lotus file on disk.

Syntax

```
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. 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.

Options

File type options

t = dat, txt	ASCII (plain text) files.
t = wk1, wk3	Lotus spreadsheet files.
t = xls	Excel spreadsheet files.

If you do not specify the “t” option, EViews uses the file name extension to determine the file type. If you specify the “t” option, the file name extension will not be used to determine the file type.

Options for ASCII text files

<code>t</code>	Read data organized by column (transposed). Default is to read by row.
<code>na = text</code>	Specify text for NAs. Default is “NA”.
<code>d = t</code>	Treat tab as delimiter (note: you may specify multiple delimiter options). The <i>default</i> is “d = c” only.
<code>d = c</code>	Treat comma as delimiter.
<code>d = s</code>	Treat space as delimiter.
<code>d = a</code>	Treat alpha numeric characters as delimiter.
<code>custom = symbol</code>	Specify symbol/character to treat as delimiter.
<code>mult</code>	Treat multiple delimiters as one.
<code>rect (default) / norect</code>	[Treat / Do not treat] file layout as rectangular.
<code>skipcol = integer</code>	Number of columns to skip. Must be used with the “rect” option.
<code>skiprow = integer</code>	Number of rows to skip. Must be used with the “rect” option.
<code>comment = symbol</code>	Specify character/symbol to treat as comment sign. Everything to the right of the comment sign is ignored. Must be used with the “rect” option.
<code>singlequote</code>	Strings are in single quotes, not double quotes.
<code>dropstrings</code>	Do not treat strings as NA; simply drop them.
<code>negparen</code>	Treat numbers in parentheses as negative numbers.
<code>allowcomma</code>	Allow commas in numbers (note that using commas as a delimiter takes precedence over this option).

Options for spreadsheet (Lotus, Excel) files

<code>t</code>	Read data organized by column (transposed). Default is to read by row.
<code>letter_number (default = “b2”)</code>	Coordinate of the upper-left cell containing data.
<code>s = sheet_name</code>	Sheet name for Excel 5–8 Workbooks.

Examples

```
m1.read(t=dat,na=.) a:\mydat.raw
```

reads data into matrix M1 from an ASCII file MYDAT.RAW in the A: drive. The data in the file are listed by row, and the missing value NA is coded as a “.” (dot or period).

```
m1.read(t,a2,s=sheet3) cps88.xls
```

reads data into matrix M1 from an Excel file CPS88 in the default directory. The data are organized by column (transposed), the upper left data cell is A2, and the data is read from a sheet named SHEET3.

```
m2.read(a2, s=sheet2) "\\network\dr 1\cps91.xls"
```

reads the Excel file CPS91 into matrix M2 from the network drive specified in the path.

Cross-references

See [“Importing Data” on page 95](#) of the *User’s Guide I* for a discussion and examples of importing data from external files.

For powerful, easy-to-use tools for reading data into a new workfile, see [“Creating a Workfile by Reading from a Foreign Data Source” on page 41](#) of the *User’s Guide I*, [pageload](#) (p. 750), and [wfopen](#) (p. 802).

See also [Matrix::write](#) (p. 268).

setformat	Matrix Procs
-----------	------------------------------

Set the display format for cells in a matrix object spreadsheet view.

Syntax

```
matrix_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.

For matrices, `setformat` operates on all of the cells in the matrix.

To format numeric values, you should use one of the following format specifications:

<i>g[.precision]</i>	significant digits
<i>f[.precision]</i>	fixed decimal places
<i>c[.precision]</i>	fixed characters
<i>e[.precision]</i>	scientific/float
<i>p[.precision]</i>	percentage
<i>r[.precision]</i>	fraction

In order to set a format that groups digits into thousands, place a “t” after a specified format. For example, to obtain a fixed number of decimal places, with commas used to sepa-

rate thousands, use “ft[.precision]”. To obtain a fixed number of characters with a period used to separate thousands, use “ct[.precision]”.

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)”).

Examples

To set the format for all cells in the matrix to fixed 5-digit precision, simply provide the format specification:

```
matrix1.setformat f.5
```

Other format specifications include:

```
matrix1.setformat f(.7)
matrix1.setformat e.5
```

Cross-references

See [Matrix::setWidth \(p. 266\)](#), [Matrix::setindent \(p. 265\)](#) and [Matrix::setjust \(p. 266\)](#) for details on setting spreadsheet widths, indentation and justification.

setindent	Matrix Procs
-----------	------------------------------

Set the display indentation for cells in a matrix object spreadsheet view.

Syntax

```
matrix_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 spreadsheet (the EViews spreadsheet default font at the time the spreadsheet 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 spreadsheet view is created.

For matrices, `setindent` operates on all of the cells in the matrix.

Examples

To set the indentation for all the cells in a matrix object:

```
matrix1.setindent 2
```

Cross-references

See [Matrix::setWidth \(p. 266\)](#) and [Matrix::setjust \(p. 266\)](#) for details on setting spreadsheet widths and justification.

setjust	Matrix Procs
---------	------------------------------

Set the display justification for cells in a matrix object spreadsheet view.

Syntax

```
matrix_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.

For matrices, `setjust` operates on all of the cells in the matrix.

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 taken from the Global Defaults for spreadsheet views.

Examples

```
mat1.setjust middle
```

sets the vertical justification to the middle.

```
mat1.setjust top left
```

sets the vertical justification to top and the horizontal justification to left.

Cross-references

See [Matrix::setWidth](#) (p. 266) and [Matrix::setindent](#) (p. 265) for details on setting spreadsheet widths and indentation.

setWidth	Matrix Procs
----------	------------------------------

Set the column width for all columns in a matrix object spreadsheet.

Syntax

```
matrix_name.setWidth width_arg
```

where *width_arg* specifies the width unit value. The width unit is computed from representative characters in the default font for the current spreadsheet (the EViews spreadsheet default font at the time the spreadsheet 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

```
mat1.setwidth 12
```

sets the width of all columns in matrix MAT1 to 12 width units.

Cross-references

See [Matrix::setindent \(p. 265\)](#) and [Matrix::setjust \(p. 266\)](#) for details on setting spreadsheet indentation and justification.

sheet	Matrix Views
-------	------------------------------

Spreadsheet view of a matrix object.

Syntax

```
matrix_name.sheet(options)
```

Options

p	Print the spreadsheet view.
---	-----------------------------

Examples

```
mat1.sheet(p)
```

displays and prints the spreadsheet view of matrix MAT1.

stats	Matrix Views
-------	------------------------------

Descriptive statistics.

Computes and displays a table of means, medians, maximum and minimum values, standard deviations, and other descriptive statistics of each column in the matrix.

Syntax

```
matrix_name.stats(options)
```

Options

p	Print the stats table.
---	------------------------

Examples

```
mat1.stats
```

displays the descriptive statistics view of matrix MAT1.

Cross-references

See [“Descriptive Statistics & Tests” on page 306](#) and [page 379](#) of the *User’s Guide I* for a discussion of descriptive statistics views.

write	Matrix Procs
--------------	------------------------------

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

`matrix_name.write(options) [path\filename]`

Follow the name of the matrix object by a period, the keyword, and the name for the output file. 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. The entire matrix will be exported.

Note that EViews cannot, at present, write into an existing file. The file that you select will, if it exists, 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

<code>na = string</code>	Specify text string for NAs. Default is “NA”.
<code>d = arg</code>	Specify delimiter (<i>default</i> is tab): “s” (space), “c” (comma).
<code>t</code>	Write by column (transpose the data). Default is to write by row.

Spreadsheet (Lotus, Excel) files

<code>letter_number</code>	Coordinate of the upper-left cell containing data.
<code>t</code>	Write by column (transpose the data). Default is to write by row.

Examples

```
m1.write(t=txt,na=.) a:\dat1.csv
```

Writes the matrix M1 into an ASCII file named DAT1.CSV on the A: drive. NAs are coded as “.” (dot).

```
m1.write(t=txt,na=.) dat1.csv
```

writes the same file in the default directory.

```
m1.write(t=xls) "\\network\drive a\results"
```

saves the contents of M1 in an Excel file “Results.xls” in the specified directory.

Cross-references

See [“Exporting to a Spreadsheet or Text File” on page 105](#) of the *User’s Guide I* for a discussion.

See also [pagesave \(p. 752\)](#) and [Matrix::read \(p. 262\)](#).

Model

Set of simultaneous equations used for forecasting and simulation.

Model Declaration

modeldeclare model object (p. 284).

Declare an object by entering the keyword `model`, followed by a name:

```
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 429 of the *User’s Guide II*.

Model Views

blockdisplay model block structure (p. 275).
eqsview of model organized by equation (p. 277).
labelview or set label information for the model (p. 279).
msgdisplay model solution messages (p. 284).
textshow text showing equations in the model (p. 291).
traceview of trace output from model solution (p. 292).
varsview of model organized by variable (p. 294).

Model Procs

addassignassign add factors to equations (p. 272).
addinitinitialize add factors (p. 273).
appendappend a line of text to a model (p. 275).
controlsolve for values of control variable so that target matches trajectory (p. 276).
displaynameset display name (p. 276).
excludespecifies (or merges) excluded series to the active scenario (p. 277).
innovsolve options for stochastic simulation (p. 278).
makegraphmake graph object showing model series (p. 281).
makegroupmake group out of model series and display dated data table (p. 282).
mergemerge objects into the model (p. 283).
overridespecifies (or merges) override series to the active scenario (p. 285).
scenarioset the active, alternate, or comparison scenario (p. 286).
solvesolve the model (p. 287).
solveoptset solve options for model (p. 288).

- [spec](#) display the text specification view (p. 290).
- [stochastic](#) stochastic solution options (p. 290).
- [trace](#) specify endogenous variables to trace (p. 292).
- [track](#) specify endogenous variables to track (p. 292).
- [unlink](#) break links in specification (p. 293).
- [update](#) update model specification (p. 294).

Model Examples

The commands:

```
model mod1
mod1.append y=324.35+x
mod1.append x=-234+7.3*z
mod1.solve (m=100, c=.008)
```

create, specify, and solve the model MOD1.

The command:

```
mod1(g).makegraph gr1 x y z
```

plots the endogenous series X, Y, and Z, in the active scenario for model MOD1.

Model Entries

The following section provides an alphabetical listing of the commands associated with the “Model” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

addassign	Model Procs
-----------	-----------------------------

Assign add factors to equations.

Syntax

```
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 short-hand 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

i	Intercept shifts (default).
v	Variable shift.
n	None—remove add factors.
c	Change existing add factors to the specified type—if the “c” option is not used, only newly assigned add factors will be given the specified type.

Examples

```
m1.addassign(v) @all
```

assigns a variable shift to all equations in the model.

```
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 [“Using Add Factors” on page 432](#) of the *User’s Guide II*. See also [Chapter 36. “Models,” beginning on page 407](#) of the *User’s Guide II* for a general discussion of models.

See [Model::addinit \(p. 273\)](#).

addinit	Model Procs
---------	-----------------------------

Initialize add factors.

Syntax

```
model_name.addinit(options) equation_spec
```

where *equation_spec* identifies the equations for which you wish to initialize the add factors. You may either provide a list of endogenous variables, or you may use one of the following shorthand keywords:

@all	All equations
@stochastic	All stochastic equations (no identities)
@identity	All identities

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

<i>v = arg</i> (<i>default</i> = “z”)	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; <i>override</i> = actual).
<i>s = arg</i> (<i>default</i> = “a”)	Scenario selection: “a” (set active scenario add factors), “b” (set baseline scenario/actuals add factors), “o” (set active scenario override add factors).

Examples

```
m1.addinit(v=b) @all
```

sets all of the add factors in the active scenario to the values of the baseline.

```
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).

```
m1.addinit(s=b, v=z) @stochastic
```

sets the baseline scenario add factors to zero.

Cross-references

See [“Using Add Factors” on page 432](#) of the *User’s Guide II*. See also [Chapter 36. “Models,” on page 407](#) of the *User’s Guide II* for a general discussion of models.

See also `Model::addassign` (p. 272).

append[Model Procs](#)

Append a specification line to a model.

Syntax

```
model_name.append text
```

Type the text to be added after the `append` keyword.

Examples

```
model macro2
macro2.merge eq_m1
macro2.merge eq_gdp
macro2.append assign @all f
macro1.append @trace gdp
macro2.solve
```

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.

Cross-references

For details, see [“Models” on page 407](#) of the *User’s Guide II*.

block[Model Views](#)

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_name.block(options)
```

Options

p	Print the block structure view.
---	---------------------------------

Cross-references

See [“Block Structure View” on page 428](#) of the *User’s Guide II* for details. [Chapter 36](#) of the *User’s Guide II* provides a general discussion of models.

See also [Model::eqs](#) (p. 277), [Model::text](#) (p. 291) and [Model::vars](#) (p. 294) for alternative representations of the model.

control	Model Procs
----------------	-----------------------------

Solve for values of control variable so that the target series matches a trajectory.

Syntax

```
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

```
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

See “Solve Control for Target” on page 451 of the *User’s Guide II*. See [Chapter 36. “Models,”](#) on page 407 of the *User’s Guide II* for a general discussion of models.

displayname	Model Procs
--------------------	-----------------------------

Display name for model objects.

Attaches a display name to a model object which may be used in place of the standard model object name.

Syntax

```
model_name.displayname display_name
```

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in model object names.

Examples

```
mod1.displayname Sept 2006
mod1.label
```

The first line attaches a display name “Sept 2006” to the model object MOD1, and the second line displays the label view of MOD1, including its display name.

Cross-references

See “Labeling Objects” on page 72 of the *User’s Guide I* for a discussion of labels and display names.

See also `Model::label` (p. 279).

endog	Model Views
-------	-----------------------------

Note that `endog` and `makeendog` are no longer supported for model objects. See instead, `Model::makegroup` (p. 282).

eqs	Model Views
-----	-----------------------------

View of model organized by equation.

Lists the equations in the model. This view also allows you to identify which equations are entered by text, or by link, and to access and modify the equation specifications.

Syntax

`model_name.eqs`

Cross-references

See “Equation View” on page 426 of the *User’s Guide II* for details. See Chapter 36. “Models,” on page 407 of the *User’s Guide II* for a general discussion of models.

See also `Model::block` (p. 275), `Model::text` (p. 291), and `Model::vars` (p. 294) for alternative representations of the model.

exclude	Model Procs
---------	-----------------------------

Specifies (or merges) excluded endogenous variables in the active scenario.

Syntax

`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

m	Merge into instead of replace the existing exclude list.
actexist = arg	arg may be “t” (true) or “f” (false). When true, EViews will exclude periods for all endogenous variables where values of the actuals exist. (Applies to all endogenous variables, not just those explicitly listed in the proc.)

Examples

```
mod1.exclude fedfunds govexp("1990:01 1995:02")
```

will create an exclude list containing the variables FEDFUNDS and GOVEXP. FEDFUNDS will be excluded for the entire solution sample, while GOVEXP will only be excluded for the specified sample.

If you then issue the command:

```
mod1.exclude govexp
```

EViews will replace the original exclude list with one containing only GOVEXP. To add excludes to an existing list, use the “m” option:

```
mod1.exclude(m) fedfunds
```

The excluded list now contains both GOVEXP and FEDFUNDS.

```
mod1.exclude(actexist=t,m)
```

instructs EViews to keep all existing excludes (the “m” option) in the current active scenario and in addition to exclude all endogenous variables in periods where actuals exist.

Cross-references

See the discussion in [“Specifying Scenarios” on page 430](#) of the *User’s Guide II*.

See also [Model::model \(p. 284\)](#), [Model::override \(p. 285\)](#) and [Model::solveopt \(p. 288\)](#).

innov	Model Procs
-------	-----------------------------

Solve options for stochastic simulation.

Syntax

```
model_name.innov var1 option [var2 option, var3 option, ...]
```

Follow the `innov` keyword with a list of model variables and options. If the variable is an endogenous variable (or add factor), it identifies a model equation and will use different options than an exogenous variable.

Options

Options for endogenous variables

<code>“i” or “identity”</code>	Specifies that the equation is an identity in stochastic solution.
<code>“s” or “stochastic”</code>	Specifies that the equation is stochastic with unknown innovation variance in stochastic solution. Note: if a value has been previously specified in the <i>positive_num</i> option, it will be kept.
<i>positive_num</i>	Specifies that the equation is stochastic with an equation innovation standard error equal to the positive number <i>positive_num</i> . Note: the innovation standard error is only relevant when used with the <code>Model::stochastic</code> command, with the “v=t” option set.

Options for exogenous variables

<i>number</i>	<i>number</i> specifies the forecast standard error of the exogenous variable. You may use “NA” to specify an unknown (or zero) forecast error.
---------------	---

Examples

```
usmacro.innov gdp i
```

specifies that the endogenous variable GDP be treated as an identity in stochastic solution.

```
model01.innov cons 5600 gdp i cpi s
```

indicates that the endogenous variable CONS is stochastic with standard error equal to 5600, GDP is an identity, and CPI is stochastic with unknown innovation variance.

```
model01.innov govexp 12210
```

specifies that the forecast standard error of the exogenous variable GOVEXP is 12210.

Cross-references

See the discussion in [“Stochastic Options” on page 442](#) of the *User’s Guide II*.

See also [Model::model \(p. 284\)](#), [Model::stochastic \(p. 290\)](#), and [Model::solve \(p. 287\)](#).

label	Model Views Model Procs
-------	---

Display or change the label view of a model object, including the last modified date and display name (if any).

As a procedure, `label` changes the fields in the model object label.

Syntax

```
model_name.label
model_name.label(options) [text]
```

Options

The first version of the command displays the label view of the model. The second version may be used to modify the label. 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 <i>text</i> .
s	Sets the source field to <i>text</i> .
u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of M1 with “Data from CPS 1988 March File”:

```
m1.label(r)
m1.label(r) Data from CPS 1988 March File
```

To append additional remarks to M1, and then to print the label view:

```
m1.label(r) Log of hourly wage
m1.label(p)
```

To clear and then set the units field, use:

```
m1.label(u) Millions of bushels
```

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels.

See also [Model::displayname \(p. 276\)](#).

makeendog	Model Procs
-----------	-----------------------------

Note that in `endog` and `makeendog` are no longer supported for model objects. See instead, [Model::makegroup \(p. 282\)](#).

makegraph	Model Procs
-----------	-----------------------------

Make graph object showing model series.

Syntax

```
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 *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

a	Include actuals.
c	Include comparison scenarios.
d	Include deviations.
n	Do not include active scenario (by default the active scenario is included).
t = <i>trans_type</i> (default = level)	Transformation type: “level” (display levels in graph, “pch” (display percent change in graph), “pcha” (display percent change - annual rates - in graph), “pchy” (display 1-year percent change in graph), “dif” (display 1-period differences in graph), “dify” (display 1-year differences in graph).
s = <i>sol_type</i> (default = “d”)	Solution type: “d” (deterministic), “m” (mean of stochastic), “s” (mean and ± 2 std. dev. of stochastic), “b” (mean and confidence bounds of stochastic).
g = <i>grouping</i> (default = “v”)	Grouping setting for graphs: “v” (group series in graph by model variable), “s” (group series in graph by scenario), “u” (ungrouped - each series in its own graph).

Examples

```
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.

```
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 [“Displaying Data” on page 453](#) of the *User’s Guide II* for details. See [Chapter 36. “Models,” on page 407](#) of the *User’s Guide II* for a general discussion of models.

See `Model::makegroup` ([p. 282](#)).

makegroup	Model Procs
-----------	-----------------------------

Make a group out of model series and display dated data table.

Syntax

```
model_name.makegroup(options) grp_name model_vars
```

The `makegroup` keyword should be followed by 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

a	Include actuals.
c	Include comparison scenarios.
d	Include deviations.
n	Do not include active scenario (by default the active scenario is included).

<code>t = arg</code> (<i>default</i> = level)	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).
<code>s = arg</code> (<i>default</i> = “d”)	Solution type: “d” (deterministic), “m” (mean of stochastic), “s” (mean and ± 2 std. dev. of stochastic), “b” (mean and confidence bounds of stochastic).
<code>g = arg</code> (<i>default</i> = “v”)	Grouping setting for graphs: “v” (group series in graph by model variable), “s” (group series in graph by scenario).

Examples

```
model1.makegroup(a,n) group1 @endog
```

places all of the actual endogenous series in the group GROUP1.

Cross-references

See [“Displaying Data” on page 453](#) of the *User’s Guide II* for details. See [Chapter 36. “Models,” on page 407](#) of the *User’s Guide II* for a general discussion of models.

See also [Model::makegraph \(p. 281\)](#).

merge	Model Procs
--------------	-----------------------------

Merge equations from an estimated equation, model, pool, system, or var object.

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 directory 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 and an optional 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.

Syntax

```
model_name.merge(options) object_name
```

Follow the keyword with a name of an object containing estimated equation(s) to merge.

Options

<code>t</code>	Merge an ASCII text file.
----------------	---------------------------

Examples

```
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.

model	Model Declaration
-------	-----------------------------------

Declare a model object.

Syntax

`model model_name`

The keyword `model` should be followed by a name for the model. To fill the model, you may use [Model::append \(p. 275\)](#) or [Model::merge \(p. 283\)](#).

Examples

```
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
```

declares an empty model named MACRO and adds three lines to MACRO.

Cross-references

See [Chapter 36. “Models,” on page 407](#) of the *User’s Guide II* for a discussion of specifying and solving models in EViews.

See also [Model::append \(p. 275\)](#), [Model::merge \(p. 283\)](#) and [Model::solve \(p. 287\)](#).

msg	Model Views
-----	-----------------------------

Display model solution messages.

Show view containing messages generated by the most recent model solution.

Syntax

`model_name.msg(options)`

Options

p	Print the model solution messages.
---	------------------------------------

Cross-references

See [Chapter 36. “Models,” on page 407](#) of the *User’s Guide II* for a discussion of specifying and solving models in EViews.

See also [Model::solve \(p. 287\)](#) and [Model::solveopt \(p. 288\)](#).

override	Model Procs
----------	-----------------------------

Specifies (or merges) overridden exogenous variables and add factors in the active scenario.

Syntax

```
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 430](#) of the *User’s Guide II*. See also [Chapter 36. “Models,” on page 407](#) of the *User’s Guide II* for a general discussion of models.

See also [Model::model \(p. 284\)](#), [Model::exclude \(p. 277\)](#) and [Model::solveopt \(p. 288\)](#).

scenario	Model Procs
----------	-----------------------------

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_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

c	Set the comparison scenario to the named scenario.
n	Create a new scenario with the specified name.
i = " <i>name</i> "	Copy the Excludes and Overrides from the named scenario.
d	Delete the named scenario.
a = <i>string</i>	Set the scenario alias string to be used when creating aliased variables (<i>string</i> 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: "_5", "_T", "S2". The string "A" may not be used since it may conflict with add factor specifications.

Examples

The command string,

```
mod1.scenario "baseline"
```

sets the active scenario to the baseline, while:

```
mod1.scenario(c) "actuals"
```

sets the comparison scenario to the actuals (warning: this action will overwrite any historical data in the solution period).

A newly created scenario will become the active scenario. Thus:

```
mod1.scenario(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:

```
modl.scenario(a=_ps) "Peace Scenario"
```

changes the alias for “Peace Scenario” to “_PS” and makes this scenario the active scenario.

The command:

```
modl.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.

```
modl.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, the single character option “a” may be used to set the comparison scenario, but future support for this option is not guaranteed.

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

```
modl.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 422](#) of the *User’s Guide II*. [Chapter 36](#), “Models,” on [page 407](#) of the *User’s Guide II* documents EViews models in great depth.

See also [Model::solve](#) (p. 287).

solve	Model Procs
--------------	-----------------------------

Solve the model.

`solve` finds the solution to a simultaneous equation model for the set of observations specified in the current workfile sample.

Syntax

```
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. 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 [Model::solveopt \(p. 288\)](#).

Examples

```
mod1.solve
```

solves the model MOD1 using the default solution method.

```
nonlin2.solve (m=500,e)
```

solves the model NONLIN2 with an extended search of up to 500 iterations.

Cross-references

See [Chapter 36. “Models,” on page 407](#) of the *User’s Guide II* for a discussion of models.

See also [Model::model \(p. 284\)](#), [Model::msg \(p. 284\)](#) and [Model::solveopt \(p. 288\)](#).

solveopt	Model Procs
-----------------	-----------------------------

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_name.solveopt(options)
```

Options

<code>s = arg</code> (<i>default</i> = “d”)	Solution type: “d” (deterministic), “s” (stochastic). See Model::stochastic (p. 290) for setting stochastic solution options.
<code>d = arg</code> (<i>default</i> = “d”)	Model solution dynamics: “d” (dynamic solution), “s” (static solution), “f” (fitted values – single equation solution).
<code>m = integer</code> (<i>default</i> = 5000)	Maximum number of iterations for solution (maximum 100,000).
<code>c = number</code> (<i>default</i> = 1e-8)	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.
<code>a = arg</code> (<i>default</i> = “f”)	Alternate scenario solution: “t” (true - solve both active and alternate scenario and collect deviations for stochastic), “f” (false - solve only the active scenario).
<code>o = arg</code> (<i>default</i> = “g”)	Solution method: “g” (Gauss-Seidel), “n” (Newton), “b” (Broyden).
<code>i = arg</code> (<i>default</i> = “a”)	Set initial (starting) solution values: “a” (actuals), “p” (values in period prior to start of solution period).
<code>n = arg</code> (<i>default</i> = “t”)	NA behavior: “t” (true - stop on “NA” values), “f” (false - do not stop when encountering “NA” values). Only applies to deterministic solution; EViews will always stop on “NA” values in stochastic solution.
<code>e = arg</code> (<i>default</i> = “t”)	Excluded variables initialized from actuals: “t” (true), “f” (false).
<code>t = arg</code> (<i>default</i> = “u”)	Terminal condition for forward solution: “u” (user supplied - actuals), “l” (constant level), “d” (constant difference), “g” (constant growth rate).
<code>g = arg</code> (<i>default</i> = 7)	Number of digits to round solution: an integer value (number of digits), “n” (do not roundoff).
<code>z = arg</code> (<i>default</i> = 1e-7)	Zero value: a positive number below which the solution (absolute value) is set to zero, “n” (do not set to zero).
<code>f = arg</code> (<i>default</i> = “t”)	Order simultaneous blocks for minimum feedback: “t” (true), “f” (false).

Cross-references

See [Chapter 36. “Models,”](#) on page 407 of the *User’s Guide II* for a discussion of models.

See also [Model::model](#) (p. 284), [Model::msg](#) (p. 284) and [Model::solve](#) (p. 287).

spec	Model Views
------	-----------------------------

Display the text specification view for model objects.

Syntax

`model_name.spec(options)`

Options

p	Print the specification text.
---	-------------------------------

Examples

`model1.spec`

displays the specification of the object MODEL1.

Cross-references

See also [Model::append \(p. 275\)](#), [Model::merge \(p. 283\)](#), [Model::text \(p. 291\)](#).

stochastic	Model Procs
------------	-----------------------------

Stochastic solution options for models.

`stochastic` sets options for stochastic model solution but does not solve the model.

Syntax

`model_name.stochastic(options)`

Options

Note that these options have no effect on the current solve if deterministic solution has been selected. See the “s = ” option in [Model::solveopt \(p. 288\)](#).

i = arg (default = “n”)	Innovation generation: “n” (normal random number) or “b” (bootstrap).
d = arg (default = “f”)	Diagonal covariance matrix (for bootstrap: draw resid independently for each equation): “t” (true), “f” (false).
v = arg (default = “t”)	Scale covariance matrix to equation specified innovation variances: “t” (true), “f” (false). Does not apply to Bootstrap.
m = pos_number (default = 1.0)	Multiply resid covariance or bootstrap by the positive number <i>pos_number</i> .

<code>s = <i>quoted_sample</i></code>	Covariance estimation sample (Bootstrap residual draw sample). For example, <code>s = "1970.1 2003.4"</code>
<code>r = <i>integer</i></code> <code>(default = 1000)</code>	Number of stochastic repetitions.
<code>f = <i>number</i></code> <code>(default = .02)</code>	Fraction of failed repetitions before stopping.
<code>b = <i>number</i></code> <code>(default = .95)</code>	Size of stochastic confidence intervals.
<code>c = <i>arg</i></code> <code>(default = "f")</code>	Include coefficient uncertainty: "t" (true), "f" (false).
<code>p = <i>page_name</i></code>	Page name for a new workfile page to save the results of all repetitions of the stochastic solve. If blank (default) only summaries (mean, sd, etc.) of the repetitions are maintained.

Cross-references

See [Chapter 36. “Models,” on page 407](#) of the *User’s Guide II* for a discussion of models. See [Model::innov \(p. 278\)](#) to set options on individual series in stochastic solution.

See also [Model::model \(p. 284\)](#), [Model::solve \(p. 287\)](#) and [Model::solveopt \(p. 288\)](#).

text	Model Views
-------------	-----------------------------

Display text representation of the model specification.

Syntax

```
model_name.text(options)
```

The `text` command is equivalent to [Model::spec \(p. 290\)](#).

Options

<code>p</code>	Print the model text specification.
----------------	-------------------------------------

Examples

```
model1.text
```

displays the text representation of the object `MODEL1`.

Cross-references

See [Chapter 36. “Models,” on page 407](#) of the *User’s Guide II* for further details on models.

See also [Model::spec \(p. 290\)](#).

trace	Model Views
-------	-----------------------------

Display trace view of a model showing iteration history for selected solved variables.

Syntax

`model_name.trace(options)`

Options

p	Print the block structure view.
---	---------------------------------

Cross-references

See [“Diagnostics” on page 446](#) of the *User’s Guide II* for further details on tracing model solutions.

See also [Model::msg \(p. 284\)](#), [Model::solve \(p. 287\)](#) and [Model::solveopt \(p. 288\)](#).

trace	Model Procs
-------	-----------------------------

Specify endogenous variables to trace.

Sets the list of endogenous variables that will be traced at the next simulation. Intermediate results of all traced variables will be part of the model solution output.

Syntax

`model_name.trace endog1 [endog2 endog3 ...]`

Examples

```
modell1.trace gdp cons interest cpi
```

specifies that GDP, CONS, INTEREST, and CPI should be traced at the next simulation.

Cross-references

See also [Model::model \(p. 284\)](#) and [Model::track \(p. 292\)](#).

track	Model Procs
-------	-----------------------------

Specify endogenous variables to track.

Sets the list of endogenous variables that will be tracked at the next simulation. Results of all tracked endogenous variables will be part of the model solution output.

Syntax

```
model_name.track endog1 [endog2 endog3 ...]
```

Specify a list of endogenous variables to be tracked. You may use `@all` to track all endogenous variables.

Examples

```
modell.track gdp cons interest cpi
```

specifies that GDP, CONS, INTEREST, and CPI should be tracked at the next simulation.

```
modell.track @all
```

tracks all endogenous variables at the next simulation.

Cross-references

See also [Model::model](#) (p. 284) and [Model::trace](#) (p. 292).

unlink	Model Procs
---------------	-----------------------------

Break links in models.

Syntax

```
object.unlink spec
```

`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, 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 expressions:

```
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 36. “Models,” on page 407](#) of the *User’s Guide II* for a discussion of specifying and solving models in EViews. See also [Model::append \(p. 275\)](#), [Model::merge \(p. 283\)](#) and [Model::solve \(p. 287\)](#).

update	Model Procs
--------	-----------------------------

Update model specification.

Recompiles the model and updates all links.

Syntax

`model.update`

Follow the name of the model object by a period and the keyword `update`.

Examples

`mod1.update`

recompiles and updates all of the links in MOD1.

Cross-references

See [Chapter 36. “Models,” on page 407](#) of the *User’s Guide II* for a discussion of specifying and solving models in EViews. See also [Model::append \(p. 275\)](#), [Model::merge \(p. 283\)](#) and [Model::solve \(p. 287\)](#).

vars	Model Views
------	-----------------------------

View of model organized by variable.

Display the model in variable form with identification of endogenous, exogenous, and identity variables, with dependency tracking.

Syntax

`model_name.vars`

Cross-references

See [“Variable View” on page 428](#) of the *User’s Guide II* for details. See [Chapter 36. “Models,” on page 407](#) of the *User’s Guide II* for a general discussion of models.

See also [Model::block \(p. 275\)](#), [Model::text \(p. 291\)](#), and [Model::eqs \(p. 277\)](#) for alternative representations of the model.

Pool

Pooled time series, cross-section object. Used when working with data with both time series and cross-section structure.

Pool Declaration

pooldeclare pool object (p. 318).

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:

```
pool mypool
pool g7 _can _fr _ger _ita _jpn _us _uk
```

Pool Methods

lsestimate linear regression models including cross-section weighted least squares, and fixed and random effects models (p. 311).
tslslinear two-stage least squares (TSLS) regression models (p. 327).

Pool Views

cellipseConfidence ellipses for coefficient restrictions (p. 298).
coefcovcoefficient covariance matrix (p. 299).
cointJohansen's cointegration test (p. 300).
describecalculate pool descriptive statistics (p. 303).
fixedtesttest significance of estimates of fixed effects (p. 308).
labellabel information for the pool object (p. 310).
outputtable of estimation results (p. 317).
ranhausHausman test for correlation between random effects and regressors (p. 318).
representationstext showing equations in the model (p. 322).
residcorresidual correlation matrix (p. 322).
residcovresidual covariance matrix (p. 322).
residstable or graph of residuals for each pool member (p. 323).
resultstable of estimation results (p. 324).
sheetspreadsheet view of series in pool (p. 324).
testaddlikelihood ratio test for adding variables to pool equation (p. 326).
testdroplikelihood ratio test for dropping variables from pool equation (p. 327).
urootunit root test on a pool series (p. 331).
waldWald coefficient restriction test (p. 334).

Pool Procs

add add cross section members to pool (p. 298).
define define cross section identifiers (p. 302).
delete delete pool series (p. 303).
displayname set display name (p. 305).
drop drop cross section members from pool (p. 305).
fetch fetch series into workfile using a pool (p. 306).
genr generate pool series using the “?” (p. 309).
makegroup create a group of series from a pool (p. 314).
makemodel creates a model object from the estimated pool (p. 314).
makeresids make series containing residuals from pool (p. 315).
makestats make descriptive statistic series (p. 315).
makesystem creates a system object from the pool for other estimation methods (p. 317).
read import pool data from disk (p. 319).
store store pool series in database/bank files (p. 325).
updatecoefs update coefficient vector from pool (p. 330).
write export pool data to disk (p. 335).

Pool Data Members

String Values

@idname(i) i -th cross-section identifier.
@idnameest(i) i -th cross-section identifier for estimated equation.

Scalar Values

@aic Akaike information criterion.
@cofcov(i,j) covariance of coefficients i and j .
@coefs(i) coefficient i .
@dw 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.
@ncoef total number of estimated coefficients.
@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.

Vectors and Matrices

Pool Examples

```
mypool1.read(b2) data.xls x? y? z?
```

```
mypool1.delete x? y?
mypool1.store z?
```

```
mypool1.describe(m) z?
```

```
mypool1.ls y? c z? @ w?  
vector tstat1 = mypool1.@tstats
```

The following section provides an alphabetical listing of the commands associated with the “Pool” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

add	Pool Procs
------------	----------------------------

Add cross section members to a pool.

Syntax

```
pool_name.add id1 [id2 id3 ...]
```

List the cross-section identifiers to add to the pool.

Examples

```
countries.add us gr
```

Adds US and GR as cross-section members of the pool object COUNTRIES.

Cross-references

See [“Cross-section Identifiers” on page 461](#) of the *User’s Guide II* for a discussion of pool identifiers.

See also [Pool::drop \(p. 305\)](#).

cellipse	Pool Views
-----------------	----------------------------

Confidence ellipses for coefficient restrictions.

The `cellipse` view displays confidence ellipses for pairs of coefficient restrictions for an estimation from a pool object.

Syntax

```
pool_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

<code>ind = arg</code>	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.
<code>size = number</code> <i>(default = 0.95)</i>	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.
<code>dist = arg</code>	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 χ^2 distribution.
<code>p</code>	Print the graph.

Examples

The two commands:

```
pool1.ellipse c(1), c(2), c(3)
pool1.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 [“Confidence Ellipses” on page 142](#) of the *User’s Guide II* for discussion.

See also [Pool::wald \(p. 334\)](#).

coefcov	Pool Views
----------------	----------------------------

Coefficient covariance matrix.

Displays the covariances of the coefficient estimates for an estimated pool object.

Syntax

```
pool_name.coefcov(options)
```

Options

p	Print the coefficient covariance matrix.
---	--

Examples

```
pool1.coefcov
```

displays the coefficient covariance matrix for POOL1 in a window. To store the coefficient covariance matrix as a sym object, use “@coefcov”:

```
sym eqcov = pool1.@coefcov
```

Cross-references

See also [Coef::coef](#) (p. 16).

coint	Pool Views
-------	----------------------------

Johansen’s cointegration test.

Syntax

```
pool_name.coint(option) pool_ser1 pool_ser2 [pool_ser3]...
```

Follow the pool name with the `coint` keyword, any options, and a list of two or more ordinary or pool series.

Options

You may specify the type using one of the following keywords:

Pedroni (default)	Pedroni (1994 and 2004).
Kao	Kao (1999)
Fisher	Fisher - pooled Johansen

Depending on the type selected above, the following may be used to indicate deterministic trends:

const (default)	Include a constant in the test equation. Applicable to Pedroni and Kao tests.
trend	Include a constant and a linear time trend in the test equation. Applicable to Pedroni tests.
none	Do not include a constant or time trend. Applicable to Pedroni tests.
a	No deterministic trend in the data, and no intercept or trend in the cointegrating equation. Applicable to Fisher tests.
b	No deterministic trend in the data, and an intercept but no trend in the cointegrating equation. Applicable to Fisher tests.
c	Linear trend in the data, and an intercept but no trend in the cointegrating equation. Applicable to Fisher tests.
d	Linear trend in the data, and both an intercept and a trend in the cointegrating equation. Applicable to Fisher tests.
e	Quadratic trend in the data, and both an intercept and a trend in the cointegrating equation. Applicable to Fisher tests.

Additional options:

<code>ac = arg</code> (<i>default</i> = “bt”)	Method of estimating the frequency zero spectrum: “bt” (Bartlett kernel), “pr” (Parzen kernel), “qs” (Quadratic Spectral kernel). Applicable to Pedroni and Kao tests.
<code>band = arg</code> (<i>default</i> = “nw”)	Method of selecting the bandwidth, where <i>arg</i> may be “nw” (Newey-West automatic variable bandwidth selection), or a number indicating a user-specified common bandwidth. Applicable to Pedroni and Kao tests.
<code>lag = arg</code>	For Pedroni and Kao tests, the method of selecting lag length (number of first difference terms) to be included in the residual regression. For Fisher tests, a pair of numbers indicating lag.

<code>info = arg</code> <code>(default = "aic")</code>	Information criterion to use when computing automatic lag length selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn). Applicable to Pedroni and Kao tests.
<code>maxlag = int</code>	Maximum lag length to consider when performing automatic lag length selection, where <i>int</i> is an integer. Default = $\text{floor}[\min(12, T_i/3)(T_i/100)^{0.25}]$ where T_i is the length of the cross-section. Applicable to Pedroni and Kao tests.
<code>disp = arg</code> <code>(default = 500)</code>	Maximum number of individual results to be displayed.

Examples

```
pool01.coint(fisher, lag=1 2, c) y? x1? x2?
```

performs a Johansen test for pool series Y?, X1?, and X2? with a lag of 1 to 2 and linear trend in the data, and an intercept but no trend in the cointegrating equation is assumed as exogenous variables.

Cross-references

See [“Cointegration Testing” on page 363](#) of the *User’s Guide II* for details on the Johansen test. See also `Pool::uroot` (p. 331).

define	Pool Procs
--------	----------------------------

Define cross section members (identifiers) in a pool.

Syntax

```
pool_name.define id1 [id2 id3 ...]
```

List the cross section identifiers after the `define` keyword.

Examples

```
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 37. “Pooled Time Series, Cross-Section Data,” on page 459](#) of the *User’s Guide II* for a discussion of cross-section identifiers.

See also [Pool::add \(p. 298\)](#), [Pool::drop \(p. 305\)](#) and [Pool::pool \(p. 318\)](#).

delete	Pool Procs
---------------	----------------------------

Deletes series based upon identifiers in a pool.

Syntax

```
pool_name.delete pool_ser1 [pool_ser2 pool_ser3 ...]
```

Follow the keyword by a list of the names of any series you wish to remove from the current workfile. Deleting does *not* remove objects that have been stored on disk in EViews database files.

The `delete` command allows you to delete series from the workfile using ordinary and pool series names.

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.

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 series in the workfile with names beginning with “CPI” that are followed by identifiers in the pool object MYPOOL:

```
mypool.delete cpi?
```

Cross-references

See [Chapter 4. “Object Basics,” on page 63](#) of the *User’s Guide I* for a discussion of working with objects, and [Chapter 10. “EViews Databases,” on page 257](#) of the *User’s Guide I* for a discussion of EViews databases.

describe	Pool Views
-----------------	----------------------------

Computes and displays descriptive statistics for the pooled data.

Syntax

```
pool_name.describe(options) pool_ser1 [pool_ser2 pool_ser3 ...]
```

List the name of ordinary and pool series for which you wish to compute descriptive statistics.

By default, statistics are computed for each stacked pool series, using only common observations where *all* of the cross-sections for a given series have nonmissing data. A missing observation for a series in any one cross-section causes that observation to be dropped for all cross-sections for the corresponding series. You may change the 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

m	Stack data and subtract cross-section specific means from each variable—this option provides the within estimators.
c	Do not stack data—compute statistics individually for each cross-sectional unit.
t	Time period specific—compute statistics for each period, taken over all cross-section identifiers.
i	Individual sample—includes every valid observation for the series even if data are missing from other series in the list.
b	Balanced sample—constrains each cross-section to have the <i>same observations</i> . If an observation is missing for any series, in any cross-section, it will be dropped for all cross-sections.
p	Print the descriptive statistics.

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 37. “Pooled Time Series, Cross-Section Data,” on page 459](#) of the *User’s Guide II* for a discussion of the computation of these statistics, and a description of individual and balanced samples.

displayname	Pool Procs
-------------	----------------------------

Display name for pool objects.

Attaches a display name to a pool object which may be used to label output in place of the standard pool object name.

Syntax

```
pool_name.displayname display_name
```

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in pool object names.

Examples

```
hrs.displayname Hours Worked
hrs.label
```

The first line attaches a display name “Hours Worked” to the pool object HRS, and the second line displays the label view of HRS, including its display name.

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels and display names.

See also [Pool::label](#) (p. 310).

drop	Pool Procs
------	----------------------------

Drops cross-section members from a pool.

Syntax

```
pool_name.drop id1 [id2 id3 ...]
```

List the cross-section members to be dropped from the pool.

Examples

```
crosssc.drop jpn kor hk
```

drops the cross-section members JPN, KOR, and HK from the pool CROSSSC.

Cross-references

[“Cross-section Identifiers” on page 461](#) of the *User’s Guide II* discusses pool identifiers.

See also [Pool::add](#) (p. 298).

fetch[Pool Procs](#)

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 using the object in the database or using the databank file name. 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 an attached conversion method, EViews will use the method set by **Options/Date-Frequency** in the main menu. You can override the conversion method by specifying an explicit conversion method option.

Syntax

```
pool_name.fetch(options) pool_ser1 [pool_ser2 pool_ser3 ...]
```

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 optionally fetch from individual databank files or search among registered databases.

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.

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 including identifying extensions.

Options

`d = db_name` Fetch from specified database.

d	Fetch all registered databases in registry order.
i	Fetch from individual databank files.
notifyillegal	When in a program, report illegal EViews object names. By default, objects with illegal names are automatically renamed. (Has no effect in the command window.)

The database specified by the double colon “::” takes precedence over the database specified by the “d = ” option.

In addition, there are a number of 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:

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	<p><i>High to low conversion methods removing NAs:</i> “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).</p> <p><i>High to low conversion methods propagating NAs:</i> “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).</p>
---------	--

If no conversion method is specified, the series-specific or global default conversion method will be employed.

Examples

To fetch M1, GDP, and UNEMP pool series from the default database, use:

```
pool1.fetch m1? gdp? unemp?
```

To fetch M1 and GDP from the US1 database and UNEMP from the MACRO database, use the command:

```
pool1.fetch(d=us1) m1? gdp? macro::unemp
```

Use the “notifyillegal” option to display a dialog when fetching the series MYIL-LEG@LNAME that will suggest a valid name and give you to opportunity to name the object before it is inserted into a workfile:

```
pool2.fetch(notifyillegal) myilleg@lname
```

Cross-references

See [Chapter 10. “EViews Databases,” on page 257](#) of the *User’s Guide I* for a discussion of databases, databank files, and frequency conversion. [Appendix C. “Wildcards,” on page 775](#) of the *User’s Guide I* describes the use of wildcard characters.

See also [Series::setconvert \(p. 377\)](#), [Pool::store \(p. 325\)](#), and [Pool::store \(p. 325\)](#).

fixedtest	Pool Views
------------------	----------------------------

Test joint significance of the fixed effects estimates.

Tests the hypothesis that the estimated fixed effects are jointly significant using F and LR test statistics. If the estimated specification involves two-way fixed effects, three separate tests will be performed; one for each set of effects, and one for the joint effects.

Only valid for panel or pool regression equations estimated with fixed effects. Not currently available for specifications estimated using instrumental variables.

Syntax

```
pool_name.fixedtest(options)
```

Options

p	Print output from the test.
---	-----------------------------

Examples

```
pool1.fixedtest
```

tests whether the fixed effects are jointly significant.

Cross-references

See also [Pool::testadd \(p. 326\)](#), [Pool::testdrop \(p. 327\)](#), [Pool::ranhaus \(p. 318\)](#), and [Pool::wald \(p. 334\)](#).

genr	Pool Procs
-------------	----------------------------

Generate series.

This procedure allows you to generate multiple series using the cross-section identifiers in a pool.

Syntax

```
pool_name.genr ser_name = expression
```

You may use the cross section identifier “?” in the series name and/or in the expression on the right-hand side.

Examples

The commands,

```
pool pool1
pool1.add 1 2 3
pool1.genr y? = x? - @mean(x?)
```

are equivalent to generating separate series for each cross-section:

```
genr y1 = x1 - @mean(x1)
genr y2 = x2 - @mean(x2)
genr y3 = x3 - @mean(x3)
```

Similarly:

```
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.

It is worth noting that the pool `genr` command simply loops across the cross-section identifiers, performing the evaluations using the appropriate substitution. Thus, the command,

```
pool2.genr z = y_?
```

is equivalent to entering:

```
genr z = y_us
genr z = y_uk
genr z = y_can
```

so that upon completion, the ordinary series Z will contain Y_CAN.

Cross-references

See [Chapter 37. “Pooled Time Series, Cross-Section Data,” on page 459](#) of the *User’s Guide II* for a discussion of the computation of pools, and a description of individual and balanced samples.

See [Series::series \(p. 375\)](#) for a discussion of the expressions allowed in `genr`.

label	Pool Views Pool Procs
-------	---

Display or change the label view of a pool object, including the last modified date and display name (if any).

As a procedure, `label` changes the fields in the pool object label.

Syntax

```
pool_name.label
pool_name.label(options) [text]
```

Options

The first version of the command displays the label view of the pool object. The second version may be used to modify the label. 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 <i>text</i> .
s	Sets the source field to <i>text</i> .
u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of POOL1 with “Data from CPS 1988 March File”:

```
pool1.label(r)
pool1.label(r) Data from CPS 1988 March File
```

To append additional remarks to POOL1, and then to print the label view:

```
pool1.label(r) Log of hourly wage
```

```
pool1.label(p)
```

To clear and then set the units field, use:

```
pool1.label(u) Millions of bushels
```

Cross-references

See “[Labeling Objects](#)” on page 72 of the *User’s Guide I* for a discussion of labels.

See also [Pool::displayname](#) (p. 305).

ls	Pool Methods
----	------------------------------

Estimation by linear or nonlinear least squares regression.

ls estimates cross-section weighed least squares, feasible GLS, and fixed and random effects models.

Syntax

```
pool_name.ls(options) y x1 [x2 x3...] [@cxreg z1 z2 ...] [@perreg z3 z4 ...]
```

ls carries out pooled data estimation. Type the name of the dependent variable followed by one or more 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 pool variables that have different coefficients for each cross-section member of the pool. If there is a cross-section specific regressor list, the two lists must be separated by “@CXREG”. The third list, if provided, should contain pool variables that have different coefficients for each period. The list should be separated from the previous lists by “@PERREG”.

You may include AR terms as regressors in either the common or cross-section specific lists. AR terms are, however, not allowed for some estimation methods. MA terms are not supported.

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. The criterion will be set to the nearest value between 1e-24 and 0.2.
s	Use the current coefficient values in “C” as starting values for equations with AR or MA terms (see also param (p. 761)).

<code>s = number</code>	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 to be used. Note that out of range values are set to “s = 1”. Specifying “s = 0” initializes coefficients to zero. By default EViews uses “s = 1”.
<code>showopts / -showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
<code>deriv = keyword</code>	Set derivative methods. The argument <i>keyword</i> 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.
<code>cx = arg</code>	Cross-section effects: (default) none, fixed effects (“cx = f”), random effects (“cx = r”).
<code>per = arg</code>	Period effects: (default) none, fixed effects (“per = f”), random effects (“per = r”).
<code>wgt = arg</code>	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”).
<code>cov = arg</code>	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/PCSE (“cov = perdiag”).
<code>keepwghts</code>	Keep full set of GLS weights used in estimation with object, if applicable (by default, only small memory weights are saved).
<code>rancalc = arg (default = “sa”)</code>	Random component method: Swamy-Arora (“rancalc = sa”), Wansbeek-Kapteyn (“rancalc = wk”), Wallace-Hussain (“rancalc = wh”).
<code>nodf</code>	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 = <i>arg</i>	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the “C” coefficient vector.
iter = <i>arg</i> (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.
p	Print basic estimation results.

Examples

```
pool1.ls dy? c inv? edu? year
```

estimates pooled OLS of DY? on a constant, INV?, EDU? and YEAR.

```
pool1.ls(cx=f) dy? @cxreg inv? edu? year ar(1)
```

estimates a fixed effects model without restricting any of the coefficients to be the same across pool members.

Cross-references

[Chapter 24. “Basic Regression,” on page 5](#) and [Chapter 25. “Additional Regression Methods,” on page 23](#) of the *User’s Guide II* discuss the various regression methods in greater depth.

See [Chapter 37. “Pooled Time Series, Cross-Section Data,” on page 459](#) of the *User’s Guide II* for a discussion of pool estimation, and [Chapter 39. “Panel Estimation,” on page 541](#) of the *User’s Guide II* for a discussion of panel equation estimation.

See [Chapter 5. “Special Expression Reference,” on page 815](#), for special terms that may be used in `ls` specifications.

See also [Pool::tsls \(p. 327\)](#) for instrumental variables estimation.

makegroup	Pool Procs
-----------	----------------------------

Make a group out of pool and ordinary series using a pool object.

Syntax

```
pool_name.makegroup(group_name) pool_series1 [pool_series2 pool_series3 ...]
```

You should provide a name for the new group in parentheses, then list the ordinary and pool series to be placed in the group.

Examples

```
pool11.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.

Cross-references

See [“Displaying Data” on page 453](#) of the *User’s Guide II* for details.

makemodel	Pool Procs
-----------	----------------------------

Make a model from a pool object.

Syntax

```
pool_name.makemodel(name) assign_statement
```

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

```
pool13.ls m1? gdp? tb3?  
pool13.makemodel(poolmod) @prefix s_
```

estimates a VAR and makes a model named POOLMOD from the estimated pool object. POOLMOD includes an assignment statement “ASSIGN @PREFIX S_”. Use the command “show poolmod” or “poolmod.spec” to open the POOLMOD window.

Cross-references

See [Chapter 36. “Models,” on page 407](#) of the *User’s Guide II* for a discussion of specifying and solving models in EViews.

See also [Model::merge \(p. 283\)](#) and [Model::solve \(p. 287\)](#).

makeresids[Pool Procs](#)

Create residual series.

Creates and saves residuals in the workfile from a pool object.

Syntax

```
pool_name.makeresids [poolser]
```

Follow the object name with a period and the `makeresids` keyword, then provide a list of names to be given to the stored residuals. You may use a cross section identifier “?” to specify a set of names.

Options

<code>n = arg</code>	Create group object to hold the residual series.
----------------------	--

Examples

```
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

See [“Weighted Least Squares” on page 32](#) of the *User’s Guide II* for a discussion of standardized residuals after weighted least squares and [Chapter 30. “Discrete and Limited Dependent Variable Models,” on page 209](#) of the *User’s Guide II* for a discussion of standardized and generalized residuals in binary, ordered, censored, and count models.

makestats[Pool Procs](#)

Create and save series of descriptive statistics computed from a pool object.

Syntax

```
pool_name.makestats(options) pool_series1 [pool_series2 ...] @ stat_list
```

You should provide options, a list of series names, an “@” separator, and a list of command names for the statistics you wish to compute. The 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.
o	Force the overwrite of the computed statistics series if they already exist. The default creates a new series using the next available names.

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.
jarq	Jarque-Bera test statistic.
min	Minimum value.
max	Maximum value.

Examples

```
pool11.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, and EDU_SD.

```
pool11.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 37. “Pooled Time Series, Cross-Section Data,” on page 459](#) of the *User’s Guide II* for details on the computation of these statistics and a discussion of the use of individual, common, and balanced samples in pool.

See also [Pool::describe \(p. 303\)](#).

makesystem[Pool Procs](#)

Create system from a pool object.

Syntax

```
pool_name.makesystem(options) pool_spec
```

Creates a system out of the pool equation specification. See [Pool::ls \(p. 311\)](#) for details on the syntax of *pool_spec*. Note that period specific coefficients and effects are not available in this routine.

Options

`name = name` Specify name for the object.

Examples

```
pool1.makesystem(name=sys1) inv? cap? @inst 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(name=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 33. “System Estimation,” on page 307](#) of the *User's Guide II* for a discussion of system objects in EViews.

output[Pool Views](#)

Display estimation output.

`output` changes the default object view to display the estimation output (equivalent to using [Pool::results \(p. 324\)](#)).

Syntax

```
pool_name.output
```

Options

p	Print estimation output for estimation object
---	---

Examples

The `output` keyword may be used to change the default view of an estimation object. Entering the command:

```
pool1.output
```

displays the estimation output for pool POOL1.

Cross-references

See [Pool::results \(p. 324\)](#).

pool	Pool Declaration
------	----------------------------------

Declare pool object.

Syntax

```
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 [Pool::add \(p. 298\)](#) and [Pool::drop \(p. 305\)](#).

Examples

```
pool zool dog cat pig owl ant
```

Declares a pool object named ZOO1 with the listed cross-section identifiers.

Cross-references

See [Chapter 37. “Pooled Time Series, Cross-Section Data,” on page 459](#) of the *User’s Guide II* for a discussion of working with pools in EViews.

See [Pool::add \(p. 298\)](#) and [Pool::drop \(p. 305\)](#). See also [Pool::ls \(p. 311\)](#) for details on estimation using a pool object.

ranhaus	Pool Views
---------	----------------------------

Test for correlation between random effects and regressors using Hausman test.

Tests the hypothesis that the random effects (components) are correlated with the right-hand side variables in a pool equation setting. Uses Hausman test methodology to compare

the results from the estimated random effects specification and a corresponding fixed effects specification. If the estimated specification involves two-way random effects, three separate tests will be performed; one for each set of effects, and one for the joint effects.

Only valid for pool regression equations estimated with random effects. Note that the test results may be suspect in cases where robust standard errors are employed.

Syntax

```
pool_name.ranhaus(options)
```

Options

p	Print output from the test.
---	-----------------------------

Examples

```
pool1.ls(cx=r) sales? c adver? lsales?
```

```
pool1.ranhaus
```

estimates a specification with cross-section random effects and tests whether the random effects are correlated with the right-hand side variables ADVER and LSALES using the Hausman test methodology.

Cross-references

See also [Pool::testadd](#) (p. 326), [Pool::testdrop](#) (p. 327), [Pool::fixedtest](#) (p. 308), and [Pool::wald](#) (p. 334).

read	Pool Procs
------	----------------------------

Import data from a foreign disk file into a pool object.

May be used to import data into an existing workfile from a text, Excel, or Lotus file on disk.

Syntax

```
pool_name.read(options) [path\]file_name pool_ser1 [pool_ser2 pool_ser3 ...]
```

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. 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.

Follow the source file name with a list of ordinary or pool series.

Options

File type options

t = dat, txt	ASCII (plain text) files.
t = wk1, wk3	Lotus spreadsheet files.
t = xls	Excel spreadsheet files.

If you do not specify the “t” option, EViews uses the file name extension to determine the file type. If you specify the “t” option, the file name extension will not be used to determine the file type.

Options for ASCII text files

t	Read data organized by series. Default is to read by observation with series in columns.
na = <i>text</i>	Specify text for NAs. Default is “NA”.
d = t	Treat tab as delimiter (note: you may specify multiple delimiter options). The <i>default</i> is “d = c” only.
d = c	Treat comma as delimiter.
d = s	Treat space as delimiter.
d = a	Treat alpha numeric characters as delimiter.
custom = <i>symbol</i>	Specify symbol/character to treat as delimiter.
mult	Treat multiple delimiters as one.
name	Series names provided in file.
label = <i>integer</i>	Number of lines between the header line and the data. Must be used with the “name” option.
rect (<i>default</i>) / norect	[Treat / Do not treat] file layout as rectangular.
skipcol = <i>integer</i>	Number of columns to skip. Must be used with the “rect” option.
skiprow = <i>integer</i>	Number of rows to skip. Must be used with the “rect” option.
comment = <i>symbol</i>	Specify character/symbol to treat as comment sign. Everything to the right of the comment sign is ignored. Must be used with the “rect” option.
singlequote	Strings are in single quotes, not double quotes.
dropstrings	Do not treat strings as NA; simply drop them.

<code>negparen</code>	Treat numbers in parentheses as negative numbers.
<code>allowcomma</code>	Allow commas in numbers (note that using commas as a delimiter takes precedence over this option).
<code>currency = sym- bol</code>	Specify symbol/character for currency data.

Options for spreadsheet (Lotus, Excel) files

<code>t</code>	Read data organized by series. Default is to read by observation with series in columns.
<code>letter_number</code> (<i>default</i> = “b2”)	Coordinate of the upper-left cell containing data.
<code>s = sheet_name</code>	Sheet name for Excel 5–8 Workbooks.

Options for pool reading

<code>bycross</code> (<i>default</i>) /byper	Structure of stacked pool data [cross-section / date or period] (only for pool read).
---	---

Examples

```
pool1.read(t=dat,na=.) a:\mydat.raw year lwage? hrs?
```

reads stacked data from an ASCII file MYDAT.RAW in the A: drive. The data in the file are stacked by cross-section, the missing value NA is coded as a “.” (dot or period). We read one ordinary series YEAR, and three two pool series LWAGE? and HRS?.

```
pool1.read(a2,s=sheet3,byper) statepan.xls inc? educ? pop?
```

reads data from an Excel file STATEPAN in the default directory. The data are stacked by period in the sheet SHEET3 with the upper left data cell A2. We read three pool series INC? EDUC? and POP?.

Cross-references

See [“Importing Data” on page 95](#) of the *User’s Guide I* for a discussion and examples of importing data from external files, and [Chapter 38. “Working with Panel Data,” beginning on page 509](#) for panel data alternatives to working with pooled data.

For powerful, easy-to-use tools for reading data into a new workfile, see [“Creating a Workfile by Reading from a Foreign Data Source” on page 41](#) of the *User’s Guide I*, [pageload](#) (p. 750), and [wfoopen](#) (p. 802).

See also [Pool::write](#) (p. 335).

representations	Pool Views
-----------------	----------------------------

Display text of specification for pool objects.

Syntax

`pool_name.representation(options)`

Options

p	Print the representation text.
---	--------------------------------

Examples

```
pool1.representations
```

displays the specifications of the estimation object POOL1.

residcor	Pool Views
----------	----------------------------

Residual correlation matrix.

Displays the correlations of the residuals from each pool cross-section equation.

Syntax

`pool_name.residcor(options)`

Options

p	Print the correlation matrix.
---	-------------------------------

Examples

```
pool1.residcor
```

displays the residual correlation matrix of POOL1.

Cross-references

See also [Pool::residcov](#) (p. 322) and [Pool::makeresids](#) (p. 315).

residcov	Pool Views
----------	----------------------------

Residual covariance matrix.

Displays the covariances of the residuals from each pool cross-section equation.

Syntax

```
pool_name.residcov(options)
```

Options

p	Print the covariance matrix.
---	------------------------------

Examples

```
pool1.residcov
```

displays the residual covariance matrix of POOL1.

Cross-references

See also [Pool::residcor](#) (p. 322) and [Pool::makeresids](#) (p. 315).

resids	Pool Views
--------	----------------------------

Display residuals.

Display the actual, fitted values and residuals in either tabular or graphical form. `resids` displays multiple graphs, where each graph will contain the residuals for each cross-section in the pool.

Syntax

```
pool_name.resids(options)
```

Options

g (<i>default</i>)	Display graph(s) of residuals.
p	Print the table/graph.

Examples

```
pool1.ls m1? c inc? tb3?  
pool1.resids
```

regresses M1 on a constant, INC, and TB3, and displays a table of actual, fitted, and residual series.

```
pool1.resids(g)
```

displays a graph of the actual, fitted, and residual series.

Cross-references

See also [Pool::makeresids](#) (p. 315).

results	Pool Views
---------	------------

Displays the results view of a pool object.

Syntax

```
pool_name.results(options)
```

Options

p	Print the view.
---	-----------------

Examples

```
pool1.ls m1? c inc? tb3?  
pool1.results(p)
```

estimates an equation using least squares, and displays and prints the results.

sheet	Pool Views
-------	------------

Spreadsheet view of a pool object.

Syntax

```
pool_name.sheet(options) pool_ser1 [pool_ser2 pool_ser3 ...]
```

The `sheet` view 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

```
pool1.sheet(p) x? y? z?
```

displays and prints the pool spreadsheet view of the series X?, Y?, and Z?.

Cross-references

See [Chapter 37. “Pooled Time Series, Cross-Section Data,” on page 459](#) of the *User’s Guide II* for a discussion of pools.

store	Pool Procs
-------	----------------------------

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. EViews will first expand the list of series using the pool operator, and then perform the operation.

Syntax

```
pool_name.store(options) pool_ser1 [pool_ser2 pool_ser3 ...]
```

Follow the `store` command keyword with a list of 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 2 and earlier where the default was to store objects in individual databank files).*

You may 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.

You may use the wild card character “*” 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, 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

<code>d = db_name</code>	Store to the specified database.
<code>i</code>	Store to individual databank files.
<code>1 / 2</code>	Store series in [single / double] precision to save space.
<code>o</code>	Overwrite object in database (default is to merge data, where possible).
<code>g = arg</code>	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 766](#) of the *User’s Guide II*.

Examples

```
pool1.store m1? gdp? unemp?
```

stores the three pool objects M1, GDP, UNEMP in the default database.

```
pool1.store(d=us1) m1? gdp? macro::unemp?
```

Cross-references

[“Basic Data Handling” on page 77](#) of the *User’s Guide I* discusses exporting data in other file formats. See [Chapter 10. “EViews Databases,” on page 257](#) of the *User’s Guide I* for a discussion of EViews databases and databank files.

For additional discussion of wildcards, see [Appendix C. “Wildcards,” on page 775](#) of the *User’s Guide I*.

See also [Pool::fetch \(p. 306\)](#).

testadd	Pool Views
---------	----------------------------

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

```
pool_name.testadd(options) [x1 x2 ...] [@cxreg z1 z2 ...] [@perreg z3 z4 ...]
```

List the names of the series to test for omission after the keyword.

Options

p	Print output from the test.
---	-----------------------------

Examples

```
pool1.testadd gdp? @cxreg inc?
```

tests the addition of the pool series GDP? to the common coefficients list and INC? to the cross-section specific coefficients list.

Cross-references

See [“Coefficient Tests” on page 142](#) of the *User’s Guide II* for further discussion.

See also [Pool::testdrop](#) (p. 327) and [Pool::wald](#) (p. 334).

testdrop	Pool Views
-----------------	----------------------------

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

```
pool_name.testdrop(options) arg1 [arg2 arg3 ...]
```

List the names of the series to test for omission after the keyword.

Options

p	Print output from the test.
---	-----------------------------

Examples

```
pool1.testdrop(p) x?
```

drops X? from the existing pool specification and prints the results of the test.

Cross-references

See “Coefficient Tests” on page 142 of the *User’s Guide II* for further discussion of testing coefficients.

See also [Pool::testadd](#) (p. 326) and [Pool::wald](#) (p. 334).

tsls	Pool Methods
-------------	------------------------------

Two-stage least squares.

Syntax

```
pool_name.tsls(options) y x1 [x2 x3 ...] [@cxreg w1 w2 ...] [@perreg w3 w4 ...] [@inst  
z1 z2 ...] [@cxinst z3 z4 ...] [@perinst z5 z6 ...]
```

Type the name of the dependent variable followed by one or more 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 pool variables that have different coefficients for each cross-section member of the pool. If there is a cross-section specific regressor list, the two lists must be separated by “@CXREG”. The

third list, if provided, should contain pool variables that have different coefficients for each period. The list should be separated from the previous lists by “@PERREG”.

You may include AR terms as regressors in either the common or cross-section specific lists. AR terms are, however, not allowed for some estimation methods. MA terms are not supported.

Instruments should be specified in one of three lists. The “@INST” list should contain instruments that are common across all cross-sections and periods. The “@CXINST” should contain instruments that differ across cross-sections, while the “@PERINST” list specifies instruments that differ across periods.

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 corresponding instrument list. A constant is included in the common instrumental list if not explicitly specified.

Options

General options

m = integer	Set maximum number of iterations.
c = number	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
deriv = keyword	Set derivative methods. The argument <i>keyword</i> 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.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
s	Use the current coefficient values in “C” as starting values for equations with AR or MA terms (see also param (p. 761)).
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”.

<code>cx = arg</code>	Cross-section effects. For fixed effects estimation, use “cx = f”; for random effects estimation, use “cx = r”.
<code>per = arg</code>	Period effects. For fixed effects estimation, use “cx = f”; for random effects estimation, use “cx = r”.
<code>wgt = arg</code>	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”).
<code>cov = arg</code>	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”).
<code>keepwghts</code>	Keep full set of GLS weights used in estimation with object, if applicable (by default, only small memory weights are saved).
<code>rancalc = arg</code> (<i>default</i> = “sa”)	Random component method: Swamy-Arora (“rancalc = sa”), Wansbeek-Kapteyn (“rancalc = wk”), Wallace-Hussain (“rancalc = wh”).
<code>nodf</code>	Do not perform degree of freedom corrections in computing coefficient covariance matrix. The default is to use degree of freedom corrections.
<code>coef = arg</code>	Specify the name of the coefficient vector (if specified by list); the default is to use the “C” coefficient vector.
<code>iter = arg</code> (<i>default</i> = “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 param (p. 761)).
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”.
p	Print estimation results.

Examples

```
pool1.tsls y? c x? @inst z?
```

estimates TSLS on the pool specification using common instruments Z?

Cross-references

See [“Two-stage Least Squares” on page 37](#) and [“Two-Stage Least Squares” on page 309](#) of the *User’s Guide II* for details on two-stage least squares estimation in single equations and systems, respectively. [“Instrumental Variables” on page 502](#) of the *User’s Guide II* discusses estimation using pool objects, while [“Instrumental Variables Estimation” on page 544](#) of the *User’s Guide II* discusses estimation in panel structured workfiles.

See also [Pool::ls](#) (p. 311).

updatecoefs	Pool Procs
-------------	----------------------------

Update coefficient object values from pool object.

Copies coefficients from the pool into the appropriate coefficient vector.

Syntax

```
pool_name.updatecoef
```

Follow the name of the pool object by a period and the keyword `updatecoef`.

Examples

```
pool1.ls y? c x1? x2? x3?  
pool2.ls z? c z1? z2? z3?  
pool1.updatecoef
```

places the coefficients from POOL1 in the default coefficient vector C.

Cross-references

See also [Coef::coef](#) (p. 16).

uroot	Pool Views
-------	----------------------------

Carries out unit root tests on a pool series.

When used with a pool series, the procedure will perform panel unit root testing. 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.

Note that simulation evidence suggests that in various settings (for example, small T), Hadri's panel unit root test experiences significant size distortion in the presence of autocorrelation when there is no unit root. In particular, the Hadri test appears to over-reject the null of stationarity, and may yield results that directly contradict those obtained using alternative test statistics (see Hlouskova and Wagner (2006) for discussion and details).

Syntax

```
pool_name.uroot(options) pool_series
```

Enter the pool object name followed by a period, the keyword, and 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:

const (default)	Include a constant in the test equation.
trend	Include a constant and a linear time trend in the test equation.
none	Do not include a constant or time trend (only available for the ADF and PP tests).
dif = integer (default = 0)	Order of differencing of the series prior to running the test. Valid values are {0, 1, 2}.

For panel testing, you may use one of the following keywords to specify the test:

sum (default)	Summary of the first five panel unit root tests (where applicable).
llc	Levin, Lin, and Chu.
breit	Breitung.

ips	Im, Pesaran, and Shin.
adf	Fisher - ADF.
pp	Fisher - PP.
hadri	Hadri.

Panel Specification Options

The following additional panel specific options are available:

balance	Use balanced (across cross-sections or series) data when performing test.
hac = <i>arg</i> (<i>default</i> = "bt")	Method of estimating the frequency zero spectrum: "bt" (Bartlett kernel), "pr" (Parzen kernel), "qs" (Quadratic Spectral kernel). Applicable to "Summary", LLC, Fisher-PP, and Hadri tests.
band = <i>arg</i> , b = <i>arg</i> (<i>default</i> = "nw")	Method of selecting the bandwidth: "nw" (Newey-West automatic variable bandwidth selection), "a" (Andrews automatic selection), <i>number</i> (user-specified common bandwidth), <i>vector_name</i> (user-specified individual bandwidth). Applicable to "Summary", LLC, Fisher-PP, and Hadri tests.

`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,

$$default = \begin{cases} 1 & \text{if } (T_{\min} \leq 60) \\ 2 & \text{if } (60 < T_{\min} \leq 100) \\ 4 & \text{if } (T_{\min} > 100) \end{cases}$$

where T_{\min} is the length of the shortest cross-section or series, otherwise *default* = “a”.

Applicable to “Summary”, LLC, Breitung, IPS, and Fisher-ADF tests.

`info = arg`
(*default* = “sic”) Information criterion to use when computing automatic lag length selection: “aic” (Akaike), “sic” (Schwarz), “hqc” (Hannan-Quinn).
Applicable to “Summary”, LLC, Breitung, IPS, and Fisher-ADF tests.

`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)

$$default = \text{int}(\min_i(12, T_i/3) \cdot (T_i/100)^{0.25})$$

where T_i is the length of the cross-section or series.

Other options

`p` Print output from the test.

Examples

```
Pool1.uroot(llc,trend) gdp?
```

performs the LLC panel unit root test with exogenous individual trends and individual effects on pool series GDP?

```
Pool1.uroot(IPS, const, maxlag=4, info=AIC) inv?
```

performs the IPS panel unit root test on pool series INV?. The test includes individual effects, lag will be chosen by AIC from maximum lag of three.

```
Pool1.uroot(sum, const, lag=3, hac=pr,b=2.3) mm?
```

performs a summary of the panel unit root tests on the pool series `MM?`. The test equation includes a constant term and three lagged first-difference terms. The frequency zero spectrum is estimated using kernel methods (with a Parzen kernel), and a bandwidth of 2.3.

Cross-references

See [“Unit Root Tests” on page 88](#) of the *User’s Guide II* for discussion of standard unit root tests performed on a single series, and [“Panel Unit Root Tests” on page 100](#) of the *User’s Guide II* for discussion of unit roots tests performed on panel structured workfiles, groups of series, or pooled data.

See also [`Pool::coint` \(p. 300\)](#).

wald	Pool Views
------	----------------------------

Wald coefficient restriction test.

The `wald` view carries out a Wald test of coefficient restrictions for a pool object.

Syntax

`pool_name.wald restrictions`

Enter the pool object name, followed by a period, and the keyword. You must provide a list of the coefficient restrictions, with joint (multiple) coefficient restrictions separated by commas.

Options

<code>p</code>	Print the test results.
----------------	-------------------------

Examples

```
pool panel us uk jpn
panel.ls cons? c inc? @cxreg 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 145](#) of the *User’s Guide II* for a discussion of Wald tests.

See also [`Pool::cellipse` \(p. 298\)](#), [`Pool::testdrop` \(p. 327\)](#), [`Pool::testadd` \(p. 326\)](#).

write[Pool Procs](#)

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

```
pool_name.write(options) [path\filename] pool_series1 [pool_series2 pool_series3 ...]
```

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.

Note that EViews cannot, at present, write into an existing file. The file that you select will, if it exists, 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 = <i>string</i>	Specify text string for NAs. Default is “NA”.
names (<i>default</i>) / nonames	[Write / Do not write] series names.
id	Write dates/obs and cross-section identifiers.
d = <i>arg</i>	Specify delimiter (<i>default</i> is tab): “s” (space), “c” (comma).
t	Write by series. Default is to write by obs with series in columns.

Spreadsheet (Lotus, Excel) files

<i>letter_number</i>	Coordinate of the upper-left cell containing data.
<i>names (default) / nonames</i>	[Write / Do not write] series names.
<i>id</i>	Write dates/obs and cross-section identifiers.
<i>dates = arg</i>	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).
<i>t</i>	Write by series. Default is to write by obs with series in columns.

Pooled data writing

<i>bycross (default) / byper</i>	Stack pool data by [cross-section / date or period].
----------------------------------	--

Examples

```
pool1.write(t=txt,na=.,d=c,id) a:\dat1.csv gdp? edu?
```

Writes 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 and cross-section identifiers are written together with the series names.

```
pool1.write(t=txt,na=.,d=c,id) dat1.csv gdp? edu?
```

writes the same file in the default directory.

```
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 observation, and are listed by period/time.

Cross-references

See [“Exporting to a Spreadsheet or Text File” on page 105](#) of the *User’s Guide I* for a discussion. Pool writing is discussed in [“Exporting Pooled Data” on page 478](#) of the *User’s Guide II*.

See also [pagesave \(p. 752\)](#) and [Pool::read \(p. 319\)](#).

Rowvector

Row vector. (One dimensional array of numbers).

Rowvector Declaration

[rowvector](#).....declare rowvector object ([p. 343](#)).

There are several ways to create a rowvector object. First, you can enter the `rowvector` keyword (with an optional dimension) followed by a name:

```
rowvector scalarmat
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:

```
rowvector(10) y=3
rowvector z=results
```

Rowvector Views

[label](#)label information for the rowvector ([p. 340](#)).

[sheet](#).....spreadsheet view of the vector ([p. 346](#)).

[stats](#).....(trivial) descriptive statistics ([p. 347](#)).

Rowvector Graph Views

Graph creation types are discussed in detail in “[Graph Creation Commands](#)” on [page 601](#).

[bar](#).....bar graph of each column (element) of the data against the row index ([p. 609](#)).

[boxplot](#)boxplot graph ([p. 613](#)).

[distplot](#)distribution graph ([p. 615](#)).

[dot](#).....dot plot graph ([p. 622](#)).

[errbar](#)error bar graph view ([p. 626](#)).

[pie](#).....pie chart view ([p. 633](#)).

[qqplot](#).....quantile-quantile graph ([p. 636](#)).

[scat](#).....scatter diagrams of the columns of the rowvector ([p. 640](#)).

[scatmat](#)matrix of all pairwise scatter plots ([p. 644](#)).

[scatpair](#).....scatterplot pairs graph ([p. 647](#)).

[seasplot](#).....seasonal line graph of the columns of the rowvector ([p. 651](#)).

[spike](#).....spike graph ([p. 652](#)).

[xybar](#)XY bar graph ([p. 659](#)).

[xypair](#)XY pairs graph ([p. 664](#)).

Rowvector Procs

- [displayname](#) set display name (p. 338).
- [fill](#) fill elements of the vector (p. 339).
- [read](#) import data from disk (p. 341).
- [setformat](#) set the display format for the vector spreadsheet (p. 343).
- [setindent](#) set the indentation for the vector spreadsheet (p. 344).
- [setjust](#) set the justification for the vector spreadsheet (p. 345).
- [setwidth](#) set the column width in the vector spreadsheet (p. 346).
- [write](#) export data to disk (p. 347).

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)
```

Rowvector Entries

The following section provides an alphabetical listing of the commands associated with the “Rowvector” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

displayname	Rowvector Procs
-------------	---------------------------------

Display name for rowvector objects.

Attaches a display name to a rowvector object which may be used to label output in tables and graphs in place of the standard rowvector object name.

Syntax

```
vector_name.displayname display_name
```

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in rowvector object names.

Examples

```
hrs.displayname Hours Worked
hrs.label
```

The first line attaches a display name “Hours Worked” to the rowvector object HRS, and the second line displays the label view of HRS, including its display name.

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels and display names.

See also [Rowvector::label \(p. 340\)](#).

fill	Rowvector Procs
------	---------------------------------

Fill a rowvector object with specified values.

Syntax

```
vector_name.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.*

Running out of values before the object is completely filled is not an error; the remaining cells or observations will be unaffected, unless the “1” option is specified. If, however, you list more values than the object can hold, EViews will not modify any observations and will return an error message.

Options

1	Loop repeatedly over the list of values as many times as it takes to fill the object.
<i>o</i> = integer (default = 1)	Fill the object from the specified element. Default is the first element.

Examples

The following example declares a four element rowvector MC, initially filled with zeros. The second line fills MC with the specified values and the third line replaces from column 3 to the last column with -1.

```
rowvector(4) mc
```

```
mc.fill 0.1, 0.2, 0.5, 0.5
mc.fill(o=3,1) -1
```

Cross-references

See [Chapter 18. “Matrix Language,” on page 627](#) of the *User’s Guide I* for a detailed discussion of vector and matrix manipulation in EViews.

label	Rowvector Views Rowvector Procs
-------	---

Display or change the label view of a rowvector object, including the last modified date and display name (if any).

As a procedure, `label` changes the fields in the rowvector label.

Syntax

```
vector_name.label
vector_name.label(options) [text]
```

Options

The first version of the command displays the label view of the rowvector. The second version may be used to modify the label. 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 <i>text</i> .
s	Sets the source field to <i>text</i> .
u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of rowvector RV1 with “Data from CPS 1988 March File”:

```
rv1.label(r)
rv1.label(r) Data from CPS 1988 March File
```

To append additional remarks to RV1, and then to print the label view:

```
rv1.label(r) Log of hourly wage
rv1.label(p)
```


To clear and then set the units field, use:

```
rv1.label(u) Millions of bushels
```

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels.

See also [Rowvector::displayname \(p. 338\)](#).

read	Rowvector Procs
------	---------------------------------

Import data from a foreign disk file into a rowvector.

May be used to import data into an existing workfile from a text, Excel, or Lotus file on disk.

Syntax

```
vector_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. 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.

Options

File type options

t = dat, txt	ASCII (plain text) files.
t = wk1, wk3	Lotus spreadsheet files.
t = xls	Excel spreadsheet files.

If you do not specify the “t” option, EViews uses the file name extension to determine the file type. If you specify the “t” option, the file name extension will not be used to determine the file type.

Options for ASCII text files

na = text	Specify text for NAs. Default is “NA”.
d = t	Treat tab as delimiter (note: you may specify multiple delimiter options). The <i>default</i> is “d = c” only.
d = c	Treat comma as delimiter.
d = s	Treat space as delimiter.
d = a	Treat alpha numeric characters as delimiter.
custom = symbol	Specify symbol/character to treat as delimiter.

<code>mult</code>	Treat multiple delimiters as one.
<code>rect (default) / norect</code>	[Treat / Do not treat] file layout as rectangular.
<code>skipcol = integer</code>	Number of columns to skip. Must be used with the “rect” option.
<code>skiprow = integer</code>	Number of rows to skip. Must be used with the “rect” option.
<code>comment = symbol</code>	Specify character/symbol to treat as comment sign. Everything to the right of the comment sign is ignored. Must be used with the “rect” option.
<code>singlequote</code>	Strings are in single quotes, not double quotes.
<code>dropstrings</code>	Do not treat strings as NA; simply drop them.
<code>negparen</code>	Treat numbers in parentheses as negative numbers.
<code>allowcomma</code>	Allow commas in numbers (note that using commas as a delimiter takes precedence over this option).

Options for spreadsheet (Lotus, Excel) files

<code>letter_number (default = “b2”)</code>	Coordinate of the upper-left cell containing data.
<code>s = sheet_name</code>	Sheet name for Excel 5–8 Workbooks.

Examples

```
rv1.read(t=dat, na=.) a:\mydat.raw
```

reads data into rowvector RV1 from an ASCII file MYDAT.RAW in the A: drive. The data in the file are listed by row, and the missing value NA is coded as a “.” (dot or period).

```
rv1.read(a2, s=sheet3) cps88.xls
```

reads data into rowvector RV1 from an Excel file CPS88 in the default directory. The upper left data cell is A2, and the data is read from a sheet named SHEET3.

```
rv2.read(a2, s=sheet2) "\\network\dr 1\cps91.xls"
```

reads the Excel file CPS91 into rowvector RV1 from the network drive specified in the path.

Cross-references

See [“Importing Data” on page 95](#) of the *User’s Guide I* for a discussion and examples of importing data from external files.

For powerful, easy-to-use tools for reading data into a new workfile, see “[Creating a Workfile by Reading from a Foreign Data Source](#)” on page 41 of the *User’s Guide I*, [pageload](#) (p. 750), and [wfopen](#) (p. 802).

See also [Rowvector::write](#) (p. 347).

rowvector	Rowvector Declaration
------------------	---------------------------------------

Declare a rowvector object.

The `rowvector` command declares and optionally initializes a (row) vector object.

Syntax

```
rowvector(n1) vector_name
```

```
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

```
rowvector rvec1
```

```
rowvector(20) coefvec = 2
```

```
rowvector newcoef = coefvec
```

RVEC1 is a row vector of size one with value 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 the same values as COEFVEC.

Cross-references

See also [Coef::coef](#) (p. 16) and [Vector::vector](#) (p. 587).

setformat	Rowvector Procs
------------------	---------------------------------

Set the display format for cells in a rowvector object spreadsheet view.

Syntax

```
vector_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.

For rowvectors, `setformat` operates on all of the cells in the rowvector.

To format numeric values, you should use one of the following format specifications:

<code>g[.precision]</code>	significant digits
<code>f[.precision]</code>	fixed decimal places
<code>c[.precision]</code>	fixed characters
<code>e[.precision]</code>	scientific/float
<code>p[.precision]</code>	percentage
<code>r[.precision]</code>	fraction

In order to set a format that groups digits into thousands, place a “t” after a specified format. For example, to obtain a fixed number of decimal places, with commas used to separate thousands, use “`ft[.precision]`”. To obtain a fixed number of characters with a period used to separate thousands, use “`ct[.precision]`”.

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)`”).

Examples

To set the format for all cells in the rowvector to fixed 5-digit precision, simply provide the format specification:

```
rv1.setformat f.5
```

Other format specifications include:

```
rv1.setformat f(.7)
rv1.setformat e.5
```

Cross-references

See [Rowvector::setWidth \(p. 346\)](#), [Rowvector::setindent \(p. 344\)](#) and [Rowvector::setjust \(p. 345\)](#) for details on setting spreadsheet widths, indentation and justification.

setindent	Rowvector Procs
-----------	---------------------------------

Set the display indentation for cells in a rowvector object spreadsheet view.

Syntax

```
vector_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 spreadsheet (the

EViews spreadsheet default font at the time the spreadsheet 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 spreadsheet view is created.

For rowvectors, `setindent` operates on all of the cells in the vector.

Examples

To set the indentation for all the cells in a matrix object:

```
rv1.setindent 2
```

Cross-references

See [Rowvector::setWidth \(p. 346\)](#) and [Rowvector::setjust \(p. 345\)](#) for details on setting spreadsheet widths and justification.

setjust	Rowvector Procs
---------	---------------------------------

Set the display justification for cells in a rowvector spreadsheet view.

Syntax

```
vector_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.

For rowvectors, `setjust` operates on all of the cells in the vector.

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 taken from the Global Defaults for spreadsheet views.

Examples

```
rv1.setjust middle
```

sets the vertical justification to the middle.

```
rv1.setjust top left
```

sets the vertical justification to top and the horizontal justification to left.

Cross-references

See [Rowvector::setwidth](#) (p. 346) and [Rowvector::setindent](#) (p. 344) for details on setting spreadsheet widths and indentation.

setwidth	Rowvector Procs
----------	---------------------------------

Set the column width for all columns in a rowvector object spreadsheet.

Syntax

```
vector_name.setwidth width_arg
```

where *width_arg* specifies the width unit value. The width unit is computed from representative characters in the default font for the current spreadsheet (the EViews spreadsheet default font at the time the spreadsheet 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

```
rv1.setwidth 12
```

sets the width of all columns in rowvector RV1 to 12 width units.

Cross-references

See [Rowvector::setindent](#) (p. 344) and [Rowvector::setjust](#) (p. 345) for details on setting spreadsheet indentation and justification.

sheet	Rowvector Views
-------	---------------------------------

Spreadsheet view of a rowvector object.

Syntax

```
vector_name.sheet(options)
```

Options

p	Print the spreadsheet view.
---	-----------------------------

Examples

```
rv1.sheet(p)
```

displays and prints the spreadsheet view of rowvector RV1.

stats	Rowvector Views
-------	---------------------------------

Descriptive statistics.

Computes and displays a table of means, medians, maximum and minimum values, standard deviations, and other descriptive statistics for a rowvector.

The `stats` command computes the statistics for each column. Note that in the case of a rowvector, this will be for a single observation.

Syntax

```
vector_name.stats(options)
```

Options

p	Print the stats table.
---	------------------------

Examples

```
rv1.stats
```

displays the descriptive statistics view of rowvector RV1.

Cross-references

See [“Descriptive Statistics & Tests” on page 306](#) and [page 379](#) of the *User’s Guide I* for a discussion of the descriptive statistics views of series and groups.

write	Rowvector Procs
-------	---------------------------------

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

```
vector_name.write(options) [path\filename]
```

Follow the name of the rowvector object by a period, the keyword, and the name for the output file. 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. The entire rowvector will be exported.

Note that EViews cannot, at present, write into an existing file. The file that you select will, if it exists, be replaced.

Options

Options are specified in parentheses after the keyword and are used to specify the format of the output file.

File type

<code>t = dat, txt</code>	ASCII (plain text) files.
<code>t = wk1, wk3</code>	Lotus spreadsheet files.
<code>t = xls</code>	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

<code>na = string</code>	Specify text string for NAs. Default is “NA”.
<code>d = arg</code>	Specify delimiter (<i>default</i> is tab): “s” (space), “c” (comma).

Spreadsheet (Lotus, Excel) files

<code>letter_number</code>	Coordinate of the upper-left cell containing data.
----------------------------	--

Examples

```
rv1.write(t=txt,na=.) a:\dat1.csv
```

writes the rowvector RV1 into an ASCII file named DAT1.CSV on the A: drive. NAs are coded as “.” (dot).

```
rv1.write(t=txt,na=.) dat1.csv
```

writes the same file in the default directory.

```
rv1.write(t=xls) "\\network\drive a\results"
```

saves the contents of RV1 in an Excel file “Results.xls” in the specified directory.

Cross-references

See [“Exporting to a Spreadsheet or Text File” on page 105](#) of the *User’s Guide I* for a discussion.

See also [pagesave](#) (p. 752) and [Rowvector::read](#) (p. 341).

Sample

Sample of observations. Description of a set of observations to be used in operations.

Sample Declaration

`sample`declare sample object (p. 351).

To declare a sample object, use the keyword `sample`, followed by a name and a sample string:

```
sample mysample 1960:1 1990:4
sample altsample 120 170 300 1000 if x>0
```

Sample Views

`label`label information for the sample (p. 350).

Sample Procs

`displayname`.....set display name (p. 349).

`set`reset the sample range (p. 352).

Sample Example

To change the observations in a sample object, you can use the `set` proc:

```
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:

```
smpl mysample
equation eql.ls y x c
```

Sample Entries

The following section provides an alphabetical listing of the commands associated with the “Sample” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

displayname	Sample Procs
-------------	------------------------------

Display name for sample objects.

Attaches a display name to a sample object which may be used to label output in place of the standard sample object name.

Syntax

```
sample_name.displayname display_name
```

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in sample object names.

Examples

```
sm1.displayname Annual Sample
sm1.label
```

The first line attaches a display name “Annual Sample” to the sample object SM1, and the second line displays the label view of SM1, including its display name.

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels and display names.

See also [Sample::label \(p. 350\)](#).

label	Sample Views Sample Procs
-------	---

Display or change the label view of a sample object, including the last modified date and display name (if any).

As a procedure, `label` changes the fields in the sample object label.

Syntax

```
sample_name.label
sample_name.label(options) [text]
```

Options

The first version of the command displays the label view of the sample object. The second version may be used to modify the label. 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 <i>text</i> .
s	Sets the source field to <i>text</i> .
u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of the sample SP1 with “1988 March”

```
sp1.label(r)
sp1.label(r) 1988 March
```

To append additional remarks to SP1, and then to print the label view:

```
sp1.label(r) if X is greater than 3
sp1.label(p)
```

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels.

See also [Sample::displayname \(p. 349\)](#).

sample	Sample Declaration
--------	------------------------------------

Declare a sample object.

The `sample` statement declares, and optionally defines, a sample object.

Syntax

```
sample smpl_name [smpl_statement]
```

Follow the `sample` keyword with a name for the sample object and a sample statement. 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 [Sample::set \(p. 352\)](#) procedure.

Examples

```
sample ss
```

declares a sample object named SS and sets it to the current workfile sample.

```
sample s2 1974q1 1995q4
```

declares a sample object named S2 and sets it from1974Q1 to 1995Q4.

```
sample fe_bl @all if gender=1 and race=3
smpl fe_bl
```

The first line declares a sample FE_BL that includes observations where GENDER = 1 and RACE = 3. The second line sets the current sample to FE_BL.

```
sample sf @last-10 @last
```

declares a sample object named SF and sets it to the last 10 observations of the current workfile range.

```
sample s1 @first 1973q1
s1.set 1973q2 @last
```

The first line declares a sample object named S1 and sets it from the beginning of the workfile range to 1973Q1. The second line resets S1 from 1973Q2 to the end of the workfile range.

Cross-references

See [“Samples” on page 86](#) and [“Dates” on page 704](#) of the *User’s Guide I* for a discussion of using dates and samples in EViews.

See also [Sample::set \(p. 352\)](#) and [smp1 \(p. 780\)](#).

set	Sample Procs
-----	------------------------------

Set the sample in a sample object.

The `set` procedure resets the sample of an existing sample object.

Syntax

```
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 86](#) of the *User’s Guide I* for a discussion of samples in EViews.

See also [Sample::sample \(p. 351\)](#) and [smp1 \(p. 780\)](#).

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. 353](#)).

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:

```
scalar pi=3.14159
scalar shape=beta(7)
show shape
```

or you can declare a scalar and use it in an expression:

```
scalar inner=@transpose(mydata)*mydata
series x=1/@sqrt(inner)*y
```

Scalar Entries

The following section provides an alphabetical listing of the commands associated with the “[Scalar](#)” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

scalar	Scalar Declaration
---------------	------------------------------------

Declare a scalar object.

The `scalar` command declares a scalar object and optionally assigns a value.

Syntax

```
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 initialized with a value of zero.

Examples

```
scalar alpha
```

declares a scalar object named ALPHA with value zero.

```
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.

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. 368).
genr create numeric series object (p. 370).
series declare numeric series object (p. 375).

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

bdstest BDS independence test (p. 357).
correlogram correlogram, autocorrelation and partial autocorrelation functions (p. 363).
edftest empirical distribution function tests (p. 365).
freq one-way tabulation (p. 367).
hist descriptive statistics and histogram (p. 370).
label label information for the series (p. 372).
sheet spreadsheet view of the series (p. 383).
statby statistics by classification (p. 387).
stats descriptive statistics table (p. 389).
testby equality test by classification (p. 391).
teststat simple hypothesis tests (p. 392).
uroot unit root test on an ordinary or panel series (p. 396).

Series Graph Views

Graph creation types are discussed in detail in “Graph Creation Commands” on page 601.

area area graph of the series (p. 603).
bar bar graph of the series (p. 609).
boxplot boxplot graph (p. 613).
distplot distribution graph (p. 615).
dot dot plot graph (p. 622).
line line graph of the series (p. 630).

qqplot quantile-quantile plot (p. 636).
seasplot seasonal line graph (p. 651).
spike spike graph (p. 652).

Series Procs

bpf compute and display band-pass filter (p. 358).
classify recode series into classes defined by a grid, specified limits, or quantiles (p. 361).
displayname set display name (p. 364).
fill fill the elements of the series (p. 366).
hpf Hodrick-Prescott filter (p. 371).
map assign or remove value map setting (p. 373).
resample resample from the observations in the series (p. 373).
seas seasonal adjustment for quarterly and monthly time series (p. 375).
setconvert set default frequency conversion method (p. 377).
setformat set the display format for the series spreadsheet (p. 379).
setindent set the indentation for the series spreadsheet (p. 382).
setjust set the justification for the series spreadsheet (p. 382).
setwidth set the column width in the series spreadsheet (p. 383).
smooth exponential smoothing (p. 384).
sort change display order for series spreadsheet (p. 386).
stom convert a series to a vector (p. 389).
stomna convert a series to a vector without dropping NAs (p. 390).
tramoseats seasonal adjustment using Tramo/Seats (p. 393).
x11 seasonal adjustment by Census X11 method for quarterly and monthly time series (p. 400).
x12 seasonal adjustment by Census X12 method for quarterly and monthly time series (p. 402).

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)
```

Series Entries

The following section provides an alphabetical listing of the commands associated with the “[Series](#)” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

bdstest	Series Views
----------------	------------------------------

Perform BDS test for independence.

The BDS test is a Portmanteau test for time-based dependence in a series. The test may be used for testing against a variety of possible deviations from independence, including linear dependence, non-linear dependence, or chaos.

Syntax

```
series_name.bds(options)
```

Options

$m = \textit{arg}$ (<i>default</i> = “p”)	Method for calculating ϵ : “p” (fraction of pairs), “v” (fixed value), “s” (standard deviations), “r” (fraction of range).
$e = \textit{number}$	Value for calculating ϵ .
$d = \textit{integer}$	Maximum dimension.

<code>b = integer</code>	Number of repetitions for bootstrap <i>p</i> -values. If option is omitted, no bootstrapping is performed.
<code>o = arg</code>	Name of output vector for final BDS <i>z</i> -statistics.
<code>p</code>	Print output.

Cross-references

See [“BDS Test” on page 327](#) of the *User’s Guide I* for additional discussion.

boxplotby	Series Views
------------------	------------------------------

Display the boxplots of a series classified into categories.

The `boxplotby` command is no longer supported. See [boxplot \(p. 613\)](#) for the replacement categorical graph command.

bpf	Series Procs
------------	------------------------------

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_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

<code>type = arg</code> (<i>default</i> = “bk”)	Specify the type of band-pass filter: “bk” is the Baxter-King fixed length symmetric filter, “cfix” is the Christiano-Fitzgerald fixed length symmetric filter, “cfasym” is the Christiano-Fitzgerald full sample asymmetric filter.
<code>low = number,</code> <code>high = number</code>	<p>Low (P_L) and high (P_H) values for the cycle range to be passed through (specified in periods of the workfile frequency).</p> <p>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”.</p> <p>The arguments must satisfy $2 \leq P_L < P_H$. The corresponding frequency range to be passed through will be $(2\pi/P_H, 2\pi/P_L)$.</p>
<code>lag = integer</code>	Fixed lag length (positive integer). Sets the fixed lead/lag length for fixed length filters (“type = bk” or “type = cfix”). Must be less than half the sample size. Defaults to the workfile equivalent of 3 years for semi-annual to daily workfiles; otherwise sets “lag = 3”.
<code>iorder = [0,1]</code> (<i>default</i> = 0)	<p>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.</p> <p>The integration order is only used in the computation of Christiano-Fitzgerald filter weights (“type = cfix” or “type = cfasym”). When “iorder = 1”, the filter weights are constrained to sum to zero.</p>
<code>detrend = arg</code> (<i>default</i> = “n”)	<p>Detrending method for Christiano-Fitzgerald filters (“type = cfix” or “type = cfasym”).</p> <p>You may select the default argument “n” for no detrending, “c” to demean, or “t” to remove a constant and linear trend.</p> <p>You may use the argument “d” to remove drift, if the option “iorder = 1” is also specified.</p>
<code>nogain</code>	Suppresses plotting of the frequency response (gain) function for fixed length symmetric filters (“type = bk” or “type = cfix”). By default, EViews will plot the gain function.

`noncyc = arg` 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.

`w = arg` 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 a matrix 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 z_t may 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 $t = q + 1, \dots, n - q$.

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. The filtered series may be computed as:

$$z_t = \sum_{c=1}^T w(r, c) y_c \quad r = 1, \dots, T$$

where 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.

`p` Print the graph.

Examples

Suppose we are working in a quarterly workfile and we issue the following command:

```
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:

```
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:

```
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 361 of the *User’s Guide I*. See also [Series::hpf](#) (p. 371).

cdfplot	Series Views
----------------	------------------------------

Empirical distribution functions.

The `cdfplot` command is no longer supported. See [distplot](#) (p. 615).

classify	Series Procs
-----------------	------------------------------

Recode series into classes defined by a grid, specified limits, or quantiles.

Syntax

```
series_name.classify(options) spec @ outname [mapname]
```

Follow the `classify` keyword with any desired options, the “@”-sign, the name to be given the output series, and optionally the name for a valmap object describing the classification.

The form for the specification *spec* will depend on which of the four supported methods for classification is employed (using the “method=” option).

- If the default “method = step” is employed, EViews will construct the classification using the set of intervals of size *step* from *start* through *end*. The *spec* specification is of the form

```
stepsize start end
```

where *stepsize* is a positive numeric value and *start* and *end* are numeric values. If *start* or *end* are explicitly set to NAs, EViews will use the corresponding minimum and maximum value of the data extended by 5% (e.g., $0.95 \cdot \min$ or $1.05 \cdot \max$).

- If “method = bins”, EViews will construct the classification by dividing the range between *start* and *end* into a specified number of bins. The specification is of the form:

nbins start end

where *nbins* is the integer number of bins. Note that depending upon whether you have selected left or right-closed intervals (using the “rightclosed” option), observations with values equal to the *start* or *end* may fall out-of-range.

- Using “method = limits” specifies a classification using bins defined by a set of limit values. The *spec* is given by:

arg1 [arg2 arg3 ...]

where the arguments are limit values or EViews vectors containing limit values. Note that there must be at least two limit values and that the values *need not* be provided in ascending or descending order.

- If “method = quants” is given, EViews uses the specified number of quantiles for the data, specified as an integer value. The specification is:

nquants

where *nquants* is the integer for the number of quantiles. For deciles you should set *nquants* = 10, for quartiles, *nquants* = 4.

Options

<i>method = arg</i> (<i>default</i> = “step”)	Method for classification values: “step” – create a grid from <i>start</i> through <i>end</i> using the <i>stepsize</i> ; “bins” – create bins by dividing the region from <i>start</i> to <i>end</i> into a specified number of bins; “quants” – create bins using the quantile values; “limits” – create bins using the specified limit points.
<i>rightclosed</i>	Bins formed using right-closed intervals. <i>x</i> is defined to be in the bin from <i>a</i> to <i>b</i> if $a < x \leq b$.
<i>rangeerr</i>	Generate error if data value is found outside of defined bins. The default is to classify out-of-range values as NAs.
<i>q = arg</i> (<i>default</i> = “r”)	Quantile calculation method. “b” (Blom), “r” (Rankit-Cleveland), “o” (Ordinary), “t” (Tukey), “v” (van der Waerden), “g” (Gumbel). Only relevant where “method = quants”.
<i>encode = arg</i> (<i>default</i> = “index”)	Encoding method for output series: “index” – encode as integers from 0 to <i>k</i> where <i>k</i> is the number of bins, where the 0 is reserved for NA encoding if “keepna” is specified; “left” – encode using the left-most value defining the bin; “right” – encode using the right-most value defining the bin; “mid” – encode using the midpoint of the bin.

keepna Classify NA values as 0 (for “encode = index” only).

Examples

```
api5b.classify 100 200 @ api5b_ct api5b_mp
```

classifies the values of API5B into bins of width 100 starting at 200 and ending at the data maximum times 1.05. The classification results are saved in the series API5B_CT with associated map API5B_MP.

```
api5b.classify(encode=right) 100 200 1100 @ api5b_ct1
```

classifies API5B into bins of size 100 from 200 through 1100. The output series API5B_CT1 will have values taken from the right endpoints of the classification intervals.

```
api5b.classify(method=bins, rightclosed, rangeerr) 9 200 1100 @
api5b_ct2 api5b_mp2
```

defines 9 equally sized bins starting at 200 and ending at 1100, and classifies the data into the series API5B_CT2 with map API5B_MP2. The bins are closed on the right, and out-of-range values will generate an error.

```
api5b.classify(method=quants, q=g, keepna) 4 @ api5b_ct3
```

classifies the values of API5B into quartiles (using the Gumbel definition) in the series API5B_CT3. NA values for API5B will be encoded as 0 in the output series.

Cross-references

See [“Generate by Classification” on page 333](#) of the *User’s Guide I* for additional discussion.

correl	Series Views
--------	------------------------------

Display autocorrelation and partial correlations.

Displays the autocorrelation and partial correlation functions of the series, together with the *Q*-statistics and *p*-values associated with each lag.

Syntax

```
series_name.correl(n, options)
```

You must specify the largest lag *n* to use when computing the autocorrelations.

Options

p	Print the correlograms.
---	-------------------------

Examples

```
ser1.correl(24)
```

Displays the correlograms of the SER1 series for up to 24 lags.

Cross-references

See [“Autocorrelations \(AC\)” on page 325](#) and [“Partial Autocorrelations \(PAC\)” on page 326](#) of the *User’s Guide I* for a discussion of autocorrelation and partial correlation functions, respectively.

displayname	Series Procs
-------------	------------------------------

Display name for series objects.

Attaches a display name to a series object which may be used to label output in tables and graphs in place of the standard series object name.

Syntax

`series_name.displayname display_name`

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in series object names.

Examples

```
hrs.displayname Hours Worked
hrs.label
```

The first line attaches a display name “Hours Worked” to the series 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 [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels and display names.

See also [Series::label \(p. 372\)](#) and [Series::label \(p. 372\)](#).

edftest	Series Views
---------	------------------------------

Computes goodness-of-fit tests based on the empirical distribution function.

Syntax

```
series_name.edftest(options)
```

Options

General Options

<code>dist = arg</code> (<i>default</i> = "normal")	Distribution to test: "normal" (Normal distribution), "chisq" (Chi-square distribution), "exp" (Exponential distribution), "xmax" (Extreme Value - Type I maximum), "xmin" (Extreme Value Type I minimum), "gamma" (Gamma), "logit" (Logistic), "pareto" (Pareto), "uniform" (Uniform).
<code>p1 = number</code>	Specify the value of the first parameter of the distribution (as it appears in the dialog). If this option is not specified, the first parameter will be estimated.
<code>p2 = number</code>	Specify the value of the second parameter of the distribution (as it appears in the dialog). If this option is not specified, the second parameter will be estimated.
<code>p3 = number</code>	Specify the value of the third parameter of the distribution (as it appears in the dialog). If this option is not specified, the third parameter will be estimated.
<code>p</code>	Print test results.

Estimation Options

The following options apply if iterative estimation of parameters is required:

<code>b</code>	Use Berndt-Hall-Hausman (BHHH) algorithm. The default is Marquardt.
<code>m = integer</code>	Maximum number of iterations.
<code>c = number</code>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.
<code>showopts / -showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
<code>s</code>	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, pl=5)
```

tests whether the series x comes from a χ^2 distribution with 5 degrees of freedom. The output is stored as a table object TAB1.

Cross-references

See [“Empirical Distribution Tests” on page 321](#) of the *User’s Guide I* for a description of the goodness-of-fit tests.

See also [qqplot \(p. 636\)](#).

fill	Series Procs
------	------------------------------

Fill a series object with specified values.

Syntax

```
series_name.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 provide 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, unless the “l” option is specified. If, however, you list more values than the object can hold, EViews will not modify any observations and will return an error message.

Options

l	Loop repeatedly over the list of values as many times as it takes to fill the series.
o = [date, integer]	Set starting date or observation from which to start filling the series. Default is the beginning of the workfile range.
s	Fill the series only for the current workfile sample. The “s” option overrides the “o” option.
s = sample_name	Fill the series only for the specified subsample. The “s” option overrides the “o” option.

Examples

To generate a series D70 that takes the value 1, 2, and 3 for all observations from 1970:1:

```
series d70=0
d70.fill(o=1970:1,1) 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:

```
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:

```
series d34
d34.fill(l) 0, 0, 1, 1
```

Note that this series could more easily be generated using @seas or the special workfile functions (see [“Basic Date Functions” on page 728](#)).

freq	Series Views
------	------------------------------

Compute frequency tables.

The `freq` command performs a one-way frequency tabulation. The options allow you to control binning (grouping) of observations.

Syntax

```
series_name.freq(options)
```

Options

dropna (default) / keepna	[Drop/Keep] NA as a category.
v = integer (default = 100)	Make bins if the number of distinct values or categories exceeds the specified number.
nov	Do not make bins on the basis of number of distinct values; ignored if you set “v = integer.”
a = number (default = 2)	Make bins if average count per distinct value is less than the specified number.

noa	Do not make bins on the basis of average count; ignored if you set “a = <i>number</i> .”
b = <i>integer</i> (default = 5)	Maximum number of categories to bin into.
n, obs, count (default)	Display frequency counts.
nocount	Do not display frequency counts.
total (default) / nototal	[Display / Do not display] totals.
pct (default) / nopct	[Display / Do not display] percent frequencies.
cum (default) / nocum	(Display/Do not) display cumulative frequency counts/percentages.
p	Print the 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.

Cross-references

See [“One-Way Tabulation” on page 323](#) of the *User’s Guide I* for a discussion of frequency tables.

frml	Series Declaration
------	------------------------------------

Declare a series object with a formula for auto-updating, or specify a formula for an existing series.

Syntax

```
frml series_name = series_expression  
frml series_name = @clear
```

Follow the `frml` keyword with a name for the series, and an assignment statement. The special keyword “@CLEAR” is used to return the auto-updating series to an ordinary numeric series.

Examples

To define an auto-updating numeric series, you must use the `frml` keyword prior to entering an assignment statement. The following example creates a series named `LOW` that uses a formula to compute its values.:

```
frml low = inc<=5000 or edu<13
```

The auto-updating series takes the value 1 if either `INC` is less than or equal to 5000 or `EDU` is less than 13, and 0 otherwise, and will be re-evaluated whenever `INC` or `EDU` change.

You may apply a `frml` to an existing series. The commands:

```
series z = 3
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 = 3
z = (x+y)/2
```

while similar, produce quite different results, since the absence of the `frml` keyword in the second example means that `EViews` will generate fixed values in the series instead of defining a formula to compute the series values. In this latter case, the values in the series `Z` are fixed, and 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 obtains 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, you should use the special expression “@CLEAR” in your `frml` assignment. The command:

```
frml z = @clear
```

sets the series to numeric value format, freezing the contents of the series at the current values.

Cross-references

See [“Auto-Updating Series” on page 145](#) of the *User’s Guide I*.

See also [Link::link](#) (p. 225).

genr	Series Declaration
------	------------------------------------

Generate series.

Syntax

```
genr ser_name = expression
```

Examples

```
genr y = 3 + x
```

generates a numeric series that takes the values from the series X and adds 3.

Cross-references

See [Series::series](#) (p. 375) for a discussion of the expressions allowed in `genr`.

hist	Series Views
------	------------------------------

Histogram and descriptive statistics of a series.

The `hist` command computes descriptive statistics and displays a histogram for the series.

Syntax

```
series_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 306](#) of the *User’s Guide I* for a discussion of the descriptive statistics reported in the histogram view.

hpf	Series Procs
-----	------------------------------

Smooth a series using the Hodrick-Prescott filter.

Syntax

```
series_name.hpf(options) filtered_name
```

Smoothing Options

The degree of smoothing may be specified as an option. You may specify the smoothing as a value, or using a power rule:

lambda = <i>arg</i>	Set smoothing parameter value to <i>arg</i> ; a larger number results in greater smoothing.
power = <i>arg</i> (default = 2)	Set smoothing parameter value using the frequency power rule of Ravn and Uhlig (2002) (the number of periods per year divided by 4, raised to the power <i>arg</i> , and multiplied by 1600). Hodrick and Prescott recommend the value 2; Ravn and Uhlig recommend the value 4.

If no smoothing option is specified, EViews will use the power rule with a value of 2.

Other Options

p	Print the graph of the smoothed series and the original series.
---	---

Examples

```
gdp.hpf(lambda=1000) gdp_hp
```

smooths the GDP series with a smoothing parameter “1000” and saves the smoothed series as GDP_HP.

Cross-references

See “[Hodrick-Prescott Filter](#)” on page 360 of the *User’s Guide I* for details.

kdensity	Series Views
----------	------------------------------

Kernel density plots.

The `kdensity` command is no longer supported. See [distplot](#) (p. 615).

label	Series Views Series Procs
-------	---

Display or change the label view of a series object, including the last modified date and display name (if any).

As a procedure, `label` changes the fields in the series label.

Syntax

```
series_name.label
series_name.label(options) [text]
```

Options

The first version of the command displays the label view of the series. The second version may be used to modify the label. 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 <i>text</i> .
s	Sets the source field to <i>text</i> .
u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of SER1 with “Data from CPS 1988 March File”:

```
ser1.label(r)
ser1.label(r) Data from CPS 1988 March File
```

To append additional remarks to SER1, and then to print the label view:

```
ser1.label(r) Log of hourly wage
ser1.label(p)
```

To clear and then set the units field, use:

```
ser1.label(u) Millions of bushels
```

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels.

See also [Series::displayname](#) (p. 364).

map	Series Procs
-----	------------------------------

Assign or remove value map setting.

Syntax

```
series_name.map [valmap_name]
```

If the optional valmap name is provided, the procedure will assign the specified value map to the series. If no name is provided, EViews will remove an existing valmap assignment.

Examples

```
series1.map mymap
```

assigns the valmap object MYMAP to SERIES1.

```
series1.map
```

removes an existing valmap assignment from SERIES1.

Cross-references

See [“Value Maps” on page 159](#) of the *User’s Guide I* for a discussion of valmap objects in EViews.

resample	Series Procs
----------	------------------------------

Resample from observations in a series.

Syntax

```
series_name.resample(options) [output_spec]
```

You should follow the `resample` keyword and options with a list of names or a 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

<code>outsmpl = smpl_spec</code>	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.
<code>permute</code>	Draw from rows without replacement. Default is to draw with replacement.
<code>weight = series_name</code>	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.

<code>block = integer</code>	Block length for each draw. Must be a positive integer. The default block length is 1.
<code>withna (default)</code>	[Draw / Do not draw] from all rows in the current sample, including those with NAs.
<code>dropna</code>	Do not draw from rows that contain missing values in the current workfile sample.
<code>fixna</code>	Excludes NAs from draws but copies rows containing missing values to the output series.

- 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 rows containing missing values and the output sample is adjusted so as 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

```
ser1.resample
```

creates a new series SER1_B by drawing with replacement from the rows of SER1 in the current workfile sample. If SER1_B already exists in the workfile, it will be overwritten if it is a series objects, otherwise EViews will error. Note that only values of SER_B (in this case the current workfile sample) will be overwritten.

```
ser1.resample(weight=wt,suffix=_2)
```

will append “_2” to the SER1 for the name of the new series, SER_2. 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 [“Resample” on page 337](#) of the *User’s Guide I* for a discussion of the resampling procedure. For additional discussion of wildcards, see [Appendix C. “Wildcards,” on page 775](#) of the *User’s Guide I*.

See also [@resample \(p. 860\)](#) and [@permute \(p. 859\)](#) for sampling from matrices.

seas	Series Procs
------	------------------------------

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 [Series::x11 \(p. 400\)](#) and [Series::x12 \(p. 402\)](#).

Syntax

```
series_name.seas(options) name_adjust [name_fac]
```

Options

m	Multiplicative (ratio to moving average) method.
a	Additive (difference from moving average) method.

Examples

```
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 339](#) of the *User’s Guide I* for a discussion of seasonal adjustment methods.

See also [seasplot \(p. 651\)](#), [Series::x11 \(p. 400\)](#) and [Series::x12 \(p. 402\)](#).

series	Series Declaration
--------	------------------------------------

Declare a series object.

The `series` command creates and optionally initializes a series, or modifies an existing series.

Syntax

```
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 121](#) of the *User’s Guide I*.

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  
low = inc<=5000 or edu<13
```

This example solves for the implicit relation and generates a series named Z which is the double log of Y so that $Z = \log(\log(Y))$.

```
series exp(exp(z)) = y
```

The command:

```
series z = (x+y)/2
```

creates a series named Z which is the average of series X and Y.

```
series cwage = wage*(hrs>5)
```

generates a series named CWAGE which is equal to WAGE if HRS exceeds 5, and zero otherwise.

```
series 10^z = y
```

generates a series named Z which is the base 10 log of Y.

The commands:

```
series y_t = y  
smpl if y<0  
y_t = na  
smpl @all
```

generate a series named Y_T which replaces negative values of Y with NAs.

```
series z = @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.

```
series z = (.5*x(6)+@movsum(x(5),11)+.5*x(-6))/12
```

generates a series named Z which is the *centered* moving average of the series X over twelve periods.

```
genr y = 2+(5-2)*rnd
```

creates a series named Y which is a random draw from a uniform distribution between 2 and 5.

```
series y = 3+@sqr(5)*nrnd
```

generates a series named Y which is a random draw from a normal distribution with mean 3 and variance 5.

Cross-references

There is an extensive set of functions that you may use with series:

- A list of functions is presented in [Appendix A. “Operator and Function Reference,” on page 733](#) of the *User’s Guide I*.

See “[Numeric Expressions](#)” on [page 121](#) of the *User’s Guide I* for a discussion of rules for forming EViews expressions.

setconvert	Series Procs
------------	------------------------------

Set frequency conversion method.

Determines the default frequency conversion method for a series when copied or linked between different frequency workfiles.

You may override this default conversion method by specifying a frequency conversion method as an option in the specific command (using [copy](#) (p. 696), [fetch](#) (p. 717), or [Link::linkto](#) (p. 226)).

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

```
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:

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:

High to low conversion methods	<i>High to low conversion methods removing NAs:</i> “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). <i>High to low conversion methods propagating NAs:</i> “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).
--------------------------------	---

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 106](#) of the *User’s Guide I* for a discussion of frequency conversion and the treatment of missing values.

See also [copy \(p. 696\)](#), [fetch \(p. 717\)](#), and [Link::linkto \(p. 226\)](#).

setformat	Series Procs
-----------	------------------------------

Set the display format for cells in a series object spreadsheet view.

Syntax

```
series_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.

For series, `setformat` operates on all of the cells in the series.

To format numeric values, you should use one of the following format specifications:

<i>g[.precision]</i>	significant digits
<i>f[.precision]</i>	fixed decimal places
<i>c[.precision]</i>	fixed characters
<i>e[.precision]</i>	scientific/float
<i>p[.precision]</i>	percentage
<i>r[.precision]</i>	fraction

In order to set a format that groups digits into thousands, place a “t” after a specified format. For example, to obtain a fixed number of decimal places, with commas used to separate thousands, use “ft[.precision]”. To obtain a fixed number of characters with a period used to separate thousands, use “ct[.precision]”.

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 “[Date Formats](#)” on page 707 of the *User’s Guide I*). 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 given by:

WF	(uses current EViews workfile date display format)
YYYY	“2002”
YYYY-Mon	“2002-Jan”
YYYYMon	“2002 Jan”
YYYY[M]MM	“2002[M]01”
YYYY:MM	“2002:01”

YYYY[Q]Q	“2002[Q]1”
YYYY:Q	“2002:Q
YYYY[S]S	“2002[S]1” (semi-annual)
YYYY:S	“2002:1”
YYYY-MM-DD	“2002-01-07”
YYYY Mon dd	“2002 Jan 7”
YYYY Month dd	“2002 January 7”
YYYY-MM-DD HH:MI	“2002-01-07 22:40”
YYYY-MM-DD HH:MI:SS	“2002-01-07 22:40:30”
YYYY-MM-DD HH:MI:SS.SSS	“2002-01-07 22:40:30.125”
Mon-YYYY	“Jan-2002”
Mon dd YYYY	“Jan 7 2002”
Mon dd, YYYY	“Jan 7, 2002”
Month dd YYYY	“January 7 2002”
Month dd, YYYY	“January 7, 2002”
MM/DD/YYYY	“01/07/2002”
mm/DD/YYYY	“1/07/2002”
mm/DD/YYYY HH:MI	“1/07/2002 22:40”
mm/DD/YYYY HH:MI:SS	“1/07/2002 22:40:30”
mm/DD/YYYY HH:MI:SS.SSS	“1/07/2002 22:40:30.125”
mm/dd/YYYY	“1/7/2002”
mm/dd/YYYY HH:MI	“1/7/2002 22:40”
mm/dd/YYYY HH:MI:SS	“1/7/2002 22:40:30”
mm/dd/YYYY HH:MI:SS.SSS	“1/7/2002 22:40:30.125”
dd/MM/YYYY	“7/01/2002”
dd/mm/YYYY	“7/1/2002”
DD/MM/YYYY	“07/01/2002”
dd Mon YYYY	“7 Jan 2002”
dd Mon, YYYY	“7 Jan, 2002”
dd Month YYYY	“7 January 2002”
dd Month, YYYY	“7 January, 2002”
dd/MM/YYYY HH:MI	“7/01/2002 22:40”
dd/MM/YYYY HH:MI:SS	“7/01/2002 22:40:30”
dd/MM/YYYY HH:MI:SS.SSS	“7/01/2002 22:40:30.125”
dd/mm/YYYY hh:MI	“7/1/2002 22:40”

dd/mm/YYYY hh:MI:SS	“7/1/2002 22:40:30”
dd/mm/YYYY hh:MI:SS.SSS	“7/1/2002 22:40:30.125”
hm:MI am	“10:40 pm”
hm:MI:SS am	“10:40:30 pm”
hm:MI:SS.SSS am	“10:40:30.125 pm”
HH:MI	“22:40”
HH:MI:SS	“22:40:30”
HH:MI:SS.SSS	“22:40:30.125”
hh:MI	“22:40”
hh:MI:SS	“22:40:30”
hh:MI:SS.SSS	“22:40:30.125”

Note that the “hh” formats display 24-hour time without leading zeros. In our examples above, there is no difference between the “HH” and “hh” formats for 10 p.m.

Also note that all of the “YYYY” formats above may be displayed using two-digit year “YY” format.

Examples

To set the format for all cells in the series to fixed 5-digit precision, simply provide the format specification:

```
ser1.setformat f.5
```

Other format specifications include:

```
ser1.setformat f(.7)
```

```
ser1.setformat e.5
```

You may use any of the date formats given above:

```
ser1.setformat YYYYMon
```

```
ser1.setformat "YYYY-MM-DD HH:MI:SS.SSS"
```

to set the series display characteristics.

Cross-references

See [Series::setwidth \(p. 383\)](#), [Series::setindent \(p. 382\)](#) and [Series::setjust \(p. 382\)](#) for details on setting spreadsheet widths, indentation and justification.

setindent	Series Procs
-----------	------------------------------

Set the display indentation for cells in a series object spreadsheet view.

Syntax

`series_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 spreadsheet (the EViews spreadsheet default font at the time the spreadsheet 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 spreadsheet view is created.

For series, `setindent` operates on all of the cells in the series.

Examples

To set the indentation for a series object:

```
ser1.setindent 2
```

Cross-references

See [Series::setWidth \(p. 383\)](#) and [Series::setjust \(p. 382\)](#) for details on setting spreadsheet widths and justification.

setjust	Series Procs
---------	------------------------------

Set the display justification for cells in a series spreadsheet view.

Syntax

`series_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.

For series, `setjust` operates on all of the cells in the series.

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 taken from the Global Defaults for spreadsheet views.

Examples

```
ser1.setjust middle
```

sets the vertical justification to the middle.

```
ser1.setjust top left
```

sets the vertical justification to top and the horizontal justification to left.

Cross-references

See [Series::setwidth \(p. 383\)](#) and [Series::setindent \(p. 382\)](#) for details on setting spreadsheet widths and indentation.

setwidth	Series Procs
----------	------------------------------

Set the column width for a series spreadsheet.

Syntax

```
series_name.setwidth width_arg
```

where *width_arg* specifies the width unit value. The width unit is computed from representative characters in the default font for the current spreadsheet (the EViews spreadsheet default font at the time the spreadsheet 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

```
ser1.setwidth 12
```

sets the width of series SER1 to 12 width units.

Cross-references

See [Series::setindent \(p. 382\)](#) and [Series::setjust \(p. 382\)](#) for details on setting spreadsheet indentation and justification.

sheet	Series Views
-------	------------------------------

Spreadsheet view of a series object.

Syntax

```
series_name.sheet(options)
```

Options

p	Print the spreadsheet view.
---	-----------------------------

Examples

```
ser1.sheet(p)
```

displays and prints the spreadsheet view of series SER1.

Cross-references

See [Chapter 5. “Basic Data Handling,” on page 77](#) of the *User’s Guide I* for a discussion of the spreadsheet view of series and groups.

smooth	Series Procs
--------	------------------------------

Exponential smoothing.

Forecasts a series using one of a number of exponential smoothing techniques. By default, `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.

`smooth` automatically calculates in-sample forecast errors and puts them into the series `RESID`.

Syntax

```
series_name.smooth(method) smooth_name [freq]
```

You should follow the `smooth` keyword with a name for the smoothed series. You must also specify the smoothing method in parentheses. 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

s[,x]	Single exponential smoothing for series with no trend. You may optionally specify a number <i>x</i> between zero and one for the mean parameter.
d[,x]	Double exponential smoothing for series with a trend. You may optionally specify a number <i>x</i> between zero and one for the mean parameter.

<code>n[,x,y]</code>	Holt-Winters without seasonal component. You may optionally specify numbers x and y between zero and one for the mean and trend parameters, respectively.
<code>a[,x,y,z]</code>	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.
<code>m[,x,y,z]</code>	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.

Other Options:

<code>p</code>	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

```
sales.smooth(s) 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.

```
tb3.smooth(n,e,.3) tb3_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.

```
smpl @first @last-10
order.smooth(m,.1,.1,.1) order_hw
smpl @all
graph gra1.line order order_hw
show 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

See [“Exponential Smoothing” on page 354](#) of the *User’s Guide I* for a discussion of exponential smoothing methods.

sort	Series Procs
------	------------------------------

Change display order for series spreadsheet.

The `sort` command changes the sort order settings for spreadsheet display of the series.

Syntax

`series_name.sort([opt])`

By default, EViews will sort by the value of the series, in ascending order. For purposes of sorting, NAs are considered to be smaller than any other value.

You may modify the default sort order by providing a sort option. If you provide the integer “0”, or the keyword “obs”, EViews will sort using the original workfile observation order. To sort in descending order, simply include the minus sign (“-”).

Examples

`ser1.sort`

change the display order for the series SER1 so that spreadsheet rows are ordered from low to high values of the series.

`ser1.sort(-)`

sorts in descending order.

`ser1.sort(obs)`

returns the display order for group SER1 to the original (by observation).

Cross-references

See [“Spreadsheet” on page 368](#) of the *User’s Guide II* for additional discussion.

statby	Series Views
--------	------------------------------

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

`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

<code>sum</code>	Display sums.
<code>med</code>	Display medians.
<code>max</code>	Display maxima.
<code>min</code>	Display minima.
<code>quant = arg</code> (<i>default</i> = .5)	Display quantile with value given by the argument.
<code>q = arg</code> (<i>default</i> = “r”)	Compute quantiles using the specified definition: “b” (Blom), “r” (Rankit-Cleveland), “o” (Ordinary), “t” (Tukey), “v” (van der Waerden), “g” (Gumbel).
<code>skew</code>	Display skewness.
<code>kurt</code>	Display kurtosis.
<code>na</code>	Display counts of NAs.
<code>nomean</code>	Do not display means.
<code>nostd</code>	Do not display standard deviations.
<code>nocount</code>	Do not display counts.

Options to control layout

<code>l</code>	Display in list mode (for more than one classifier).
<code>nor</code>	Do not display row margin statistics.
<code>noc</code>	Do not display column margin statistics.

nom	Do not display table margin statistics (unconditional tables); for more than two classifier series.
nos	Do not display sub-margin totals in list mode; only used with “l” option and more than two classifier series.
sp	Display sparse labels; only with list mode option, “l”.

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

```
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 749](#) for functions to compute by-group statistics. See also [“Stats by Classification” on page 308](#) and [“Descriptive Statistics” on page 379](#) of the *User’s Guide I*.

See also [Series::hist \(p. 370\)](#), [boxplot \(p. 613\)](#) and [Link::linkto \(p. 226\)](#).

stats	Series Views
--------------	------------------------------

Descriptive statistics.

Computes and displays a table of means, medians, maximum and minimum values, standard deviations, and other descriptive statistics of a series.

Syntax

```
series_name.stats(options)
```

Options

p	Print the stats table.
---	------------------------

Examples

```
wage.stats
```

displays the descriptive statistics view of the series WAGE.

Cross-references

See “[Descriptive Statistics & Tests](#)” on page 306 of the *User’s Guide I* for a discussion of the descriptive statistics views of series.

See also [boxplot](#) (p. 613) and [Series::hist](#) (p. 370).

stom	Series Procs
-------------	------------------------------

Convert a series to a vector.

Fills a vector with the data from a series.

Syntax

```
stom(series_name, vector[, sample])
```

Include the series name in parentheses followed by a comma and the vector name. By default, the series values in the current workfile sample are used to fill the vector; 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 (for alternative behavior, see [Series::stomna](#) (p. 390)).
- If the vector already exists in the workfile, EViews automatically resizes the vector to fit the series.

Examples

```
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.

```
sample s1 1951 1990
stom(ser1,v,s1)
```

converts a series SER1 to a vector named V using sample S1. The vector V will have 1 column and 40 rows (provided there are no NAs).

Cross-references

See [Chapter 18. “Matrix Language,” on page 627](#) of the *User’s Guide I* for further discussion and examples of matrices.

See also [Series::stomna \(p. 390\)](#) and [mtos \(p. 857\)](#).

stomna	Series Procs
--------	------------------------------

Convert a series to a vector without dropping NAs.

Fills a vector with the data from a series without dropping observations with missing values.

Works in identical fashion to [Series::stom \(p. 389\)](#), but does not drop observations containing NAs.

Syntax

```
stomna(series, vector[, sample])
```

Include the series name in parentheses, followed by a comma and the vector name. By default, the series values in the current workfile sample are used to fill the vector. 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.

```
sample s1 1951 1990
stomna(ser1,v,s1)
```

converts a series SER1 to a vector named V using sample S1. The vector V will always have 1 column and 40 rows.

Cross-references

See [Chapter 18. “Matrix Language,” on page 627](#) of the *User’s Guide I* for further discussion and examples of matrices.

See also [Series::stom \(p. 389\)](#) and [mtos \(p. 857\)](#).

testby	Series Views
---------------	------------------------------

Test equality of the mean, median, or variance of a series across categories classified by a list of series or a group.

Syntax

```
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

<code>mean (default)</code>	Test equality of mean.
<code>med</code>	Test equality of median.
<code>var</code>	Test equality of variance.
<code>dropna (default), keepna</code>	[Drop /Keep] NAs as a classification category.
<code>v = integer (default = 100)</code>	Bin categories if classification series take more than the specified number of distinct values.
<code>nov</code>	Do not bin based on the number of values of the classification series.
<code>a = number (default = 2)</code>	Bin categories if average cell count is less than the specified number.
<code>noa</code>	Do not bin on the basis of average cell count.
<code>b = integer (default = 5)</code>	Set maximum number of binned categories.
<code>p</code>	Print the test results.

Examples

```
wage.testby(med) race
```

Tests equality of medians of WAGE across groups classified by RACE.

Cross-references

See “[Equality Tests by Classification](#)” on page 314 of the *User’s Guide I* for a discussion of equality tests.

See also [Group::testbtw](#) (p. 218), [Series::teststat](#) (p. 392).

teststat	Series Views
----------	------------------------------

Test simple hypotheses of whether the mean, median, or variance of a series is equal to a specified value.

Syntax

`series_name.teststat(options)`

Specify the type of test and the value under the null hypothesis as an option.

Options

<code>mean = number</code>	Test the null hypothesis that the mean equals the specified number.
<code>med = number</code>	Test the null hypothesis that the median equals the specified number.
<code>var = number</code>	Test the null hypothesis that the variance equals the specified number. The number must be positive.
<code>std = number</code>	Test equality of mean conditional on the specified standard deviation. The standard deviation must be positive.
<code>p</code>	Print the test results.

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 “[Descriptive Statistics & Tests](#)” on page 306 of the *User’s Guide I* for a discussion of simple hypothesis tests.

See also [Group::testbtw](#) (p. 218), [Series::testby](#) (p. 391).

tramoseats	Series Procs
------------	------------------------------

Run the external seasonal adjustment program Tramo/Seats using the data in the series.

tramoseats 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:

$$n = \begin{cases} 36 & \text{for monthly data} \\ \max\{12, 4s\} & \text{for other seasonal data} \end{cases} \tag{1.2}$$

Syntax

```
series_name.tramoseats(options) [base_name]
```

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 (default = "ts")	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).
save = arg	<p>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).</p> <p>To save more than one series, separate the list of key words with a space. <i>Do not use commas</i> within the list of save series.</p> <p>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".</p>
opt = arg	A space delimited list of input namelist. <i>Do not use commas within the list.</i> The syntax for the input namelist is explained in the PDF documentation file. See also "Notes" on page 394 below.

<code>reg = arg</code>	A space delimited list for one line of reg namelist. <i>Do not use commas within the list.</i> 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” on page 394 below.
<code>regname = arg</code>	Name of a series or group in the current workfile that contains the exogenous regressors specified in the previous “reg =” option. See “Notes” on page 394 below.
<code>p</code>	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 to 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 352](#) of the *User’s Guide I*.

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: “itrاد =”.
 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) 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 containing the lines:

```
$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
```

Additional examples replicating many of the example files provided by Tramo/Seats can be found in the “./Example Files” directory. You will also find files that compare seasonal adjustments from Census X12 and Tramo/Seats.

Cross-references

See [“Tramo/Seats” on page 349](#) of the *User’s Guide I* for discussion. See also the Tramo/Seats documentation that accompanied your EViews distribution.

See [Series::seas \(p. 375\)](#) and [Series::x12 \(p. 402\)](#).

uroot	Series Views
-------	------------------------------

Carries out unit root tests on a series or panel structured series.

The ordinary, single series 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).

If used on a series in a panel structured workfile, the procedure will perform panel unit root testing. 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.

Note that simulation evidence suggests that in various settings (for example, small T), Hadri's panel unit root test experiences significant size distortion in the presence of autocorrelation when there is no unit root. In particular, the Hadri test appears to over-reject the null of stationarity, and may yield results that directly contradict those obtained using alternative test statistics (see Hlouskova and Wagner (2006) for discussion and details).

Syntax

`series_name.uroot(options)`

Options

Basic Specification Options

You should specify the exogenous variables and order of dependent variable differencing in the test equation using the following options:

<code>const (default)</code>	Include a constant in the test equation.
<code>trend</code>	Include a constant and a linear time trend in the test equation.
<code>none</code>	Do not include a constant or time trend (only available for the ADF and PP tests).
<code>dif = integer (default = 0)</code>	Order of differencing of the series prior to running the test. Valid values are {0, 1, 2}.

For backward compatibility, the shortened forms of these options, “c”, “t”, and “n”, are presently supported. For future compatibility we recommend that you use the longer forms.

For ordinary (non-panel) unit root tests, you should specify the test type using one of the following keywords:

adf (<i>default</i>)	Augmented Dickey-Fuller.
dfgls	GLS detrended Dickey-Fuller (Elliot, Rothenberg, and Stock).
pp	Phillips-Perron.
kpss	Kwiatkowski, Phillips, Schmidt, and Shin.
ers	Elliot, Rothenberg, and Stock (Point Optimal).
np	Ng and Perron.

For panel testing, you may use one of the following keywords to specify the test:

sum (<i>default</i>)	Summary of all of the panel unit root tests.
llc	Levin, Lin, and Chu.
breit	Breitung.
ips	Im, Pesaran, and Shin.
adf	Fisher - ADF.
pp	Fisher - PP.
hadri	Hadri.

Options for ordinary (non-panel) unit root tests

In addition, the following panel specific options are available:

<code>hac = arg</code>	<p>Method of estimating the frequency zero spectrum: “bt” (Bartlett kernel), “pr” (Parzen kernel), “qs” (Quadratic Spectral kernel), “ar” (AR spectral), “ardt” (AR spectral - OLS detrended data), “argls” (AR spectral - GLS detrended data).</p> <p>Applicable to PP, KPSS, ERS, and NP tests. <i>The default settings are test specific</i> (“bt” for PP and KPSS, “ar” for ERS, “argls” for NP).</p>
<code>band = arg,</code> <code>b = arg</code> <i>(default = “nw”)</i>	<p>Method of selecting the bandwidth: “nw” (Newey-West automatic variable bandwidth selection), “a” (Andrews automatic selection), “number” (user specified bandwidth).</p> <p>Applicable to PP, KPSS, ERS, and NP tests when using kernel sums-of-covariances estimators (where “hac = ” is one of {bt, pz, qs}).</p>
<code>lag = arg</code> <i>(default = “a”)</i>	<p>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).</p> <p>Applicable to ADF and DFGLS tests, and for the other tests when using AR spectral density estimators (where “hac = ” is one of {ar, ardt, argls}).</p>
<code>info = arg</code> <i>(default = “sic”)</i>	<p>Information criterion to use when computing automatic lag length selection: “aic” (Akaike), “sic” (Schwarz), “hq” (Hannan-Quinn), “msaic” (Modified Akaike), “msic” (Modified Schwarz), “mhqc” (Modified Hannan-Quinn).</p> <p>Applicable to ADF and DFGLS tests, and for other tests when using AR spectral density estimators (where “hac = ” is one of {ar, ardt, argls}).</p>
<code>maxlag = integer</code>	<p>Maximum lag length to consider when performing automatic lag length selection:</p> $\text{default} = \text{int}((12 T / 100)^{0.25})$ <p>Applicable to ADF and DFGLS tests, and for other tests when using AR spectral density estimators (where “hac = ” is one of {ar, ardt, argls}).</p>

Options for panel unit root tests

The following panel specific options are available:

balance	Use balanced (across cross-sections or series) data when performing test.
hac = <i>arg</i> (<i>default</i> = “bt”)	Method of estimating the frequency zero spectrum: “bt” (Bartlett kernel), “pr” (Parzen kernel), “qs” (Quadratic Spectral kernel). Applicable to “Summary”, LLC, Fisher-PP, and Hadri tests.
band = <i>arg</i> , b = <i>arg</i> (<i>default</i> = “nw”)	Method of selecting the bandwidth: “nw” (Newey-West automatic variable bandwidth selection), “a” (Andrews automatic selection), <i>number</i> (user-specified common bandwidth), <i>vector_name</i> (user-specified individual bandwidth). Applicable to “Summary”, LLC, Fisher-PP, and Hadri tests.
lag = <i>arg</i>	Method of selecting lag length (number of first difference terms) to be included in the regression: “a” (automatic information criterion based selection), <i>integer</i> (user-specified common lag length), <i>vector_name</i> (user-specific individual lag length). If the “balance” option is used, $default = \begin{cases} 1 & \text{if } (T_{\min} \leq 60) \\ 2 & \text{if } (60 < T_{\min} \leq 100) \\ 4 & \text{if } (T_{\min} > 100) \end{cases}$ where T_{\min} is the length of the shortest cross-section or series, otherwise <i>default</i> = “a”. Applicable to “Summary”, LLC, Breitung, IPS, and Fisher-ADF tests.
info = <i>arg</i> (<i>default</i> = “sic”)	Information criterion to use when computing automatic lag length selection: “aic” (Akaike), “sic” (Schwarz), “hqc” (Hannan-Quinn). Applicable to “Summary”, LLC, Breitung, IPS, and Fisher-ADF tests.
maxlag = <i>arg</i>	Maximum lag length to consider when performing automatic lag length selection, where <i>arg</i> is an <i>integer</i> (common maximum lag length) or a <i>vector_name</i> (individual maximum lag length) $default = \text{int}(\min_i(12, T_i/3) \cdot (T_i/100)^{0.25})$ 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(dfqls,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.

```
gdp.root(llc,hac=pr, info=aic)
```

runs the LLC panel unit root test on series GDP. The frequency zero spectrum is estimated using the Parzen Kernel with lag length automatically selected using the AIC.

Cross-references

See [“Unit Root Tests” on page 88](#) of the *User’s Guide II* for discussion of standard unit root tests performed on a single series, and [“Panel Unit Root Tests” on page 100](#) of the *User’s Guide II* for discussion of unit roots tests performed on panel structured workfiles, groups of series, or pooled data.

x11	Series Procs
-----	------------------------------

Seasonally adjust series using the Census X11.2 method.

Syntax

```
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

m	Multiplicative seasonals.
a	Additive seasonals.
s	Use sliding spans.
h	Adjustment for all holidays (only for monthly data specified with the m option).
i	Adjustment for holidays if significant (only for monthly data specified with the “m” option).
t	Adjustment for all trading days (only for monthly data).
q	Adjustment for trading days if significant (only for monthly data).
p	Print the X11 results.

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

See [“Census X11 \(Historical\)” on page 348](#) of the *User’s Guide I* for a discussion of Census X11 seasonal adjustment method.

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. Additional documentation for these programs can also be found in your EViews directory in the text files X11DOC1.TXT through X11DOC3.TXT.

See also [Series::seas \(p. 375\)](#), [seasplot \(p. 651\)](#), [Series::tramoseats \(p. 393\)](#), and [Series::x12 \(p. 402\)](#).

x12	Series Procs
-----	------------------------------

Seasonally adjust series using the Census X12 method.

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

`series_name.x12(options) base_name`

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 = <i>arg</i> (default = “m”)	Seasonal adjustment method: “m” (multiplicative adjustment; <i>Series must take only non-negative values</i>), “a” (additive adjustment), “p” (pseudo-additive adjustment), “l” (log-additive seasonal adjustment; <i>Series must take only positive values</i>).
filter = <i>arg</i> (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; <i>Series must have at least 20 years of data</i>).
save = “ <i>arg</i> ”	<p>Optionally saved series keyword enclosed in quotes. List the extension (given in Table 6-8, p.71 of the <i>X12-ARIMA Reference Manual</i>) for the series you want to save. The created series will use names of the form <i>basename</i>, 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”).</p> <p>All other options are named using the option symbol. For example “save = “d16”” will store a series named <i>basename_D16</i>.</p> <p>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.</p>
tf = <i>arg</i>	Transformation for regARIMA: “logit” (Logit transformation), “auto” (automatically choose between no transformation and log transformation), <i>number</i> (Box-Cox power transformation using specified parameter; use “tf = 0” for log transformation).
s span	Sliding spans stability analysis. <i>Cannot be used along with the “h” option.</i>
history	Historical record of seasonal adjustment revisions. <i>Cannot be used along with the “sspan” option.</i>
check	Check residuals of regARIMA.
outlier	Outlier analysis of regARIMA.

x11reg = <i>arg</i>	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 <i>X12-ARIMA Reference Manual</i> . To specify more than one regressor, separate by a space within the double quotes.
reg = <i>arg_list</i>	Regressors for the regARIMA model. Regressors must be chosen from the predefined list in Table 6-17, pp. 100-101 of the <i>X12-ARIMA Reference Manual</i> . To specify more than one regressor, separate by a space within the double quotes.
arima = <i>arg</i>	ARIMA spec of the regARIMA model. Must follow the X12 ARIMA specification syntax. <i>Cannot be used together with the “amdl = ” option.</i>
amdl = f	Automatically choose the ARIMA spec. Use forecasts from the chosen model in seasonal adjustment. <i>Cannot be used together with the “arima = ” option and must be used together with the “mfile = ” option.</i>
amdl = b	Automatically choose the ARIMA spec. Use forecasts and backcasts from the chosen model in seasonal adjustment. <i>Cannot be used together with the “arima = ” option and must be used together with the “mfile = ” option.</i>
best	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). <i>Must be used together with the “amdl = ” option.</i>
modelsmpl = <i>arg</i>	Sets the subsample for fitting the ARIMA model. Either specify a sample object name or a sample range. <i>The model sample must be a subsample of the current work-file sample and should not contain any breaks.</i>
mfile = <i>arg</i>	Specifies the file name (include the extension, if any) that contains a list of ARIMA specifications to choose from. <i>Must be used together with the “amdl = ” option.</i> The default is the X12A.MDL file provided by the Census.
outsmpl	Use out-of-sample forecasts for automatic model selection. Default is in-sample forecasts. <i>Must be used together with the “amdl = ” option.</i>
plotspectra	Save graph of spectra for differenced seasonally adjusted series and outlier modified irregular series. The saved graph will be named GR_ <i>seriesname</i> _SP.
p	Print X12 procedure results.

Other Options

hma = <i>integer</i>	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 <i>odd integer between 1 and 101</i> .
sigl = <i>arg</i>	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 = <i>arg</i>	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). <i>Cannot be used together with the “easter[w]” regressor in the “reg = ” or “x11reg = ” options.</i>
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 = <i>integer</i>	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.
fback = <i>integer</i>	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.
aicx11	Test (based on AIC) whether to retain the regressors specified in “x11reg = ”. <i>Must be used together with the “x11reg = ” option.</i>
aicreg	Test (based on AIC) whether to retain the regressors specified in “reg = ”. <i>Must be used together with the “reg = ” option.</i>
sfile = <i>arg</i>	Path/name (including extension, if any) of user provided specification file. The file must follow a specific format; see the discussion below.

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 the ability to pass your own X12 specification file from EViews. The advantage of using this method (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*, with the following exceptions:

- the specification file should have neither a series spec nor a composite spec.
- the `x11` spec must include a save option for D11 (the final seasonally adjusted series) in addition to any other extensions you want to store. EViews will always look for D11, and will error if it is not found.
- to read back data for a “save” option other than D11, you must include the “save =” option in the `x12` proc. For example, to obtain 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.

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:

```
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.

```
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.

```
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.

```
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 instead specify this option 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.

```
freeze(x12out) sales.x12(tf=auto, amdl=f, mfile=
    "c:\evIEWS\mymdl.txt")
```

stores the output from X12 in the text object 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.

```
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 again be ignored (you should instead 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 340](#) of the *User’s Guide I* for a discussion of the Census X12 program. The documentation for X12, *X12-ARIMA Reference Manual*, may be found in the DOCS subdirectory of your EViews directory, in the PDF files FINALPT1.PDF and FINALPT2.PDF.

See also [Series::seas](#) (p. 375) and [Series::x11](#) (p. 400).

References

Ravn, Morten O. and Harald Uhlig (2002). “On Adjusting the Hodrick-Prescott Filter for the Frequency of Observations,” *Review of Economics and Statistics*, 84, 371-375

Spool

Spool object. Container for output objects.

Spool Declaration

spool.....create spool object (p. 421).

To declare a spool object, use the keyword `spool`, followed by the object name:

```
spool myspool
```

In addition, you may create a new spool by redirecting print jobs to the spool

```
output(s) mynewspool
tab1.print
```

Spool Views

displaydisplay the contents of the spool (p. 411).

Spool Procs

appendappend objects to a spool (p. 410).

commentassign a comment to an object in a spool (p. 411).

displaynameassign a display name to an object in a spool (p. 412).

extractextract a copy of an object in a spool (p. 412).

flattenremove tree hierarchy from the spool or specified embedded spool (p. 413).

graphmodeset the display mode for graphs in the spool (p. 413).

horizindentsets the horizontal indentation for the spool (p. 414).

insertinsert objects into a spool (p. 415).

labellabel information for the spool object (p. 416).

leftmarginsets the left margin of the spool or a specified embedded spool (p. 414).

movemove an object in the spool (p. 418).

namerename an object in a spool (p. 419).

optionsset display options for a spool (p. 420).

printprint an object in a spool (p. 420).

removeremove objects from a spool (p. 421).

tablemodeset the display mode for tables and text objects in the spool (p. 422).

topmarginsets the top margin of the spool or a specified embedded spool (p. 422).

vertindentsets the vertical indentation for the spool (p. 423).

- [vertspacing](#)..... sets the amount of vertical spacing between objects in the spool (p. 424).
- [width](#) change or reset the width of an object in the spool (p. 424).

Spool Data Members

- [@count](#) number of base objects in the spool.
- [@totalcount](#) number of objects in a flattened version of the spool.
- [@objname\(i\)](#) name of the *i*-th object in the spool.
- [@objtype\(i\)](#) type of the *i*-th object in the spool (“graph”, “table”, “text”, “spool”).

Spool Examples

```
spool myspool
myspool.append ser1.line
myspool.insert(offset=first) ser2.line
myspool.displayname untitled01 "Unemployment Rate"
myspool.options displaynames
```

Spool Entries

The following section provides an alphabetical listing of the commands associated with the “Spool” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

append	Spool Procs
--------	-----------------------------

Append objects to a spool.

Syntax

```
spool_name.append object_list
```

where *object_list* is a list of one or more objects to be appended to the spool. You may specify a view for each object, otherwise the default view will be used.

Examples

To insert a line graph view of series SER1 and a bar graph view of SER2 as the last objects in SPOOL01:

```
spool01.append ser1.line ser2.bar
```

Cross-references

For additional discussion of spools see “[Spool Objects](#),” beginning on page 554 in the *User’s Guide I*. See also [Spool::insert](#) (p. 415) and [Spool::remove](#) (p. 421).

comment	Spool Procs
---------	-----------------------------

Assign a comment to an object in the spool.

Syntax

```
spool_name.comment object_arg new_comment
```

where *new_comment* specifies the comment for the object specified in *object_arg*, where *object_arg* is the name or position of the object. Surround *object_name* with quotation marks for multiple word comments.

Examples

```
spool01.comment state/tab1 "The state population of Alabama as
found\nfrom http://www.census.gov/popest/states/NST-ann-
est.html."
```

assigns the following comment to object TAB1 embedded in the STATE object:

```
"The state population of Alabama as found
from http://www.census.gov/popest/states/NST-ann-est.html."
```

The “\n” is used to indicate the start of a new line in the comment.

Cross-references

See “[Labeling Objects](#)” on page 72 of the *User’s Guide I* for a discussion of labels and display names.

See also [Spool::label](#) (p. 416).

display	Spool Views
---------	-----------------------------

Display contents of a spool object.

Syntax

```
spool_name.display
```

`display` is the default view for a spool.

Examples

```
spool01.display
```

displays the contents of SPOOL01.

Cross-references

For additional discussion of spools see [“Spool Objects,” beginning on page 554](#) in the *User’s Guide I*.

displayname	Spool Procs
-------------	-----------------------------

Assign a display name to an object in the spool.

Syntax

```
spool_name.displayname object_arg new_name
```

where *new_name* specifies the display name for the object in *object_arg*, where *object_arg* is the name or position of the object. Surround *object_arg* with quotation marks for multiple word display names. Note that the case will be preserved in *new_name*.

Examples

```
spool01.displayname state/tab1 "Unemployment Rate"
```

assigns the “Unemployment Rate” displayname to the object TAB1, which is a child of the STATE spool.

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels and display names. See also [Spool::label \(p. 416\)](#).

extract	Spool Procs
---------	-----------------------------

Extracts a copy of the specified object in a spool.

Syntax

```
spool_name.extract(name) object_name
```

where *object_name* is the object to be extracted from the spool, and *object_name* is the optional name of the new object.

Options

name	Optional name of the new object to be created. An untitled copy will be created if a name is not provided.
------	--

Examples

```
spool01.extract(tab1_copy) tab1
```


creates a copy of table TAB1 and names the copy TAB1_COPY.

Cross-references

For additional discussion of spools see “[Spool Objects,](#)” beginning on page 554 in the *User’s Guide I*. See also [Spool::print](#) (p. 420), [Spool::insert](#) (p. 415) and [Spool::remove](#) (p. 421).

flatten	Spool Procs
---------	-----------------------------

Removes tree hierarchy from the spool or specified embedded spool.

Syntax

```
spool_name.flatten [object_list]
```

where *object_list* is an optional list of one or more embedded spools to be flattened. If an *object_list* is not provided, the entire spool will be flattened.

Examples

```
spool01.flatten
```

flattens the entire spool SPOOL01.

```
spool01.flatten myspool1
```

flattens only the embedded spool MYSPPOOL1.

Cross-references

For additional discussion of spools see “[Spool Objects,](#)” beginning on page 554 in the *User’s Guide I*.

graphmode	Spool Procs
-----------	-----------------------------

Set display mode for graphs in the spool.

Syntax

```
spool_name.graphmode(options) [size_arg]
```

where *size_arg* is an optional size argument (in virtual inches) used for the “fixed” and “variablelimit” modes, and the options are used to specify the mode. If *size_arg* is not provided, the default setting will be used.

Options

```
type = arg (default = "fixed") where arg is "fixed", "variable", or "variablelimit".
```

The “fixed” mode specifies the width of all graph objects in the spool, while “variable” allows graphs to be displayed at their native sizes. The “variablelimit” mode allows graphs to be displayed at native sizes unless their widths exceed a specified limit value.

Examples

```
spool01.graphmode(type=fixed) 5
```

sets all graphs to be displayed at a fixed size of 5 virtual inches, while

```
spool01.graphmode(type=variable)
```

displays graphs at their native sizes.

```
spool01.graphmode(type=variablelimit)
```

allows graphs to be displayed at their native sizes unless they exceed the specified variable limit. Note that native sizes for graphs are a function of the default table font.

Cross-references

For additional discussion of spools see [“Spool Objects,” beginning on page 554](#) in the *User’s Guide I*. See also [Spool::tablemode](#) (p. 422).

horizindent	Spool Procs
-------------	-----------------------------

Change the horizontal indentation size for objects in the spool.

Syntax

```
pool_name.horizindent object_arg size_arg
```

where *object_arg* is the name or the position of a specific object to which you wish to apply indenting, and *size_arg* is an new indentation in virtual inches.

Examples

```
spool01.horizindent 1 0.02
```

```
spool01.horizindent tab1 0.02
```

changes the indentation for both the first object in the spool and for TAB1 to 0.02 virtual inches.

To refer to a child object of a spool, you must specify the object’s path. For instance, given a spool SPOOL01 containing the spool SP1 which in turn contains the graph G2:

```
spool01.horizindent sp1/g2 0.03
```

also changes the horizontal indentation of G2 in the embedded spool SP1 to 0.03 virtual inches, while

```
spool01.horizindent sp1 0.03
```

sets the indentation for the object SP1 to 0.03.

Cross-references

For additional discussion of spools see “[Spool Objects](#),” beginning on page 554 in the *User’s Guide I*. See also [Spool::leftmargin](#) (p. 417), [Spool::topmargin](#) (p. 422), [Spool::vertspacing](#) (p. 424).

insert	Spool Procs
--------	-----------------------------

Insert objects into a spool.

Syntax

```
spool_name.insert(options) object_list
```

where *object_list* is a list of one or more objects to be inserted into the spool at the position specified in the *options*. If you do not specify a view for an object in the list, the default view will be used.

If neither a location nor an offset are specified in the *options*, the object will be inserted at the end of the spool.

Options

<code>loc = <i>arg</i></code>	<i>arg</i> may be an integer position in the spool or the name of an existing object in the spool. The inserted object will be placed before or after <i>arg</i> , as specified by the offset option below. An object name must include its path if it is a child of another spool. For example, use “spool1/gr1” to specify a graph GR1 in spool SPOOL1.
<code>offset = <i>arg</i></code> (<i>before, after, first, last</i>), <code>default = “before”</code>	<i>arg</i> indicates that the object should be inserted relative to the object specified in the loc option above. <i>arg</i> may be “before” or “after”. In addition, if the location specified by the “loc = ” option corresponds to a spool object, <i>arg</i> may be “first” or “last” (default), where the object will be inserted as the first or last object in the spool object specified.

If neither a location nor an offset are specified, the object will be inserted at the end of the spool. If an offset is provided without a location, the object will be inserted relative to the main spool. Providing a location without an offset instructs EViews to insert the object at the location specified, pushing all objects proceeding and including *object_name* down the list of objects.

Examples

To insert a line graph view of the series SER1 as the last object in SPOOL01:

```
spool01.insert ser1.line
```

To insert TAB1 as the first object in SPOOL01:

```
spool01.insert(offset=first) tab1
```

Given a graph GR1,

```
spool01.insert(loc=gr1) tab1 tab2
```

inserts TAB1 in the current location of GR1 and TAB2 immediately following. All objects from GR1 onward are pushed down the list of objects.

Alternately, if SP1 is a spool object,

```
spool01.insert(loc=sp1,offset=last) ser1.line ser1.bar
```

inserts a line graph and bar graph view of series SER1 as the last objects in SP1. If “offset = last” is omitted, the objects will be inserted before SP1.

To refer to a child object of a spool, you must specify the object’s path. For instance, given a spool SPOOL01 containing the spool SP1, which in turn contains a graph G2:

```
spool01.insert(loc=sp1/g2) tab1
```

inserts TAB1 before graph G2 in spool SP1, and moves the remaining objects down.

Cross-references

For additional discussion of spools see [“Spool Objects,” beginning on page 554](#) in the *User’s Guide I*. See also [Spool::append \(p. 410\)](#) and [Spool::remove \(p. 421\)](#).

label	Spool Procs
-------	-----------------------------

Display or change the label view of a spool object, including the last modified date and display name (if any).

Syntax

```
spool_name.label(options) [text]
```

Options

When used with options or the text argument, `label` displays the current spool label. The second version may be used to modify the label. 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 <i>text</i> .
s	Sets the source field to <i>text</i> .
u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of SP1 with “Data from CPS 1988 March File”:

```
sp1.label(r)
sp1.label(r) Data from CPS 1988 March File
```

To append additional remarks to SP1, and then to print the label view:

```
sp1.label(r) Log of hourly wage
sp1.label(p)
```

To clear and then set the units field, use:

```
sp1.label(u) Millions of bushels
```

Cross-references

See “Labeling Objects” on page 72 of the *User’s Guide I* for a discussion of labels. See also [Spool::displayname](#) (p. 412).

leftmargin	Spool Procs
------------	-----------------------------

Changes the left margin size of the spool or of a specified embedded spool.

Syntax

```
spool_name.leftmargin(options) size_arg
```

where *size_arg* is the new margin value specified in virtual inches.

Options

<code>obj = <i>object_opt</i></code>	where <i>object_arg</i> is the name or position of the embedded spool for which you wish to set a margin.
--------------------------------------	---

Examples

```
spool01.leftmargin 0.01
```

sets the left margin for SPOOL01 to 0.01 virtual inch,

```
spool01.topmargin(obj=sp1) 0.02
```

changes the left margin in the embedded spool SP1 to 0.02 virtual inches.

Cross-references

For additional discussion of spools see “[Spool Objects](#),” beginning on page 554 in the *User’s Guide I*. See also [Spool::horizindent](#) (p. 414), [Spool::topmargin](#) (p. 422), and [Spool::vertindent](#) (p. 423).

move	Spool Procs
------	-----------------------------

Move an object in a spool.

Syntax

```
spool_name.move(options) object_arg
```

where *object_arg* is the object to be moved specified as an integer position in the spool or the name of an existing object in the spool. The *options* specify the destination position. If neither a location nor offset are specified in the *options*, the object will be moved to the end of the spool.

Options

loc = <i>arg</i>	<i>arg</i> may be an integer position in the spool or the name of an existing object in the spool. The object will be moved before or after <i>arg</i> , as specified by the offset option below. An object name must include its path if it is a child of another spool. For example, use “spool1/gr1” to specify a graph GR1 in spool SPOOL1.
offset = <i>arg</i> (before, after, first, last), default = “before”	<i>arg</i> indicates where the object should be moved relative to the object specified in the “loc = ” option above. <i>arg</i> may be “before” or “after”. In addition, if the location specified by the “loc = ” option corresponds to a spool object, <i>arg</i> may be “first” or “last” (default), where the object will be positioned as the first or last object in the spool object specified.

Examples

To move the first object in SPOOL01 to the end of the spool:

```
spool01.move 1
```

To move TAB1 to the beginning of SPOOL01:

```
spool01.move(offset=first) tab1
```

Given objects GR1 and TAB1,

```
spool01.move(loc=gr1) tab1
```

moves TAB1 to the current location of GR1. All objects from GR1 onward are pushed down the list of objects.

Alternately, if SP1 is an embedded spool.

```
spool01.move(loc=sp1, offset=last) 3
```

moves the third object to the end of SP1. If “offset = last” is omitted, the object will be moved to just before SP1.

To refer to a child object of a spool, you must specify the object’s path. For instance, given a spool SPOOL01 containing the spool SP1 which in turn contains the graph G2:

```
spool01.move(loc=sp1/g2) tab1
```

moves TAB1 before graph G2 in spool SP1, and moves the remaining objects down.

Cross-references

For additional discussion of spools see “[Spool Objects,](#)” [beginning on page 554](#) in the *User’s Guide I*.

See also [Spool::insert](#) (p. 415).

name	Spool Procs
------	-----------------------------

Rename an object in a spool.

Syntax

```
spool_name.name object_arg new_name
```

where *object_arg* is the name or the position of the object to be renamed, and *new_name* specifies the new name. *new_name* should follow EViews’ standard naming conventions. Note that the case will be discarded; for case-sensitive names, use the [Spool::displayname](#) (p. 412) command.

Examples

```
spool01.name untitled01 tab1
```

renames the object UNTITLED01 to TAB1.

Cross-references

For additional discussion of spools see “[Spool Objects,](#)” [beginning on page 554](#) in the *User’s Guide I*.

See also [Spool::displayname](#) (p. 412).

options	Spool Procs
---------	-----------------------------

Set the display options for the spool object.

Syntax

`spool_name.options option_list`

where *option_list* contains one or more of the options listed below.

Options

tree / -tree	[Display / Hide] the tree window.
borders / -borders	[Display / Hide] borders around the child objects.
titles / -titles	[Display / Hide] the titles or names of child objects.
comments / -comments	[Display / Hide] the comments of child objects.
displaynames / -displaynames	Show the [display names / unique names] of child objects.
margins / -margins	[Apply / Don't apply] spool margins to the child objects.

Each option may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Examples

```
spool01.options -tree margins titles displaynames
```

removes the tree pane from the window, uses the global spool margins, turns on titles, and uses the display name for the title.

Cross-references

For additional discussion of spools see “[Spool Objects,](#)” beginning on page 554 in the *User’s Guide I*. See also [Spool::name](#) (p. 419), [Spool::displayname](#) (p. 412) and [Spool::label](#) (p. 416).

print	Spool Procs
-------	-----------------------------

Print an object in a spool.

The object will be printed to the location specified by the current printer settings.

Syntax

```
spool_name.print object_arg
```

where *object_arg* is the name or the position of the object to be printed.

Examples

```
spool01.print tab1
```

prints the object TAB1 found in SPOOL01.

Cross-references

For additional discussion of spools see “[Spool Objects](#),” beginning on page 554 in the *User’s Guide I*.

See also [print](#) (p. 762) and [Spool::extract](#) (p. 412).

remove	Spool Procs
--------	-----------------------------

Remove objects from a spool.

Syntax

```
spool_name.remove object_list
```

where *object_list* is a list of objects to be removed from the spool.

Examples

```
spool01.remove tab1 state/city
```

removes table object TAB1 from SPOOL01. Also removes the CITY object from the STATE spool, which is a child of SPOOL01. Note that a path is required for child objects. For instance, if TAB1 is a child of another object such as STATE, nothing will be removed.

Cross-references

For additional discussion of spools see “[Spool Objects](#),” beginning on page 554 in the *User’s Guide I*. See also [Spool::append](#) (p. 410) and [Spool::insert](#) (p. 415).

spool	Spool Declaration
-------	-----------------------------------

Declare a spool object.

Syntax

```
spool spool_name
```

where *spool_name* is the name to be given the new object.

Examples

```
spool myspool
```

Cross-references

For additional discussion of spools see [“Spool Objects,” beginning on page 554](#) in the *User’s Guide I*.

tablemode	Spool Procs
------------------	-----------------------------

Set display mode for tables and text objects in the spool.

Syntax

```
spool_name.tablemode(options) [size_arg]
```

where *size_arg* is an optional size argument (in virtual inches) used for the “variablelimit” mode, and *options* may be used to specify the mode. If *size_arg* is not provided, the default EViews setting will be used.

Options

type = *arg*

where *arg* is “variable” or “variablelimit” (*default*).

The “variablelimit” mode may be used to specify the maximum size of table objects in the spool, while “variable” allows tables to be displayed at their native sizes.

Examples

```
spool01.tablemode(type=variablelimit) 5
```

sets all table to be displayed with a maximum width of 5 virtual inches, while

```
spool01.tablemode(type=variable)
```

displays tables at their original sizes.

Cross-references

For additional discussion of spools see [“Spool Objects,” beginning on page 554](#) in the *User’s Guide I*. See also [Spool::graphmode](#) (p. 413).

topmargin	Spool Procs
------------------	-----------------------------

Changes the top margin size of the spool or of a specified embedded spool.

Syntax

```
spool_name.topmargin(options) size_arg
```

where *size_arg* is the new margin value specified in virtual inches.

Options

`obj = object_opt` where *object_arg* is the name or position of the embedded spool for which you wish to set a margin.

Examples

```
spool01.topmargin 0.01
```

sets the top margin for SPOOL01 to 0.01 virtual inch,

```
spool01.topmargin(obj=sp1) 0.02
```

changes the top margin in the embedded spool SP1 to 0.02 virtual inches.

Cross-references

For additional discussion of spools see “[Spool Objects](#),” beginning on page 554 in the *User’s Guide I*. See also [Spool::vertindent](#) (p. 423), [Spool::vertspacing](#) (p. 424), and [Spool::horizindent](#) (p. 414).

vertindent	Spool Procs
------------	-----------------------------

Change the vertical indentation size for objects in the spool.

Syntax

```
spool_name.vertindent object_arg size_arg
```

where *object_arg* is the name or the position of a specific object to which you wish to apply indenting, and *size_arg* is an new indentation in virtual inches.

Examples

```
spool01.vertindent 1 0.02
```

```
spool01.vertindent tab1 0.02
```

change the indentation for the first object and for TAB1 to 0.02 virtual inches.

To refer to a child object of a spool, you must specify the object’s path. For instance, given a spool SPOOL01 containing the spool SP1 which in turn contains the graph G2:

```
spool01.vertindent sp1/g 0.03
```

also changes the vertical indentation of G2 in the embedded spool SP1 to 0.03 virtual inches, while

```
spool01.vertindent sp1 0.03
```

sets the indentation for SP1 to 0.03.

Cross-references

For additional discussion of spools see “[Spool Objects,](#)” [beginning on page 554](#) in the *User’s Guide I*. See also [Spool::topmargin](#) (p. 422), [Spool::vertspacing](#) (p. 424), and [Spool::horizindent](#) (p. 414).

vertspacing	Spool Procs
-------------	-----------------------------

Changes the amount of vertical spacing for objects in the spool or in a specified embedded spool.

Syntax

`spool_name.vertspacing(options) size_arg`

where *size_arg* is an new spacing in virtual inches. By default, spacing will be set for all objects in the spool.

Options

`obj = object_opt` where *object_opt* is the name or the position of a specific embedded spool for which you wish to set spacing.

Examples

```
spool01.vertspacing 0.05
```

specifies the vertical spacing for all objects in the spool at 0.05 vertical inches.

```
spool01.vertspacing(obj=sp1) 0.05
```

sets the vertical spacing at 0.05 only for the objects in the embedded spool SP1.

Cross-references

For additional discussion of spools see “[Spool Objects,](#)” [beginning on page 554](#) in the *User’s Guide I*. See also [Spool::vertindent](#) (p. 423), and [Spool::topmargin](#) (p. 422).

width	Spool Procs
-------	-----------------------------

Changes the width (and height) of objects in the spool.

Syntax

`spool_name.width(options) [size_arg]`

where *size_arg* is an optional size in virtual inches. By default, widths will be set for all objects in the spool, if possible (*i.e.*, the graph object is not specified as fixed width, and the width is within limits defined by the current display mode; see “[graphmode,](#)” [on page 413](#) and “[tablemode,](#)” [on page 422](#), for details).

Heights are set proportional to the width to maintain the original aspect ratio.

If *size_arg* is not provided, the objects will be set to their default sizes.

Options

`obj = object_opt` where *object_opt* is the name or the position of a specific object or embedded spool to which you wish to apply sizing.

`type = arg` where *arg* specifies a restricted subset of objects to be resized: “graph”, “table”, “text”.

If the specified object is an embedded spool, all of its objects will be sized accordingly.

Examples

```
spool01.width 1
```

resizes all objects in the spool to 1 virtual inch, while

```
spool01.width(obj=1) 2
```

```
spool01.width(obj=tab1) 2
```

changes the widths of the first object and TAB1 to 2 virtual inches. The heights of the objects will change proportionately.

```
spool01.width(obj=1)
```

```
spool01.width(obj=tab1)
```

resets the sizes of the objects to their defaults.

To refer to a child object of a spool, you must specify the object’s path. For instance, given a spool SPOOL01 containing the spool SP1 which in turn contains the graph G2:

```
spool01.width(obj=sp1/g2) 2
```

also changes the width of G2 in the embedded spool SP1 to 2 virtual inches, while

```
spool01.width(obj=sp1) 3
```

sets the width for all of the objects in SP1 to 3 virtual inches.

```
spool01.width(type=graph) 2
```

sets the widths of graphs to 2 virtual inches.

Cross-references

For additional discussion of spools see [“Spool Objects,” beginning on page 554](#) in the *User’s Guide I*.

Sspace

State space object. Estimation and evaluation of state space models using the Kalman filter.

Sspace Declaration

[sspace](#)create sspace object (p. 446).

To declare a sspace object, use the `sspace` keyword, followed by a valid name.

Sspace Method

[ml](#)maximum likelihood estimation or filter initialization (p. 441).

Sspace Views

[cellipse](#)Confidence ellipses for coefficient restrictions (p. 430).

[coefcov](#)coefficient covariance matrix (p. 432).

[endog](#)table or graph of actual signal variables (p. 433).

[grads](#)examine the gradients of the log likelihood (p. 435).

[label](#)label information for the state space object (p. 435).

[output](#)table of estimation results (p. 442).

[residcor](#)standardized one-step ahead residual correlation matrix (p. 443).

[residcov](#)standardized one-step ahead residual covariance matrix (p. 443).

[resids](#)one-step ahead actual, fitted, residual graph (p. 444).

[results](#)table of estimation and filter results (p. 444).

[signalgraphs](#)display graphs of signal variables (p. 445).

[spec](#)text representation of state space specification (p. 446).

[statefinal](#)display the final values of the states or state covariance (p. 447).

[stategraphs](#)display graphs of state variables (p. 447).

[stateinit](#)display the initial values of the states or state covariance (p. 448).

[structure](#)examine coefficient or variance structure of the specification
(p. 449).

[wald](#)Wald coefficient restriction test (p. 450).

Sspace Procs

[append](#)add line to the specification (p. 430).

[displayname](#)set display name (p. 432).

[forecast](#)perform state and signal forecasting (p. 433).

[makeendog](#)make group containing actual values for signal variables (p. 436).

[makefilter](#)make new Kalman Filter (p. 437).

[makegrads](#)make group containing the gradients of the log likelihood (p. 438).

[makemodel](#)make a model object containing equations in sspace (p. 438).

[makesignals](#)make group containing signal and residual series (p. 439).

[makestates](#) make group containing state series (p. 440).
[updatecoefs](#) update coefficient vector(s) from sspace (p. 450).

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\(t\)](#) 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.
[@totalobs](#) sum of “[@eqregobs](#)” from each equation.

Vectors and Matrices

[@coefcov](#) covariance matrix for coefficients of equation.
[@coefs](#) coefficient vector.
[@residcov](#) (sym) covariance matrix of the residuals.
[@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_signalsematrix or vector of the standard errors of the one-step ahead predicted signals.
@pred_errmatrix or vector of one-step ahead prediction errors.
@pred_errcovmatrix 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_errsematrix or vector of the standard errors of the one-step ahead prediction errors.
@pred_errstdmatrix or vector of standardized one-step ahead prediction errors.
@pred_statematrix or vector of one-step ahead predicted states.
@pred_statecovmatrix where each row is the **@vech** of the one-step ahead predicted state covariance.
@pred_statesematrix or vector of the standard errors of the one-step ahead predicted states.
@pred_stateerrmatrix or vector of one-step ahead predicted state errors.
@curr_errmatrix or vector of filtered error estimates.
@curr_gainmatrix or vector where each row is the **@vec** of the Kalman gain.
@curr_statematrix or vector of filtered states.
@curr_statecovmatrix where every row is the **@vech** of the filtered state covariance.
@curr_statesematrix or vector of the standard errors of the filtered state estimates.
@sm_signalmatrix or vector of smoothed signal estimates.
@sm_signalcovmatrix where every row is the **@vech** of the smoothed signal covariance.
@sm_signalsematrix or vector of the standard errors of the smoothed signals.
@sm_signalerrmatrix or vector of smoothed signal error estimates.
@sm_signalerrcovmatrix where every row is the **@vech** of the smoothed signal error covariance.
@sm_signalerrse ...matrix or vector of the standard errors of the smoothed signal error.
@sm_signalerrstd ..matrix or vector of the standardized smoothed signal errors.
@sm_statematrix or vector of smoothed states.
@sm_statecovmatrix where each row is the **@vech** of the smoothed state covariances.
@sm_statesematrix or vector of the standard errors of the smoothed state.
@sm_stateerrmatrix or vector of the smoothed state errors.

`@sm_stateerrcov` .. matrix where each row is the `@vech` of the smoothed state error covariance.

`@sm_stateerrse` matrix or vector of the standard errors of the smoothed state errors.

`@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
```

Sspace Entries

The following section provides an alphabetical listing of the commands associated with the “Sspace” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

append	Sspace Procs
--------	------------------------------

Append a specification line to a sspace.

Syntax

```
sspace_name.append text
```

Type the text to be added after the `append` keyword.

Examples

```
vector(2) svec0=0
sspace1.append @mprior svec0
```

appends a line in the state space object SSPACE1 instructing EViews to use the zero vector SVEC0 as initial values for the state vector.

cellipse	Sspace Views
----------	------------------------------

Confidence ellipses for coefficient restrictions.

The `cellipse` view displays confidence ellipses for pairs of coefficient restrictions for an estimation object.

Syntax

```
sspace_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

<code>ind = arg</code>	Specifies whether and how to draw the individual coefficient intervals. The default is “ <code>ind = line</code> ” which plots the individual coefficient intervals as dashed lines. “ <code>ind = none</code> ” does not plot the individual intervals, while “ <code>ind = shade</code> ” plots the individual intervals as a shaded rectangle.
<code>size = number</code> (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.
<code>dist = arg</code>	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. “ <code>dist = f</code> ” forces use of the F -distribution, while “ <code>dist = c</code> ” uses the χ^2 distribution.
<code>p</code>	Print the graph.

Examples

The two commands:

```
s1.cellipse c(1), c(2), c(3)
s1.cellipse 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).

```
s1.cellipse(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 “[Confidence Ellipses](#)” on page 142 of the *User’s Guide II* for discussion.

See also `Sspace::wald` (p. 450).

coefcov[Sspace Views](#)

Coefficient covariance matrix.

Displays the covariances of the coefficient estimates for an estimated state space object.

Syntax

```
sspace_name.coefcov(options)
```

Options

p	Print the coefficient covariance matrix.
---	--

Examples

```
ss1.coefcov
```

displays the coefficient covariance matrix for state space object SS1 in a window. To store the coefficient covariance matrix as a sym object, use “@coefcov”:

```
sym eqcov = ss1.@coefcov
```

Cross-references

See also [Coef::coef](#) (p. 16) and [Sspace::spec](#) (p. 446).

displayname[Sspace Procs](#)

Display name for state space objects.

Attaches a display name to a state space object which may be used to label output in place of the standard state space object name.

Syntax

```
sspace_name.displayname display_name
```

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in state space object names.

Examples

```
ss1.displayname Hours Worked  
ss1.label
```

The first line attaches a display name “Hours Worked” to the state space object SS1, and the second line displays the label view of SS1, including its display name.

Cross-references

See “Labeling Objects” on page 72 of the *User’s Guide I* for a discussion of labels and display names.

See also [Sspace::label](#) (p. 435).

endog	Sspace Views
-------	------------------------------

Displays a spreadsheet or graph view of the endogenous variables.

Syntax

```
sspace_name.endog(options)
```

Options

g	Multiple line graphs of the solved endogenous series.
p	Print the table of solved endogenous series.

Examples

```
ss1.endog(g,p)
```

prints the graphs of the solved endogenous series.

Cross-references

See also [Sspace::makeendog](#) (p. 436) and [Sspace::sspace](#) (p. 446).

forecast	Sspace Procs
----------	------------------------------

Computes (*n*-period ahead) dynamic 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

```
sspace_name.forecast(options) keyword1 names1 [keyword2 names2] [keyword3
names3] ...
```

You should enter a *type*-keyword followed by a list of names for the target series or a wild-card expression, and if desired, additional *type*-keyword and target pairs. The following are valid keywords: “@STATE”, “@STATESE”, “@SIGNAL”, “@SIGNALSE”. The first two keywords instruct EViews to forecast the state series and the values of the state standard error

series. The latter two keywords 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, 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. In addition, the “*” wildcard expression may not be used for forecasting signal variables since this would overwrite the original data.

Options

<code>i = arg</code> (<i>default</i> = “o”)	State initialization options: “o” (one-step), “e” (diffuse), “u” (user-specified), “s” (smoothed).
<code>m = arg</code> (<i>default</i> = “d”)	Basic forecasting method: “n” (<i>n</i> -step ahead forecasting), “s” (smoothed forecasting), “d” (dynamic forecasting).
<code>mprior =</code> <i>vector_name</i>	Name of state initialization (use if option “i = u” is specified).
<code>n = arg</code> (<i>default</i> = 1)	Number of <i>n</i> -step forecast periods (only relevant if <i>n</i> -step forecasting is specified using the <i>method</i> option).
<code>vprior =</code> <i>sym_name</i>	Name of state covariance initialization (use if option “i = u” is specified).
<code>p</code>	Print view.

Examples

The following command performs *n*-step forecasting of the signals and states from a `sspace` object:

```
ssl.forecast(m=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

See `Sspace::makemodel` (p. 438), and `solve` (p. 782).

State space forecasting is described in [Chapter 35. “State Space Models and the Kalman Filter,” on page 383](#) of the *User’s Guide II*. For additional discussion of wildcards, see [Appendix C. “Wildcards,” on page 775](#) of the *User’s Guide I*.

grads	Sspace Views
-------	------------------------------

Gradients of the objective function.

Displays the gradients of the objective function (where available) for an estimated sspace object.

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. Evaluating the gradients at current coefficient values allows you to examine 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

```
sspace_name.grads(options)
```

Options

g	Display multiple graph showing the gradients of the objective function with respect to the coefficients evaluated at each observation.
t (default)	Display spreadsheet view of the values of the gradients of the objective function with respect to the coefficients evaluated at each observation.
p	Print results.

Examples

To show a summary view of the gradients:

```
ss1.grads
```

To display and print the table view:

```
ss1.grads(t, p)
```

Cross-references

See also [Sspace::makegrads](#) (p. 438).

label	Sspace Views Sspace Procs
-------	---

Display or change the label view of the state space object, including the last modified date and display name (if any).

As a procedure, `label` changes the fields in the state space object label.

Syntax

```
sspace_name.label
sspace_name.label(options) [text]
```

Options

The first version of the command displays the label view of the state space object. The second version may be used to modify the label. 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 <i>text</i> .
s	Sets the source field to <i>text</i> .
u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of SS1 with “Data from CPS 1988 March File”:

```
ss1.label(r)
ss1.label(r) Data from CPS 1988 March File
```

To append additional remarks to SS1, and then to print the label view:

```
ss1.label(r) Log of hourly wage
ss1.label(p)
```

To clear and then set the units field, use:

```
ss1.label(u) Millions of bushels
```

Cross-references

See “Labeling Objects” on page 72 of the *User’s Guide I* for a discussion of labels.

See also [Sspace::displayname](#) (p. 432).

makeendog	Sspace Procs
-----------	------------------------------

Make a group out of the endogenous series.

Syntax

```
sspace_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.

Examples

```
ssl.makeendog grp_v1
```

creates a group named GRP_V1 that contains the endogenous series in SS1.

Cross-references

See also [Sspace::endog \(p. 433\)](#) and [Model::makegroup \(p. 282\)](#).

makefilter	Sspace Procs
-------------------	------------------------------

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_name.makefilter [filter_name]
```

If you provide a name for the `sspace` object in parentheses after the keyword, EViews will quietly create the named object in the workfile. If you do not provide a name, EViews will open an untitled `sspace` window if the command is executed from the command line.

Examples

```
ssl.makefilter kfilter
```

creates a new `sspace` object named KFILTER, containing the specification in SS1 with estimated parameter values substituted for coefficient elements.

Cross-references

See [Chapter 35. “State Space Models and the Kalman Filter,”](#) on page 383 of the *User’s Guide II* for details on state space models.

See also [Sspace::makesignals \(p. 439\)](#) and [Sspace::makestates \(p. 440\)](#).

makegrads	Sspace Procs
------------------	------------------------------

Make a group containing individual series which hold the gradients of the objective function.

Syntax

```
sspace_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

<code>n = <i>arg</i></code>	Name of group object to contain the series.
-----------------------------	---

Examples

```
ssl.grads(n=out)
```

creates a group named OUT containing series named GRAD01, GRAD02, and GRAD03.

```
ssl.grads(n=out) g1 g2 g3
```

creates the same group, but names the series G1, G2 and G3.

Cross-references

See also [Sspace::grads](#) (p. 435).

makemodel	Sspace Procs
------------------	------------------------------

Make a model from a state space object.

Syntax

```
sspace_name.makemodel(name) assign_statement
```

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

```
sspace.makemodel(sysmod) @prefix s_
```

makes a model named SYSMOD from the estimated system. SYSMOD includes an assignment statement “ASSIGN @PREFIX S_”. Use the command “show sysmod” or “sysmod.spec” to open the SYSMOD window.

Cross-references

See [Chapter 36. “Models,” on page 407](#) of the *User’s Guide II* for a discussion of specifying and solving models in EViews.

See also [Sspace::append \(p. 430\)](#), [Sspace::makefilter \(p. 437\)](#), and [Model::solve \(p. 287\)](#).

makeresids	Sspace Procs
-------------------	------------------------------

NOTE: `makeresids` is no longer supported for the `sspace` object— see [Sspace::makesignals \(p. 439\)](#) for replacement routines.

makesignals	Sspace Procs
--------------------	------------------------------

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

`name.makesignals(options) [name_spec]`

Follow the object name with a period and the `make signal` 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

<code>t = output_type</code> (<i>default</i> = pred)	Defines output type: one-step ahead signal predictions (“pred”), RMSE of the one-step ahead signal predictions (“predse”, “residse”), error in one-step ahead signal predictions (“resid”), standardized one-step ahead prediction residual (“stdresid”), smoothed signals (“smooth”), RMSE of the smoothed signals (“smoothse”), estimate of the disturbances (“disturb”), RMSE of the estimate of the disturbances (“disturbse”), standardized estimate of the disturbances (“stddisturb”).
<code>n = group_name</code>	Name of group to hold newly created series.

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 the series SIG1, SIG2, and SIG3.

Cross-references

See [Chapter 35. “State Space Models and the Kalman Filter,” on page 383](#) of the *User’s Guide II* for details on state space models. For additional discussion of wildcards, see [Appendix C. “Wildcards,” on page 775](#) of the *User’s Guide I*.

See also [Sspace::forecast \(p. 433\)](#), [Sspace::makefilter \(p. 437\)](#), and [Sspace::makestates \(p. 440\)](#).

makestates	Sspace Procs
------------	------------------------------

Generate state series or state standard error series from an estimated sspace object.

Options allow you to generate one-step ahead, filtered, or smoothed values for the states and the state standard errors.

Syntax

```
sspace_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 a 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

<code>t = output_type</code> (default = “pred”)	Defines output type: one-step ahead state predictions (“pred”), RMSE of the one-step ahead state predictions (“predse”), error in one-step ahead state predictions (“resid”), RMSE of the one-step ahead state prediction (“residse”), filtered states (“filt”), RMSE of the filtered states (“filtse”), standardized one-step ahead prediction residual (“stdresid”), smoothed states (“smooth”), RMSE of the smoothed states (“smoothse”), estimate of the disturbances (“disturb”), RMSE of the estimate of the disturbances (“disturbse”), standardized estimate of the disturbances (“stddisturb”).
<code>n = group_name</code>	Name of group to hold newly created series.

Examples

```
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 [Chapter 35. “State Space Models and the Kalman Filter,”](#) on page 383 of the *User’s Guide II* for details on state space models. For additional discussion of wildcards, see [Appendix C. “Wildcards,”](#) on page 775 of the *User’s Guide I*.

See also [Sspace::forecast](#) (p. 433), [Sspace::makefilter](#) (p. 437) and [Sspace::makesignals](#) (p. 439).

ml	Sspace Method
----	-------------------------------

Maximum likelihood estimation of state space models.

Syntax

```
sspace_name.ml(options)
```

Options

b	Use Berndt-Hall-Hausman (BHHH) algorithm (default is Marquardt).
m = <i>integer</i>	Set maximum number of iterations.
c = <i>scalar</i>	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 [Chapter 35. “State Space Models and the Kalman Filter,” on page 383](#) of the *User’s Guide II* for a discussion of user specified state space models.

output	Sspace Views
--------	------------------------------

Display estimation output.

`output` changes the default object view to display the estimation output (equivalent to using `Sspace::results` (p. 444)).

Syntax

```
sspace_name.output
```

Options

p	Print estimation output for estimation object
---	---

Examples

The `output` keyword may be used to change the default view of an estimation object. Entering the command:

```
ss1.output
```

displays the estimation output for state space object SS1.

Cross-references

See [Sspace::results](#) (p. 444).

residcor	Sspace Views
-----------------	------------------------------

Residual correlation matrix.

Displays the correlations of the residuals from each equation in the sspace object. The sspace object residuals used in the calculation are the standardized, one-step ahead signal forecast errors.

Syntax

```
sspace_name.residcor(options)
```

Options

p	Print the correlation matrix.
---	-------------------------------

Examples

```
ss1.residcor
```

displays the residual correlation matrix of sspace object SS1.

Cross-references

See also [Sspace::residcov](#) (p. 443) and [Sspace::makeresids](#) (p. 439).

residcov	Sspace Views
-----------------	------------------------------

Residual covariance matrix.

Displays the covariances of the residuals from each equation in the sspace object. The sspace object residuals used in the calculation are the standardized, one-step ahead signal forecast errors.

Syntax

```
sspace_name.residcov(options)
```

Options

p	Print the covariance matrix.
---	------------------------------

Examples

```
ss1.residcov
```

displays the residual covariance matrix of SS1.

Cross-references

See also [Sspace::residcor](#) (p. 443) and [Sspace::makesresids](#) (p. 439).

resids	Sspace Views
--------	------------------------------

Display residuals.

`resids` allows you to display and actual-fitted-residual graph using the one-step ahead estimates.

Syntax

`sspace_name.resids(options)`

Options

p	Print the table/graph.
---	------------------------

Examples

```
ss1.resids
```

displays a graph of the actual, fitted, and residual series for the `sspace` object SS1.

Cross-references

See [Chapter 35. “State Space Models and the Kalman Filter,”](#) on page 383 of the *User’s Guide II* for a discussion of state space models.

See also [Sspace::makesresids](#) (p. 439).

results	Sspace Views
---------	------------------------------

Displays the results view of an estimated state space object.

Syntax

`sspace_name.results(options)`

Options

p	Print the view.
---	-----------------

Examples

```
ss1.results(p)
```

displays and prints the results of the `sspace` object SS1.

Cross-references

See [Chapter 35. “State Space Models and the Kalman Filter,”](#) on page 383 of the *User’s Guide II* for a discussion of state space models.

signalgraphs	Sspace Views
--------------	------------------------------

Graph signal series.

Display graphs of a set of signal series computed using the Kalman filter.

Syntax

```
sspace_name.signalgraphs(options)
```

Options

<code>t = output_type</code> (default = “pred”)	Defines output type: “pred” (one-step ahead signal predictions), “predse” (RMSE of the one-step ahead signal predictions), “resid” (error in one-step ahead signal predictions), “residse” (RMSE of the one-step ahead signal prediction; same as “predse”), “stdresid” (standardized one-step ahead prediction residual), “smooth” (smoothed signals), “smoothse” (RMSE of the smoothed signals), “disturb” (estimate of the disturbances), “disturbse” (RMSE of the estimate of the disturbances), “stddisturb” (standardized estimate of the disturbances).
--	--

Examples

```
ss1.signalgraphs(t=smooth)
ss1.signalgraphs(t=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 [Chapter 35. “State Space Models and the Kalman Filter,”](#) on page 383 of the *User’s Guide II* for a discussion of state space models.

See also [Sspace::stategraphs](#) (p. 447), [Sspace::makesignals](#) (p. 439) and [Sspace::makestates](#) (p. 440).

spec	Sspace Views
------	------------------------------

Display the text specification view for sspace objects.

Syntax

`sspace_name.spec(options)`

Options

p	Print the specification text.
---	-------------------------------

Examples

```
ss1.spec
```

displays the specification of the sspace object SS1.

Cross-references

See also [Sspace::append](#) (p. 430).

sspace	Sspace Declaration
--------	------------------------------------

Declare state space object.

Syntax

`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 35. “State Space Models and the Kalman Filter,”](#) on page 383 of the *User’s Guide II* for a discussion of state space models.

[Sspace::append](#) (p. 430) may be used to add lines to an existing sspace object. See also [Sspace::ml](#) (p. 441) for estimation of state space models.

statefinal	Sspace Views
------------	------------------------------

Display final state values.

Show the one-step ahead state predictions or the state prediction covariance matrix at the final values $(T + 1 | T)$, where T is the last observation in the estimation sample. By default, EViews shows the state predictions.

Syntax

sspace_name.statefinal(options)

Options

c	Display the state prediction covariance matrix.
p	Print the view.

Examples

```
ssl.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 [Chapter 35. “State Space Models and the Kalman Filter,”](#) on page 383 of the *User’s Guide II* for a discussion of state space models.

See also [Sspace::stateinit](#) (p. 448).

stategraphs	Sspace Views
-------------	------------------------------

Display graphs of a set of state series computed using the Kalman filter.

Syntax

sspace_name.stategraph(options)

Options

<code>t = output_type</code> (default = “pred”)	Defines output type: “pred” (one-step ahead signal predictions), “predse” (RMSE of the one-step ahead signal predictions), “resid” (error in one-step ahead signal predictions), “residse” (RMSE of the one-step ahead signal prediction; same as “predse”), “stdresid” (standardized one-step ahead prediction residual), “smooth” (smoothed signals), “smoothse” (RMSE of the smoothed signals), “disturb” (estimate of the disturbances), “disturbse” (RMSE of the estimate of the disturbances), “stddisturb” (standardized estimate of the disturbances).
--	--

Other options

<code>p</code>	Print the view.
----------------	-----------------

Examples

```
ssl.stategraphs(t=filt)
```

displays a graph view containing the filtered state values.

Cross-references

See [Chapter 35. “State Space Models and the Kalman Filter,” on page 383](#) of the *User’s Guide II* for a discussion of state space models.

See also [Sspace::signalgraphs \(p. 445\)](#), [Sspace::makesignals \(p. 439\)](#) and [Sspace::makestates \(p. 440\)](#).

stateinit	Sspace Views
------------------	------------------------------

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

```
sspace_name.stateinit(options)
```

Options

<code>c</code>	Display the covariance matrix.
<code>p</code>	Print the view.

Examples

```
ssl.stateinit
```

displays a view containing the initial state values (the one-step ahead predictions for the first period).

Cross-references

See [Chapter 35. “State Space Models and the Kalman Filter,”](#) on page 383 of the *User’s Guide II* for a discussion of state space models.

See also [Sspace::statefinal](#) (p. 447).

structure	Sspace Views
-----------	------------------------------

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 (default) or the values of the system transition matrices or covariance.

Syntax

```
sspace_name.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

v	Display the values of the system transition or covariance matrices.
c	Display the system covariance matrix.
p	Print the view.

Examples

```
ssl.structure
```

displays a system transition matrices.

```
ssl.structure 1993q4
```

displays the transition matrices evaluated at 1993Q4.

Cross-references

See [Chapter 35. “State Space Models and the Kalman Filter,”](#) on page 383 of the *User’s Guide II* for a discussion of state space models.

updatecoefs	Sspace Procs
-------------	------------------------------

Update coefficient object values from state space object.

Copies coefficients from the sspace object into the appropriate coefficient vector or vectors in the workfile.

Syntax

sspace_name.updatecoefs

Follow the name of the sspace object by a period and the keyword `updatecoefs`.

Examples

ss1.updatecoefs

places the values of the estimated coefficients from SS1 in the coefficient vector in the workfile.

Cross-references

See also [Coef::coef](#) (p. 16).

wald	Sspace Views
------	------------------------------

Wald coefficient restriction test.

The `wald` view carries out a Wald test of coefficient restrictions for a state space object.

Syntax

sspace_name.wald *restrictions*

Enter the sspace name, followed by a period, and the keyword. You must provide a list of the coefficient restrictions, with joint (multiple) coefficient restrictions separated by commas.

Options

p	Print the test results.
---	-------------------------

Examples

ss1.wald c(2)=0, c(3)=0

tests the null hypothesis that the second and third coefficients in equation SS1 are jointly zero.

```
ss1.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.

Cross-references

See “[Wald Test \(Coefficient Restrictions\)](#)” on [page 145](#) of the *User’s Guide II* for a discussion of Wald tests.

See also [Sspace::cellipse](#) (p. 430).

Sym

Symmetric matrix (symmetric two-dimensional array).

Sym Declaration

symdeclare sym object (p. 471).

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

corcorrelation matrix by columns (p. 455).
covcovariance matrix by columns (p. 458).
eigeneigenvalues calculation for a symmetric matrix (p. 462).
labellabel information for the symmetric matrix (p. 465).
sheetspreadsheet view of the symmetric matrix (p. 471).
statsdescriptive statistics by column (p. 471).

Sym Graph Views

Graph creation types are discussed in detail in “Graph Creation Commands” on page 601.

areaarea graph of the columns of the matrix (p. 603).
bandarea band graph (p. 606).
barbar graph of each column against the row index (p. 609).
boxplotboxplot graph (p. 613).
distplotdistribution graph (p. 615).
dotdot plot graph (p. 622).
errbarerror bar graph view (p. 626).
hilohigh-low(-open-close) chart (p. 628).
lineline graph of each column against the row index (p. 630).
piepie chart view (p. 633).
qqplotquantile-quantile graph (p. 636).
scatscatter diagrams of the columns of the sym (p. 640).
scatmatmatrix of all pairwise scatter plots (p. 644).
scatpairscatterplot pairs graph (p. 647).
seasplotseasonal line graph (p. 651).
spikespike graph (p. 652).

xyarea XY area graph (p. 656).
xybar XY bar graph (p. 659).
xyline XY line graph (p. 661).
xypair XY pairs graph (p. 664).

Sym Procs

displayname set display name (p. 461).
fill fill the elements of the matrix (p. 464).
read import data from disk (p. 466).
setformat set the display format for the sym spreadsheet (p. 468).
setindent set the indentation for the sym spreadsheet (p. 469).
setjust set the justification for the sym spreadsheet (p. 469).
setwidth set the column width in the sym spreadsheet (p. 470).
write export data to disk (p. 472).

Sym Data Members

(i,j) (*i,j*)-th element of the matrix. Simply append “(i,j)” to the matrix name (without a “.”).

Sym Examples

The declaration:

```
sym results(10)
results=3
```

creates the 10×10 matrix RESULTS and initializes each value to be 3. The following assignment statements also create and initialize sym objects:

```
sym copymat=results
sym covmat1=eq1.@coefcov
sym(3,3) count
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:

```
copymat.line
copymat.cov
copymat.stats
```

You can use explicit indices to refer to matrix elements:

```
scalar diagsum=cov1(1,1)+cov1(2,2)+cov(3,3)
```

Sym Entries

The following section provides an alphabetical listing of the commands associated with the “Sym” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

cor	Sym Views
-----	---------------------------

Compute measure of association for the columns in a matrix. You may compute measures related to Pearson product-moment (ordinary) correlations, rank correlations, or Kendall’s tau.

Syntax

```
matrix_name.cor(options) [keywords [@partial z1 z2 z3...]]
```

You should specify keywords indicating the statistics you wish to display from the list below, optionally followed by the keyword `@partial` and a list of conditioning series or groups (for the group view), or the name of a conditioning matrix (for the matrix view). In the matrix view setting, the columns of the matrix should contain the conditioning information, and the number of rows should match the original matrix.

You may specify keywords from one of the four sets (Pearson correlation, Spearman correlation, Kendall’s tau, Uncentered Pearson) corresponding the computational method you wish to employ. (You may not select keywords from more than one set.)

If you do not specify *keywords*, EViews will assume “corr” and compute the Pearson correlation matrix. Note that `cor` is equivalent to the `cov` (p. 458) command with a different default setting.

Note that this view has been expanded considerably from EViews 5.1, but the syntax for the earlier version of the routine is fully compatible with the current version.

Pearson Correlation

cov	Product moment covariance.
corr	Product moment correlation.
sscp	Sums-of-squared cross-products.
stat	Test statistic (<i>t</i> -statistic) for evaluating whether the correlation is zero.
prob	Probability under the null for the test statistic.
cases	Number of cases.
obs	Number of observations.

wgts	Sum of the weights.
------	---------------------

Spearman Rank Correlation

rcov	Spearman's rank covariance.
rcorr	Spearman's rank correlation.
rsscp	Sums-of-squared cross-products.
rstat	Test statistic (t -statistic) for evaluating whether the correlation is zero.
rprob	Probability under the null for the test statistic.
cases	Number of cases.
obs	Number of observations.
wgts	Sum of the weights.

Kendall's tau

taub	Kendall's tau-b.
taua	Kendall's tau-a.
taucd	Kendall's concordances and discordances.
taustat	Kendall's score statistic for evaluating whether the Kendall's tau-b measure is zero.
tauprob	Probability under the null for the score statistic.
cases	Number of cases.
obs	Number of observations.
wgts	Sum of the weights.

Uncentered Pearson

ucov	Product moment covariance.
ucorr	Product moment correlation.
usscp	Sums-of-squared cross-products.
ustat	Test statistic (t -statistic) for evaluating whether the correlation is zero.
uprob	Probability under the null for the test statistic.
cases	Number of cases.
obs	Number of observations.

`wgts` Sum of the weights.

Note that `cases`, `obs`, and `wgts` are available for each of the methods.

Options

<code>wgt = name</code> (optional)	Name of series containing weights.
<code>wgtmethod = arg</code> (default = “sstdev”)	Weighting method (when weights are specified using “weight = ”): frequency (“freq”), inverse of variances (“var”), inverse of standard deviation (“stdev”), scaled inverse of variances (“svar”), scaled inverse of standard deviations (“sstdev”). Only applicable for ordinary (Pearson) calculations. Weights specified by “wgt = ” are frequency weights for rank correlation and Kendall’s tau calculations.
<code>pairwise</code>	Compute using pairwise deletion of observations with missing cases (pairwise samples).
<code>df</code>	Compute covariances with a degree-of-freedom correction for the mean (for centered specifications), and any partial conditioning variables.
<code>multi = arg</code> (default = “none”)	Adjustment to p -values for multiple comparisons: none (“none”), Bonferroni (“bonferroni”), Dunn-Sidak (“dunn”).
<code>outfmt = arg</code> (default = “single”)	Output format: single table (“single”), multiple table (“mult”), list (“list”), spreadsheet (“sheet”). Note that “outfmt = sheet” is only applicable if you specify a single statistic keyword.
<code>out = name</code>	Basename for saving output. All results will be saved in Sym matrices named using keys (“COV”, “CORR”, “SSCP”, “TAUA”, “TAUB”, “CONC” (Kendall’s concurrences), “DISC” (Kendall’s discordances), “CASES”, “OBS”, “WGTS”) appended to the basename (e.g., the covariance specified by “out = my” is saved in the Sym matrix “MYCOV”).
<code>p</code>	Print the result.

Examples

```
sym1.cor
```

displays a 3×3 Pearson correlation matrix for the columns series in MAT1.

```
sym1.cor corr stat prob
```

displays a table containing the Pearson correlation, t -statistic for testing for zero correlation, and associated p -value, for the columns in MAT1.

```
sym1.cor(pairwise) taub taustat tauprob
```

computes the Kendall's tau-b, score statistic, and p -value for the score statistic, using samples with pairwise missing value exclusion.

Cross-references

See also [cov](#) (p. 458), [@cor](#) (p. 847), and [@cov](#) (p. 847).

COV	Sym Views
-----	---------------------------

Compute measure of association for the columns in a matrix. You may compute measures related to Pearson product-moment (ordinary) covariances, rank covariances, or Kendall's tau.

Syntax

```
matrix_name.cov(options) [keywords [@partial z1 z2 z3...]]
```

You should specify keywords indicating the statistics you wish to display from the list below, optionally followed by the keyword `@partial` and a list of conditioning series or groups (for the group view), or the name of a conditioning matrix (for the matrix view). In the matrix view setting, the columns of the matrix should contain the conditioning information, and the number of rows should match the original matrix.

You may specify keywords from one of the four sets (Pearson correlation, Spearman correlation, Kendall's tau, Uncentered Pearson) corresponding the computational method you wish to employ. (You may not select keywords from more than one set.)

If you do not specify *keywords*, EViews will assume “cov” and compute the Pearson covariance matrix. Note that `cov` is equivalent to the [cor](#) (p. 455) command with a different default setting.

Note that this view has been expanded considerably from EViews 5.1, but the syntax for the earlier version of the routine is fully compatible with the current version.

Pearson Correlation

cov	Product moment covariance.
corr	Product moment correlation.
sscp	Sums-of-squared cross-products.
stat	Test statistic (t -statistic) for evaluating whether the correlation is zero.

prob	Probability under the null for the test statistic.
cases	Number of cases.
obs	Number of observations.
wgts	Sum of the weights.

Spearman Rank Correlation

rcov	Spearman's rank covariance.
rcorr	Spearman's rank correlation.
rsscp	Sums-of-squared cross-products.
rstat	Test statistic (t -statistic) for evaluating whether the correlation is zero.
rprob	Probability under the null for the test statistic.
cases	Number of cases.
obs	Number of observations.
wgts	Sum of the weights.

Kendall's tau

taub	Kendall's tau-b.
taua	Kendall's tau-a.
taucd	Kendall's concordances and discordances.
taustat	Kendall's score statistic for evaluating whether the Kendall's tau-b measure is zero.
tauprob	Probability under the null for the score statistic.
cases	Number of cases.
obs	Number of observations.
wgts	Sum of the weights.

Uncentered Pearson

ucov	Product moment covariance.
ucorr	Product moment correlation.
usscp	Sums-of-squared cross-products.
ustat	Test statistic (t -statistic) for evaluating whether the correlation is zero.

<code>uprob</code>	Probability under the null for the test statistic.
<code>cases</code>	Number of cases.
<code>obs</code>	Number of observations.
<code>wgts</code>	Sum of the weights.

Note that `cases`, `obs`, and `wgts` are available for each of the methods.

Options

<code>wgt = name</code> (optional)	Name of series containing weights.
<code>wgtmethod = arg</code> (default = “sstdev”)	<p>Weighting method (when weights are specified using “weight = ”): frequency (“freq”), inverse of variances (“var”), inverse of standard deviation (“stdev”), scaled inverse of variances (“svar”), scaled inverse of standard deviations (“sstdev”).</p> <p>Only applicable for ordinary (Pearson) calculations. Weights specified by “wgt = ” are frequency weights for rank correlation and Kendall’s tau calculations.</p>
<code>pairwise</code>	Compute using pairwise deletion of observations with missing cases (pairwise samples).
<code>df</code>	Compute covariances with a degree-of-freedom correction for the mean (for centered specifications), and any partial conditioning variables.
<code>multi = arg</code> (default = “none”)	Adjustment to p -values for multiple comparisons: none (“none”), Bonferroni (“bonferroni”), Dunn-Sidak (“dunn”).
<code>outfmt = arg</code> (default = “single”)	Output format: single table (“single”), multiple table (“mult”), list (“list”), spreadsheet (“sheet”). Note that “outfmt = sheet” is only applicable if you specify a single statistic keyword.
<code>out = name</code>	Basename for saving output. All results will be saved in Sym matrices named using keys (“COV”, “CORR”, “SSCP”, “TAUA”, “TAUB”, “CONC” (Kendall’s concurrences), “DISC” (Kendall’s discordances), “CASES”, “OBS”, “WGTS”) appended to the basename (e.g., the covariance specified by “out = my” is saved in the Sym matrix “MYCOV”).
<code>p</code>	Print the result.

Examples

```
sym1.cov
```

displays a 3×3 Pearson covariance matrix for the columns series in MAT1.

```
sym1.cov corr stat prob
```

displays a table containing the Pearson covariance, *t*-statistic for testing for zero correlation, and associated *p*-value, for the columns in MAT1.

```
sym1.cov(pairwise) taub taustat tauprob
```

computes the Kendall’s tau-b, score statistic, and *p*-value for the score statistic, using samples with pairwise missing value exclusion.

Cross-references

See also [cor \(p. 455\)](#), [@cor \(p. 847\)](#), and [@cov \(p. 847\)](#).

displayname	Sym Procs
-------------	---------------------------

Display name for symmetric matrix objects.

Attaches a display name to a symmetric matrix object which may be used to label output in place of the standard matrix object name.

Syntax

```
matrix_name.displayname display_name
```

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in matrix object names.

Examples

```
s1.displayname Hours Worked  
s1.label
```

The first line attaches a display name “Hours Worked” to the symmetric matrix object S1, and the second line displays the label view of S1, including its display name.

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels and display names.

See also [Sym::label \(p. 465\)](#).

eigen	Sym Views
-------	---------------------------

Eigenvalues calculation for a symmetric matrix.

Syntax

There are two forms of the `eigen` command.

The first form, which applies when displaying eigenvalue table output or graphs of the ordered eigenvalues, has only options and no command argument.

```
sym_name.eigen(options)
```

The second form, which applies to the graphs of component loadings (specified with the option “out = loadings”) uses an optional argument to determine which components to plot. In this form:

```
sym_name.eigen(options) [graph_list]
```

where the *graph_list* is an optional list of integers and/or vectors containing integers identifying the components to plot. Multiple pairs are handled using the method specified in the “mult = ” option.

If the list of component indices omitted, EViews will plot only first and second components. Note that the order of elements in the list matters; reversing the order of two indices reverses the axis on which each component is displayed.

Options

<code>out = arg</code> (<i>default</i> = “table”)	Output: table of eigenvalue and eigenvector results (“out = table”), graphs of ordered eigenvalues (“graph”), graph of the eigenvectors (“loadings”). Note: when specifying the eigenvalue graph (“out = graph”), the option keywords “scree” (scree graph), “diff” (difference in successive eigenvalues), and “cproport” (cumulative proportion of total variance) may be included to control the output. By default, EViews will display the scree graph. If you specify one or more of the keywords, EViews will construct the graph using only the specified types (<i>i.e.</i> , if you specify “cproport”, a scree plot will not be provided unless requested).
---	---

<code>n = integer</code>	<p>Maximum number of components to retain when presenting table (“out = table”) or eigenvalue graph (“out = graph”) results.</p> <p>The default is to set <i>n</i> to the number of variables.</p> <p>EViews will retain the minimum number satisfying any of: “n =”, “mineig =” or “cproport =”.</p>
<code>mineig = arg</code> (default = 0)	<p>Minimum eigenvalue threshold value: we retain components with eigenvalues that are greater than or equal to the threshold.</p> <p>EViews will retain the minimum number satisfying any of: “n =”, “mineig =” or “cproport =”.</p>
<code>cproport = arg</code> (default = 1)	<p>Cumulative proportion threshold value: we retain <i>k</i>, the number of components required for the sum of the first <i>k</i> eigenvalues exceeds the specified value for the cumulative variance explained proportion.</p> <p>EViews will retain the minimum number satisfying any of: “n =”, “mineig =” or “cproport =”.</p>
<code>eigval = vec_name</code>	Specify name of vector to hold the saved the eigenvalues in workfile.
<code>eigvec = mat_name</code>	Specify name of matrix to hold the save the eigenvectors in workfile.
<code>p</code>	Print results.

Graph Options

<code>scale = arg</code> , (default = “normload”)	Diagonal matrix scaling of the loadings: normalize loadings (“normload”), normalize scores (“normscores”), symmetric weighting (“symmetric”), user-specified power (<i>arg = number</i>).
<code>mult = arg</code> (default = “first”)	Multiple series handling: plot first against remainder (“first”), plot as x-y pairs (“pair”), lower-triangular plot (“lt”).
<code>nocenter</code>	Do not center graphs around the origin.

Examples

```
sym s1 = @cov(g1)
freeze(tab1) s1.eigen(method=cor, eigval=v1, eigvec=m1)
```

The first line creates a group named G1 containing the four series X1, X2, X3, X4. The second line computes the correlation matrix S1 from the series in G1. The final line stores the table view of the eigenvalues and eigenvectors of S1 in a table object named TAB1, the eigenvalues in a vector named V1, and the eigenvectors in a matrix named M1.

Cross-references

See [“Principal Components” on page 397](#) of the *User’s Guide I* for a discussion of principal components analysis on a group of series, which describes a superset of the tools for eigen-value calculations offered by the sym matrix.

fill	Sym Procs
------	---------------------------

Fill a symmetric matrix object with specified values.

Syntax

```
matrix_name.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.*

Running out of values before the object is completely filled is not an error; the remaining cells or observations will be unaffected, unless the “l” option is specified. If, however, you list more values than the object can hold, EViews will not modify any observations and will return an error message.

Options

l	Loop repeatedly over the list of values as many times as it takes to fill the object.
o = integer (default = 1)	Fill the object from the specified element. Default is the first element.

Examples

The commands,

```
sym(2) m1
m1.fill 0, 1, 2
```

create the symmetric matrix:

$$m1 = \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix} \tag{1.3}$$

Cross-references

See [Chapter 18. “Matrix Language,” on page 627](#) of the *User’s Guide I* for a detailed discussion of vector and matrix manipulation in EViews.

label	Sym Views Sym Procs
-------	---------------------------------------

Display or change the label view of the symmetric matrix object, including the last modified date and display name (if any).

As a procedure, `label` changes the fields in the symmetric matrix object label.

Syntax

```
sym_name.label
sym_name.label(options) [text]
```

Options

The first version of the command displays the label view of the symmetric matrix. The second version may be used to modify the label. 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 <i>text</i> .
s	Sets the source field to <i>text</i> .
u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of SYM1 with “Data from CPS 1988 March File”:

```
sym1.label(r)
sym1.label(r) Data from CPS 1988 March File
```

To append additional remarks to SYM1, and then to print the label view:

```
sym1.label(r) Log of hourly wage
sym1.label(p)
```

To clear and then set the units field, use:

```
sym1.label(u) Millions of bushels
```

Cross-references

See “[Labeling Objects](#)” on page 72 of the *User’s Guide I* for a discussion of labels.

See also [Sym::displayname](#) (p. 461).

read	Sym Procs
------	---------------------------

Import data from a foreign disk file into a symmetric matrix.

May be used to import data into an existing workfile from a text, Excel, or Lotus file on disk.

Syntax

`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. 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.

Options

File type options

t = dat, txt	ASCII (plain text) files.
t = wk1, wk3	Lotus spreadsheet files.
t = xls	Excel spreadsheet files.

If you do not specify the “t” option, EViews uses the file name extension to determine the file type. If you specify the “t” option, the file name extension will not be used to determine the file type.

Options for ASCII text files

t	Read data organized by column (transposed). Default is to read by row.
na = text	Specify text for NAs. Default is “NA”.
d = t	Treat tab as delimiter (note: you may specify multiple delimiter options). The <i>default</i> is “d = c” only.
d = c	Treat comma as delimiter.
d = s	Treat space as delimiter.
d = a	Treat alpha numeric characters as delimiter.
custom = symbol	Specify symbol/character to treat as delimiter.
mult	Treat multiple delimiters as one.
rect (default) / norect	[Treat / Do not treat] file layout as rectangular.

<code>skipcol = integer</code>	Number of columns to skip. Must be used with the “rect” option.
<code>skiprow = integer</code>	Number of rows to skip. Must be used with the “rect” option.
<code>comment = symbol</code>	Specify character/symbol to treat as comment sign. Everything to the right of the comment sign is ignored. Must be used with the “rect” option.
<code>singlequote</code>	Strings are in single quotes, not double quotes.
<code>dropstrings</code>	Do not treat strings as NA; simply drop them.
<code>negparen</code>	Treat numbers in parentheses as negative numbers.
<code>allowcomma</code>	Allow commas in numbers (note that using commas as a delimiter takes precedence over this option).

Options for spreadsheet (Lotus, Excel) files

<code>t</code>	Read data organized by column (transposed). Default is to read by row.
<code>letter_number</code> (<i>default</i> = “b2”)	Coordinate of the upper-left cell containing data.
<code>s = sheet_name</code>	Sheet name for Excel 5–8 Workbooks.

Examples

```
m1.read(t=dat,na=.) a:\mydat.raw
```

reads data into matrix M1 from an ASCII file MYDAT.RAW in the A: drive. The data in the file are listed by row, and the missing value NA is coded as a “.” (dot or period).

```
m1.read(t,a2,s=sheet3) cps88.xls
```

reads data into matrix M1 from an Excel file CPS88 in the default directory. The data are organized by column (transposed), the upper left data cell is A2, and the data is read from a sheet named SHEET3.

```
m2.read(a2, s=sheet2) "\\network\dr 1\cps91.xls"
```

reads the Excel file CPS91 into matrix M2 from the network drive specified in the path.

Cross-references

See “Importing Data” on page 95 of the *User’s Guide I* for a discussion and examples of importing data from external files.

For powerful, easy-to-use tools for reading data into a new workfile, see “[Creating a Workfile by Reading from a Foreign Data Source](#)” on [page 41](#) of the *User’s Guide I*, [pageload](#) (p. 750), and [wfopen](#) (p. 802).

See also [Sym::write](#) (p. 472).

setformat	Sym Procs
------------------	---------------------------

Set the display format for cells in a symmetric matrix object spreadsheet view.

Syntax

```
matrix_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.

For symmetric matrices, `setformat` operates on all of the cells in the matrix.

To format numeric values, you should use one of the following format specifications:

<i>g[.precision]</i>	significant digits
<i>f[.precision]</i>	fixed decimal places
<i>c[.precision]</i>	fixed characters
<i>e[.precision]</i>	scientific/float
<i>p[.precision]</i>	percentage
<i>r[.precision]</i>	fraction

In order to set a format that groups digits into thousands, place a “t” after a specified format. For example, to obtain a fixed number of decimal places, with commas used to separate thousands, use “*ft[.precision]*”. To obtain a fixed number of characters with a period used to separate thousands, use “*ct[..precision]*”.

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)”).

Examples

To set the format for all cells in the symmetric matrix to fixed 5-digit precision, simply provide the format specification:

```
m1.setformat f.5
```

Other format specifications include:

```
m1.setformat f(.7)
m1.setformat e.5
```


Cross-references

See [Sym::setwidth \(p. 470\)](#), [Sym::setindent \(p. 469\)](#) and [Sym::setjust \(p. 469\)](#) for details on setting spreadsheet widths, indentation and justification.

setindent	Sym Procs
-----------	---------------------------

Set the display indentation for cells in a symmetric matrix object spreadsheet view.

Syntax

```
matrix_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 spreadsheet (the EViews spreadsheet default font at the time the spreadsheet 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 spreadsheet view is created.

For symmetric matrices, `setindent` operates on all of the cells in the matrix.

Examples

To set the indentation for all the cells in a symmetric matrix object:

```
m1.setindent 2
```

Cross-references

See [Sym::setwidth \(p. 470\)](#) and [Sym::setjust \(p. 469\)](#) for details on setting spreadsheet widths and justification.

setjust	Sym Procs
---------	---------------------------

Set the display justification for cells in a symmetric matrix object spreadsheet view.

Syntax

```
matrix_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.

For symmetric matrices, `setjust` operates on all of the cells in the matrix.

The *format_arg* may be formed using the following:

top / middle / bottom]	Vertical justification setting.
auto / left / cen- ter / right	Horizontal justification setting. “Auto” uses left justifica- tion for strings, and right for numbers.

You may enter one or both of the justification settings. The default settings are taken from the Global Defaults for spreadsheet views.

Examples

```
ml.setjust middle
```

sets the vertical justification to the middle.

```
ml.setjust top left
```

sets the vertical justification to top and the horizontal justification to left.

Cross-references

See [Sym::setwidth \(p. 470\)](#) and [Sym::setindent \(p. 469\)](#) for details on setting spreadsheet widths and indentation.

setwidth	Sym Procs
----------	---------------------------

Set the column width for all columns in a symmetric matrix object spreadsheet.

Syntax

```
matrix_name.setwidth width_arg
```

where *width_arg* specifies the width unit value. The width unit is computed from representative characters in the default font for the current spreadsheet (the EViews spreadsheet default font at the time the spreadsheet 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

```
mat1.setwidth 12
```

sets the width of all columns in symmetric matrix MAT1 to 12 width units.

Cross-references

See [Sym::setindent \(p. 469\)](#) and [Sym::setjust \(p. 469\)](#) for details on setting spreadsheet indentation and justification.

sheet	Sym Views
-------	---------------------------

Spreadsheet view of a symmetric matrix object.

Syntax

```
matrix_name.sheet(options)
```

Options

p	Print the spreadsheet view.
---	-----------------------------

Examples

```
m1.sheet(p)
```

displays and prints the spreadsheet view of symmetric matrix M1.

stats	Sym Views
-------	---------------------------

Descriptive statistics.

Computes and displays a table of means, medians, maximum and minimum values, standard deviations, and other descriptive statistics of each column in the symmetric matrix.

Syntax

```
matrix_name.stats(options)
```

Options

p	Print the stats table.
---	------------------------

Examples

```
mat1.stats
```

displays the descriptive statistics view of symmetric matrix MAT1.

Cross-references

See [“Descriptive Statistics & Tests” on page 306](#) and [page 379](#) of the *User’s Guide I* for a discussion of the descriptive statistics views of series and groups.

sym	Sym Declaration
-----	---------------------------------

Declare a symmetric matrix object.

The `sym` command declares and optionally initializes a matrix object.

Syntax

`sym(n) sym_name[= assignment]`

`sym` takes an optional argument *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 `sym` command. The `sym` will be resized, if necessary. Once declared, symmetric matrices may be resized by repeating the `sym` command for a given matrix name.

Examples

```
sym mom
```

declares a symmetric matrix named MOM with one zero element.

```
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 627](#) of the *User’s Guide I* for a discussion of matrix objects in EViews.

See also [Matrix::matrix \(p. 258\)](#).

write	Sym Procs
--------------	---------------------------

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

`matrix_name.write(options) [path\filename]`

Follow the name of the matrix object by a period, the keyword, and the name for the output file. 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. The entire matrix will be exported.

Note that EViews cannot, at present, write into an existing file. The file that you select will, if it exists, be replaced.

Options

Options are specified in parentheses after the keyword and are used to specify the format of the output file.

File type

<code>t = dat, txt</code>	ASCII (plain text) files.
<code>t = wk1, wk3</code>	Lotus spreadsheet files.
<code>t = xls</code>	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

<code>na = string</code>	Specify text string for NAs. Default is “NA”.
<code>d = arg</code>	Specify delimiter (<i>default</i> is tab): “s” (space), “c” (comma).
<code>t</code>	Write by column (transpose the data). Default is to write by row.

Spreadsheet (Lotus, Excel) files

<code>letter_number</code>	Coordinate of the upper-left cell containing data.
<code>t</code>	Write by column (transpose the data). Default is to write by row.

Examples

```
m1.write(t=txt,na=.) a:\dat1.csv
```

writes the symmetric matrix M1 into an ASCII file named DAT1.CSV on the A: drive. NAs are coded as “.” (dot).

```
m1.write(t=txt,na=.) dat1.csv
```

writes the same file in the default directory.

```
m1.write(t=xls) "\\network\drive a\results"
```

saves the contents of M1 in an Excel file “Results.xls” in the specified directory.

Cross-references

See [“Exporting to a Spreadsheet or Text File” on page 105](#) of the *User's Guide I* for a discussion.

See also [pagesave \(p. 752\)](#) and [Sym::read \(p. 466\)](#).

System

System of equations for estimation.

System Declaration

systemdeclare system object (p. 503).

Declare a system object by entering the keyword `system`, followed by a name:

```
system mysys
```

To fill a system, open the system and edit the specification view, or use `append`. Note that systems are not used for simulation. See “[Model](#)” (p. 271).

System Methods

3slsthree-stage least squares (p. 477).

archestimate generalized autoregressive conditional heteroskedasticity (GARCH) models (p. 479).

fimlfull information maximum likelihood (p. 487).

gmmgeneralized method of moments (p. 489).

lsordinary least squares (p. 493).

surseemingly unrelated regression (p. 501).

tslstwo-stage least squares (p. 503).

wlsweighted least squares (p. 506).

wtlsweighted two-stage least squares (p. 507).

System Views

ellipseConfidence ellipses for coefficient restrictions (p. 483).

coefcovcoefficient covariance matrix (p. 484).

derivsderivatives of the system equations (p. 485).

endogtable or graph of endogenous variables (p. 486).

garchconditional variance/covariance of (G)ARCH estimation (p. 488).

gradsexamine the gradients of the objective function (p. 490).

jberamultivariate residual normality test (p. 491).

labellabel information for the system object (p. 492).

outputtable of estimation results (p. 498).

qstatsmultivariate residual autocorrelation Portmanteau tests (p. 498).

residcorresidual correlation matrix (p. 499).

residcovresidual covariance matrix (p. 500).

residsresidual graphs (p. 500).

resultstable of estimation results (p. 501).

spectext representation of system specification (p. 501).

wald..... Wald coefficient restriction test (p. 505).

System Procs

append..... add a line of text to the system specification (p. 479).

displayname..... set display name (p. 486).

makeendog..... make group of endogenous series (p. 494).

makegarch..... generate conditional variance series (p. 495).

makeloglike..... create and save log likelihood contribution from system (ARCH estimation) (p. 496).

makemodel..... create a model from the estimated system (p. 496).

makeresids..... make series containing residuals from system (p. 497).

updatecoefs..... update coefficient vector(s) from system (p. 505).

System Data Members

Scalar Values (individual equation data)

@coefcov(i, j)..... covariance of coefficients *i* and *j*.

@coefs(i)..... coefficient *i*.

@dw(k)..... Durbin-Watson statistic 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*.

@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.

@residcov (sym) covariance matrix of the residuals.

@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 increx saveres
```

To test coefficients using a Wald test:

```
sys1.wald c(1)=c(4)
```

To save the coefficient covariance matrix:

```
sym covs=sys1.@coefcov
```

System Entries

The following section provides an alphabetical listing of the commands associated with the “[System](#)” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

3sls	System Methods
-------------	--------------------------------

Estimate a system of equations by three-stage least squares.

Syntax

```
system_name.3sls(options)
```

Options

i	Iterate simultaneously over the weighting matrix and coefficient vector.
s	Iterate sequentially over the weighting matrix and coefficient vector.
o (<i>default</i>)	Iterate the coefficient vector to convergence following one-iteration of the weighting matrix.
c	One step (iteration) of the coefficient vector following one-iteration of the weighting matrix.
m = <i>integer</i>	Maximum number of iterations.
c = <i>number</i>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
l = <i>number</i>	Set maximum number of iterations on the first-stage coefficient estimation 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 = <i>keyword</i>	Set derivative methods. The argument <i>keyword</i> 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

```
sys1.3sls(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

See [Chapter 33. “System Estimation,”](#) on page 307 of the *User’s Guide II* for discussion of system estimation.

append	System Procs
---------------	------------------------------

Append a specification line to a system.

Syntax

`system_name.append text`

Type the text to be added after the `append` keyword.

Examples

```
system macrol
macrol.append cons=c(1)+c(2)*gdp+c(3)*cons(-1)
macrol.append inv=c(4)+c(5)*tb3+c(6)*d(gdp)
macrol.append gdp=cons+inv+gov
macrol.append inst tb3 gov cons(-1) gdp(-1)
macrol.gmm
show macrol.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.

Cross-references

For details, see [“System Estimation Methods” on page 308](#).

arch	System Methods
-------------	--------------------------------

Estimate generalized autoregressive conditional heteroskedasticity (GARCH) models.

Syntax

For a Diagonal Vech model:

`system_name.arch(options) @diagvech c(arg) [arch(n, arg)] [tarch(n, arg)] [garch(n, arg)] [exog(series, arg)]`

Indicate a Diagonal Vech model by using the `@diagvech` keyword. Follow the keyword with the constant term, `c`, and other optional terms to include in the variance equation: `arch`, `garch`, `tarch`, or `exog` (exogenous variable).

n indicates the order of the term, and *arg* indicates the type of coefficient for the term. For the exogenous variable, *series* indicates a series name.

Diagonal VECM Argument Options

c (<i>arg</i>)	where <i>arg</i> may be “scalar”, “diag” (diagonal), “rank1” (rank one), “fullrank”, “indef” (indefinite - default), or “vt” (variance target).
arch (<i>n</i> , <i>arg</i>)	where <i>n</i> indicates the order of the term, and <i>arg</i> may be “scalar”, “diag” (diagonal), “rank1” (rank one), “fullrank”, or “indef” (indefinite - default).
garch (<i>n</i> , <i>arg</i>)	where <i>n</i> indicates the order of the term, and <i>arg</i> may be “scalar”, “diag” (diagonal), “rank1” (rank one), “fullrank”, or “indef” (indefinite - default).
tarch (<i>n</i> , <i>arg</i>)	where <i>n</i> indicates the order of the term, and <i>arg</i> may be “scalar”, “diag” (diagonal), “rank1” (rank one), “fullrank”, or “indef” (indefinite - default).
exog (<i>series</i> , <i>arg</i>)	where <i>series</i> indicates a series name, and <i>arg</i> may be “scalar”, “diag” (diagonal), “rank1” (rank one), “fullrank”, or “indef” (indefinite - default).

For example, “c(indef)” instructs EViews to use an indefinite matrix for the constant term, while “ARCH(1, fullrank)” includes a first order ARCH with a full rank matrix coefficient type.

For a Conditional Constant Correlation model:

```
system_name.arch(options) @ccc c(arg) [arch(n[, arg])] [tarch(n[, arg])] [garch(n[,  
arg])] [exog(series, arg)]
```

Indicate a Conditional Constant Correlation model by using the @ccc keyword. Follow the keyword with the constant term, c, and other optional terms to include in the variance equation: arch, garch, tarch, or exog (exogenous variable).

n indicates the order of the term, and *arg* indicates the type of coefficient for the term. For the exogenous variable, *series* indicates a series name.

Conditional Constant Correlation Argument Options

c (<i>arg</i>)	where <i>arg</i> may be “scalar” (default) or “vt” (variance target).
arch (<i>n</i> [, <i>arg</i>])	where <i>n</i> indicates the order of the term, and the optional <i>arg</i> may be “scalar” (default).

garch (<i>n</i> [, <i>arg</i>])	where <i>n</i> indicates the order of the term, and the optional <i>arg</i> may be “scalar” (default).
tarch (<i>n</i> [, <i>arg</i>])	where <i>n</i> indicates the order of the term, and the optional <i>arg</i> may be “scalar” (default).
exog (<i>series</i> , <i>arg</i>)	where <i>series</i> indicates a series name, and <i>arg</i> may be “indiv” (individual - default) or “common”.

For a Diagonal BEKK model:

```
system_name.arch(options) @diagbekk c(arg) [arch(n[, arg])] [tarch(n[, arg])]
[garch(n[, arg])] [exog(series, arg)]
```

Indicate a Diagonal BEKK model by using the **@diagbekk** keyword. Follow the keyword with the constant term, **c**, and other optional terms to include in the variance equation: **arch**, **garch**, **tarch**, or **exog** (exogenous variable).

n indicates the order of the term, and *arg* indicates the type of coefficient for the term. For the exogenous variable, *series* indicates a series name.

Diagonal BEKK Argument Options

c (<i>arg</i>)	where <i>arg</i> may be “scalar”, “diag” (diagonal), “rank1” (rank one), “fullrank”, “indef” (indefinite - default), or “vt” (variance target).
arch (<i>n</i> [, <i>arg</i>])	where <i>n</i> indicates the order of the term, and the optional <i>arg</i> may be “diag” (diagonal - default).
garch (<i>n</i> [, <i>arg</i>])	where <i>n</i> indicates the order of the term, and the optional <i>arg</i> may be “diag” (diagonal - default).
tarch (<i>n</i> [, <i>arg</i>])	where <i>n</i> indicates the order of the term, and the optional <i>arg</i> may be “diag” (diagonal - default).
exog (<i>series</i> , <i>arg</i>)	where <i>series</i> indicates a series name, and <i>arg</i> may be “scalar”, “diag” (diagonal), “rank1” (rank one), “fullrank”, or “indef” (indefinite - default).

Options

General Options

tdist	Estimate the model assuming that the residuals follow a conditional Student’s <i>t</i> -distribution (the default is the conditional normal distribution).
h	Bollerslev-Wooldridge robust standard errors.
b	Use Berndt-Hall-Hall-Hausman (BHHH) as maximization algorithm. The default is Marquardt.

<code>m = integer</code>	Set maximum number of iterations.
<code>c = scalar</code>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.
<code>s</code>	Use the current coefficient values in “C” as starting values (see also param (p. 761)).
<code>showopts / -showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
<code>deriv = keyword</code>	Set derivative method. The argument <i>keyword</i> should be a one or two-letter string. The first letter should be either “f” (fast numeric derivatives) or “a” (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.
<code>coef = arg</code>	Specify the name of the coefficient vector of a system’s variance component; the default behavior is to use the “C” coefficient vector.
<code>backcast = n</code>	Backcast weight to calculate value used as the presample conditional variance. Weight needs to be greater than 0 and less than or equal to 1; the default value is 0.7. Note that a weight of 1 is equivalent to no backcasting, i.e. using the unconditional residual variance as the presample conditional variance.
<code>p</code>	Print estimation results.

Examples

```
system sys01
sys01.append dlog(jy)=c(1)
sys01.append dlog(bp)=c(2)
sys01.arch @diagvech c(indef) arch(1,indef) garch(1,rank1)
```

creates a system SYS01, appends two equations, and estimates the system using maximum likelihood with ARCH. A Diagonal VECH model is used with the constant and order 1 ARCH coefficient matrix indefinite and order 1 GARCH coefficient rank 1 matrix.

```
sys01.arch @diagbekk c(fullrank) arch(1) garch(1)
```

estimates SYS01 using a Diagonal BEKK model of order (1,1), with constant coefficient a full rank matrix.

```
sys01.arch(backcast=1) @ccc c arch(1) garch(1) exog(x1,indiv)
exog(x2,common)
```

estimates a CCC model, with each variance equation GARCH(1,1) and two exogenous variables X1 and X2. The influence of X1 on each variance equation can be varying, while X2's coefficient is the same across all variance equations. Presample uses the unconditional variance since the backcast parameter is set to one.

Cross-references

See [Chapter 29. “ARCH and GARCH Estimation,”](#) on page 185 of the *User's Guide II* for a discussion of ARCH models. See also [System::makegarch](#) (p. 495) and [Equation::arch](#) (p. 31).

cellipse	System Views
----------	------------------------------

Confidence ellipses for coefficient restrictions.

The `cellipse` view displays confidence ellipses for pairs of coefficient restrictions for an estimation object.

Syntax

`system_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

<code>ind = arg</code>	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.
<code>size = number</code> (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.
<code>dist = arg</code>	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 χ^2 distribution.
<code>p</code>	Print the graph.

Examples

The two commands:

```
sys1.ellipse c(1), c(2), c(3)
sys1.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).

```
sys1.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 [“Confidence Ellipses” on page 142](#) of the *User’s Guide II* for discussion.

See also [System::wald](#) (p. 505).

coefcov	System Views
----------------	------------------------------

Coefficient covariance matrix.

Displays the covariances of the coefficient estimates for an estimated system.

Syntax

```
system_name.coefcov(options)
```

Options

p	Print the coefficient covariance matrix.
---	--

Examples

```
sys1.coefcov
```

displays the coefficient covariance matrix for system SYS1 in a window. To store the coefficient covariance matrix as a sym object, use “@coefcov”:

```
sym eqcov = sys1.@coefcov
```

Cross-references

See also [Coef::coef](#) (p. 16) and [System::spec](#) (p. 501).

derivs	System Views
--------	------------------------------

Examine derivatives of the system 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 of the view shows this information in a multiple line graph.

Syntax

`system_name.derivs(options)`

Options

g	Display multiple graph showing the derivatives of the equation specification with respect to the coefficients, evaluated at each observation.
t	Display spreadsheet view of the values of the derivatives with respect to the coefficients evaluated at each observation.
p	Print results.

Note that the “g” and “t” options may not be used at the same time.

Examples

To show a table view of the derivatives:

```
sys1.derivs(t)
```

To display and print the summary view:

```
sys1.derivs(p)
```

Cross-references

See [“Derivative Computation Options” on page 628](#) of the *User’s Guide II* for details on the computation of derivatives.

See also [Equation::makederivs \(p. 65\)](#) for additional routines for examining derivatives, and [System::grads \(p. 490\)](#), and [Equation::makegrads \(p. 67\)](#) for corresponding routines for gradients.

displayname	System Procs
-------------	------------------------------

Display name for system objects.

Attaches a display name to a system object which may be used to label output in place of the standard system object name.

Syntax

`system_name.displayname display_name`

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in system object names.

Examples

```
hrs.displayname Hours Worked
hrs.label
```

The first line attaches a display name “Hours Worked” to the system object HRS, and the second line displays the label view of HRS, including its display name.

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels and display names.

See also [System::label \(p. 492\)](#).

endog	System Views
-------	------------------------------

Displays a spreadsheet or graph view of the endogenous variables.

Syntax

`system_name.endog(options)`

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 [System::makeendog](#) (p. 494), [System::system](#) (p. 503).

fiml	System Methods
------	--------------------------------

Estimation by full information maximum likelihood.

`fiml` estimates a system of equations by full information maximum likelihood (assuming a multivariate normal distribution).

Syntax

`system_name.fiml(options)`

Options

<code>i</code>	Iterate simultaneously over the covariance matrix and coefficient vector.
<code>s</code> (<i>default</i>)	Iterate sequentially over the covariance matrix and coefficient vector.
<code>m = integer</code>	Maximum number of iterations.
<code>c = number</code>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
<code>b</code>	Use Berndt-Hall-Hall-Hausman (BHHH) algorithm. Default method is Marquardt.
<code>showopts / -showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
<code>deriv = keyword</code>	Set derivative methods. The argument <i>keyword</i> 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.
<code>p</code>	Print estimation results.

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

See [Chapter 33. “System Estimation,” on page 307](#) of the *User’s Guide II* for a discussion of systems in EViews.

garch	System Views
-------	------------------------------

Conditional variance/covariance of (G)ARCH estimation.

Displays the conditional variance, covariance or correlation of a system estimated by ARCH.

Syntax

```
system_name.garch(options) [arg1, arg2, ...]
```

The optional arguments following the keyword indicate which endogenous variable to include. If no argument is provided, all variables in the system will be included.

Options

cor	Display correlation.
cov (default)	Display covariance.
var	Display only variance.
sd	Display only standard deviation.
graph (default)	Display data in graph.
mat	Display data in matrix format.
list	Display data in list format.
smpl = arg	Date to return conditional covariance value.
pre	Include presample data (used with the mat option only).
p	Print the graph

Examples

```
sys1.garch(cor)
```

displays the conditional correlation graph of SYS1.

Cross-references

ARCH estimation is described in [Chapter 29. “ARCH and GARCH Estimation,” on page 185](#) of the *User’s Guide II*.

gmm	System Methods
------------	--------------------------------

Estimation by generalized method of moments (GMM).

The system object must be specified with a list of instruments.

Syntax

`system_name.gmm(options)`

Options

<code>m = integer</code>	Maximum number of iterations.
<code>c = number</code>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
<code>l = number</code>	Set maximum number of iterations on the first-stage iteration to get the one-step weighting matrix.
<code>showopts / -showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
<code>deriv = keyword</code>	Set derivative methods. The argument <i>keyword</i> 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.
<code>w</code>	Use White’s diagonal weighting matrix (for cross section data).
<code>b = arg</code> (<i>default</i> = “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).
<code>q</code>	Use the quadratic kernel. Default is to use the Bartlett kernel.
<code>n</code>	Prewhiten by a first order VAR before estimation.
<code>i</code>	Iterate simultaneously over the weighting matrix and the coefficient vector.
<code>s</code>	Iterate sequentially over the weighting matrix and coefficient vector.

o (<i>default</i>)	Iterate only on the coefficient vector with one step of the weighting matrix.
c	One step (iteration) of the coefficient vector following one step of the weighting matrix.
e	TSLS estimates with GMM standard errors.
p	Print results.

Note that some options are only available for a subset of specifications.

Examples

For system estimation, the command:

```
sys1.gmm (b=a, q, i)
```

estimates the system SYS1 by GMM with a quadratic kernel, Andrews automatic bandwidth selection, and iterates simulatenously over the weight and coefficient vectors until convergence.

Cross-references

See [Chapter 25. “Additional Regression Methods,” on page 23](#) and [Chapter 33. “System Estimation,” on page 307](#) for discussion of the various GMM estimation techniques.

grads	System Views
-------	------------------------------

Gradients of the objective function.

Displays the gradients of the objective function (where available) for an estimated system object.

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. Evaluating the gradients at current coefficient values allows you to examine 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

```
system_name.grads(options)
```

Options

p	Print results.
---	----------------

Examples

To show a summary view of the gradients:

```
sys1.grads
```

To print the table view:

```
sys1.grads(p)
```

Cross-references

See also [System::derivs \(p. 485\)](#).

jbera	System Views
-------	------------------------------

Multivariate residual normality test.

Syntax

```
system_name.jbera(options)
```

You must specify a factorization method using 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 Var::svar (p. 568) .
name = arg	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 first column contains the third moments, the second column contains the χ^2_1 statistics for the third moments, the third column contains the fourth moments, and the fourth column holds the χ^2_1 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 last 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 χ^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 = 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 statistic is stored in the first column of the $(k + 1)$ row (which will be a missing value for all other options).

Examples

```
show sys1.jbera(factor=cor,name=jb)
```

carries out the residual multivariate normality test on SYS1 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 34. “Vector Autoregression and Error Correction Models,” on page 345](#) of the *User’s Guide II* for a discussion of the test and other VAR diagnostics.

label	System Views System Procs
-------	---

Display or change the label view of the system object, including the last modified date and display name (if any).

As a procedure, `label` changes the fields in the system object label.

Syntax

```
system_name.label
system_name.label(options) [text]
```

Options

The first version of the command displays the label view of the system. The second version may be used to modify the label. 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 <i>text</i> .
s	Sets the source field to <i>text</i> .

u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of system S1 with “Data from CPS 1988 March File”:

```
s1.label(r)
s1.label(r) Data from CPS 1988 March File
```

To append additional remarks to S1, and then to print the label view:

```
s1.label(r) Log of hourly wage
s1.label(p)
```

To clear and then set the units field, use:

```
s1.label(u) Millions of bushels
```

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels.

See also [System::displayname \(p. 486\)](#).

ls	System Methods
----	--------------------------------

Estimation by linear or nonlinear least squares regression.

Syntax

```
system_name.ls(options)
```

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. The criterion will be set to the nearest value between 1e-24 and 0.2.
s	Use the current coefficient values in “C” as starting values for equations with AR or MA terms (see also param (p. 761)).

<code>showopts /</code> <code>-showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
<code>deriv = keyword</code>	Set derivative methods. The argument <i>keyword</i> 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.
<code>w =</code> <code>series_name</code>	Weighted Least Squares. Each observation will be weighted by multiplying by the specified series.
<code>p</code>	Print basic estimation results.

Examples

```
sys1.ls(m=100)
```

estimates SYS1 using least squares, with the maximum number of iterations set at 100.

Cross-references

[Chapter 24. “Basic Regression,” on page 5](#) and [Chapter 25. “Additional Regression Methods,” on page 23](#) of the *User’s Guide II* discuss the various regression methods in greater depth.

See [Chapter 5. “Special Expression Reference,” on page 815](#), for special terms that may be used in `ls` specifications.

makeendog	System Procs
-----------	------------------------------

Make a group out of the endogenous series.

Syntax

```
system_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.

Examples

```
sys1.makeendog grp_v1
```

creates a group named GRP_V1 that contains the endogenous series in SYS1.

Cross-references

See also [System::endog \(p. 486\)](#) and [Model::makegroup \(p. 282\)](#).

makegarch	System Procs
-----------	--------------

Generate conditional variance series.

Saves the estimated conditional variance (from a system estimated using ARCH) as a named series. You may also save the conditional covariance or correlation.

Syntax

```
system_name.makegarch(options) [series1_name series2_name]
```

The optional series name arguments following the `makegarch` keyword indicate which endogenous variables to include. If no argument is given, all variables in the system will be included.

Options

<code>cor</code>	Generate conditional correlation.
<code>cov (default)</code>	Generate conditional variance and covariance.
<code>var</code>	Generate conditional variance.
<code>mat</code>	Output as a matrix (default is to output as a series).
<code>name = arg</code>	Base name or matrix name of the data to be saved.
<code>date = arg</code>	Date to return conditional covariance value (used only with the <code>mat</code> option).
<code>pre</code>	Include presample data (used only with the <code>mat</code> option).

Examples

```
sys01.makegarch
```

creates conditional variances and conditional covariance series using the default names GARCH_01, GARCH_02, etc. for the conditional variance and GARCH_01_02, GARCH_01_03, etc. for the conditional covariance.

```
sys01.makegarch(mat, cor, date=12/11/2000, name=cov_mat)
```

creates a matrix named COV_MAT that contains the conditional correlation for the date 12/11/2000.

Cross-references

See [Chapter 29. “ARCH and GARCH Estimation,”](#) on page 185 of the *User’s Guide II* for a discussion of GARCH models.

See also [System::arch \(p. 479\)](#), [System::arch \(p. 479\)](#), [Equation::archtest \(p. 34\)](#), and [System::garch \(p. 488\)](#).

makeloglike	System Procs
--------------------	------------------------------

Create and save log likelihood contribution from system (ARCH estimation).

Syntax

`system_name.makeloglike [ser1]`

After the keyword, provide an optional name to save the log likelihood contribution. If you do not provide a name, EViews will name the series using the next available name of the form “LOGLIKE##”. (If LOGLIKE01 already exists, it will be named LOGLIKE02, and so on.)

Examples

```
sys1.makeloglike log1
```

creates a series of log likelihood contribution for the system and saves it in the series LOG1.

Cross-references

makemodel	System Procs
------------------	------------------------------

Make a model from a system of equations.

Syntax

`system_name.makemodel(name) assign_statement`

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

```
sys3.makemodel(sysmod) @prefix s_
```

makes a model named SYSMOD from the estimated system. SYSMOD includes an assignment statement “ASSIGN @PREFIX S_”. Use the command “show sysmod” or “sysmod.spec” to open the SYSMOD window.

Cross-references

See [Chapter 36. “Models,” on page 407](#) of the *User’s Guide II* for a discussion of specifying and solving models in EViews.

See also [System::append \(p. 479\)](#), [Model::merge \(p. 283\)](#) and [Model::solve \(p. 287\)](#).

makesresids	System Procs
-------------	--------------

Create residual series.

Creates and saves residuals in the workfile from an estimated system object.

Syntax

```
system_name.makesresids(options) []
```

Follow the system name with a period and the `makesresids` keyword, then provide a list of names to be given to the stored residuals. 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.

Options

n = <i>arg</i>	Create group object to hold the residual series.
chol	Standardized residuals factorized using the inverse of Cholesky factor of the (conditional) covariance matrix (for system ARCH).
cor	Standardized residuals factorized using the inverse square root of the (conditional) correlation matrix (for system ARCH).
cov	Standardized residuals factorized using the inverse square root of the (conditional) covariance matrix (for system ARCH).
bn = <i>arg</i>	Base name used to generate the name of the residual series.

Examples

```
sys1.makesresids res_sys1
```

creates a set of series containing the residuals from the system using RES_SYS1 to name the first equation residual, and RESID01, RESID02, *etc.*, to name the remaining residuals.

Cross-references

See [“Weighted Least Squares” on page 32](#) of the *User’s Guide II* for a discussion of standardized residuals after weighted least squares and [Chapter 30. “Discrete and Limited Dependent Variable Models,” on page 209](#) of the *User’s Guide II* for a discussion of standardized and generalized residuals in binary, ordered, censored, and count models.

output	System Views
--------	------------------------------

Display estimation output.

`output` changes the default object view to display the estimation output (equivalent to using [System::results](#) (p. 501)).

Syntax

`system_name.output`

Options

<code>p</code>	Print estimation output for estimation object
----------------	---

Examples

The `output` keyword may be used to change the default view of an estimation object. Entering the command:

```
sys1.output
```

displays the estimation output for system SYS1.

Cross-references

See [System::results](#) (p. 501).

qstats	System Views
--------	------------------------------

Multivariate residual autocorrelation Portmanteau tests.

Syntax

`system_name.qstats(h, options)`

You must specify the highest order of lag *h* to test for serial correlation.

Options

<code>maxlag = arg</code>	Maximum lag in system specification (default = 0).
<code>chol</code>	Standardized residuals factorized using the inverse of Cholesky factor of the (conditional) covariance matrix (for system ARCH).

cor	Standardized residuals factorized using the inverse square root of the (conditional) correlation matrix (for system ARCH).
cov	Standardized residuals factorized using the inverse square root of the (conditional) covariance matrix (for system ARCH).
p	Print the Portmanteau test results.

Examples

```
show sys1.qstats(10)
```

displays the portmanteau tests for lags up to 10.

Cross-references

See [“Diagnostic Views” on page 348](#) of the *User’s Guide II* for a discussion of the Portmanteau tests and other VAR diagnostics.

See [Var::arlm \(p. 546\)](#) for a related multivariate residual serial correlation LM test.

residcor	System Views
----------	------------------------------

Residual correlation matrix.

Displays the correlations of the residuals from each equation in the system.

Syntax

```
system_name.residcor(options)
```

Options

p	Print the correlation matrix.
---	-------------------------------

Examples

```
sys1.residcor
```

displays the residual correlation matrix of SYS1.

Cross-references

See also [System::residcov \(p. 500\)](#) and [System::makesresids \(p. 497\)](#).

residcov	System Views
-----------------	------------------------------

Residual covariance matrix.

Displays the covariances of the residuals from each equation in the system.

Syntax

`system_name.residcov(options)`

Options

<code>p</code>	Print the covariance matrix.
----------------	------------------------------

Examples

```
sys1.residcov
```

displays the residual covariance matrix of SYS1.

Cross-references

See also [System::residcor](#) (p. 499) and [System::makeresids](#) (p. 497).

resids	System Views
---------------	------------------------------

Display residuals.

`resids` displays multiple graphs of the residuals. Each graph will contain the residuals for each equation in the system.

Syntax

`system_name.resids(options)`

Options

<code>g</code> (<i>default</i>)	Display graph(s) of residuals.
<code>p</code>	Print the table/graph.

Examples

```
sys1.resids(g)
```

displays a graph of the residual series in system SYS1.

Cross-references

See also [System::makeresids](#) (p. 497).

results	System Views
---------	------------------------------

Displays the results view of an estimated system.

Syntax

```
system_name.results(options)
```

Options

p	Print the view.
---	-----------------

Examples

```
sys1.results(p)
```

displays and prints the results of SYS1.

spec	System Views
------	------------------------------

Display the text specification view for system objects.

Syntax

```
system_name.spec(options)
```

Options

p	Print the specification text.
---	-------------------------------

Examples

```
sys1.spec
```

displays the specification of the system object SYS1.

Cross-references

See also [System::append](#) (p. 479).

sur	System Methods
-----	--------------------------------

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_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 (<i>default</i>)	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 = <i>integer</i>	Maximum number of iterations.
c = <i>number</i>	Set convergence criterion. The criterion will be set to the nearest value between 1e-24 and 0.2.
l = <i>number</i>	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 = <i>keyword</i>	Set derivative methods. The argument <i>keyword</i> 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(showopts,m=500)
```

estimates NLSYS by SUR with up to 500 iterations. The “showopts” option displays the starting values.

Cross-references

See [Chapter 33. “System Estimation,”](#) on page 307 of the *User’s Guide II* for a discussion of system estimation.

system	System Declaration
---------------	------------------------------------

Declare system of equations.

Syntax

```
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 msys
```

creates a system named MYSYS.

Cross-references

[Chapter 33. “System Estimation,” on page 307](#) of the *User's Guide II* provides a full discussion of system objects.

See [System::append \(p. 479\)](#) for adding specification lines to an existing system.

tsls	System Methods
-------------	--------------------------------

Two-stage least squares.

Syntax

```
system_name.tsls(options)
```

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.

Options

General options

<code>m = integer</code>	Set maximum number of iterations.
<code>c = number</code>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
<code>deriv = keyword</code>	Set derivative methods. The argument <i>keyword</i> 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.
<code>showopts / -showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
<code>i</code>	Iterate on the weighting matrix and coefficient vector simultaneously.
<code>s</code>	Iterate on the weighting matrix and coefficient vector sequentially.
<code>o (default)</code>	Iterate only on the coefficient vector with one step of the weighting matrix.
<code>c</code>	One step iteration of the coefficient vector after one step of the weighting matrix.
<code>l = number</code>	Set maximum number of iterations on the first-stage iteration to get one-step weighting matrix.
<code>p</code>	Print estimation results.

Examples

```
sys1.tsls
```

estimates the system object using TSLS.

Cross-references

See [“Two-stage Least Squares” on page 37](#) and [“Two-Stage Least Squares” on page 309](#) of the *User’s Guide II* for details on two-stage least squares estimation in single equations and systems, respectively. [“Instrumental Variables” on page 502](#) of the *User’s Guide II* discusses estimation using pool objects, while [“Instrumental Variables Estimation” on page 544](#) of the *User’s Guide II* discusses estimation in panel structured workfiles.

See also [System::ls](#) (p. 493). For estimation of weighted TSLS in systems, see [System::wtsls](#) (p. 507).

updatecoefs	System Procs
--------------------	------------------------------

Update coefficient object values from system object.

Copies coefficients from the system into the appropriate coefficient vector or vectors.

Syntax

```
system_name.updatecoefs
```

Follow the name of the system object by a period and the keyword `updatecoefs`.

Examples

```
SYS1.updatecoefs
```

places the coefficients from SYS1 in the coefficient vectors used in the system.

Cross-references

See also [Coef::coef](#) (p. 16).

wald	System Views
-------------	------------------------------

Wald coefficient restriction test.

The `wald` view carries out a Wald test of coefficient restrictions for a system object.

Syntax

```
system_name.wald restrictions
```

Enter the system name, followed by a period, and the keyword. You must provide a list of the coefficient restrictions, with joint (multiple) coefficient restrictions separated by commas.

Options

<code>p</code>	Print the test results.
----------------	-------------------------

Examples

```
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.

Cross-references

See [“Wald Test \(Coefficient Restrictions\)” on page 145](#) of the *User’s Guide II* for a discussion of Wald tests.

See also [System::cellipse \(p. 483\)](#), [testdrop \(p. 790\)](#), [testadd \(p. 789\)](#).

wls	System Methods
-----	--------------------------------

Estimates a system of equations using weighted least squares.

Syntax

system_name.wls(*options*)

Options

i	Iterate simultaneously over the weighting matrix and coefficient vector.
s	Iterate sequentially over the computation of the weighting matrix and the estimation of the coefficient vector.
o (<i>default</i>)	Iterate the estimate of the coefficient vector to convergence following one-iteration of the weighting matrix.
c	One step (iteration) of the coefficient vector estimates following one iteration of the weighting matrix.
m = <i>integer</i>	Maximum number of iterations.
c = <i>number</i>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
l = <i>number</i>	Set maximum number of iterations on the first-stage coefficient estimation to get one-step weighting matrix.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
deriv = <i>keyword</i>	Set derivative methods. The argument <i>keyword</i> 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 the estimation results.

Examples

```
sys1.wls
```

estimates the system of equations in SYS1 by weighted least squares.

Cross-references

See [Chapter 33. “System Estimation,” on page 307](#) of the *User’s Guide II* for a discussion of system estimation.

See also the available options for weighted least squares in [System::ls \(p. 493\)](#).

wtls	System Methods
------	--------------------------------

Perform weighted two-stage least squares estimation of a system of equations.

Syntax

```
system_name.wtls(options)
```

Options

i	Iterate simultaneously over the weighting matrix and coefficient vector.
s	Iterate sequentially over the computation of the weighting matrix and the estimation of the coefficient vector.
o (<i>default</i>)	Iterate the coefficient vector to convergence following one-iteration of the weighting matrix.
c	One step (iteration) of the coefficient vector following one iteration of the weighting matrix.
m = <i>integer</i>	Maximum number of iterations.
c = <i>number</i>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
l = <i>number</i>	Set maximum number of iterations on the first-stage iteration to get the one-step weighting matrix.

<code>showopts /</code> <code>-showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
<code>deriv = keyword</code>	Set derivative methods. The argument <i>keyword</i> 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.
<code>p</code>	Print estimation results.

Examples

```
sys1.wtols
```

estimates the system of equations in SYS1 by weighted two-stage least squares.

Cross-references

See [“Weighted Two-Stage Least Squares” on page 309](#) of the *User’s Guide II* for further discussion.

See also `System::tols` (p. 503) for both unweighted and weighted single equation 2SLS.

Table

Table object. Formatted two-dimensional table for output display.

Table Declaration

freeze freeze tabular view of object (p. 724).
table create table object (p. 530).

To declare a table object, use the keyword `table`, followed by an optional row and column dimension, and then the object name:

```
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:

```
table newtable=outtable
```

Lastly, you may use the `freeze` command to create tables from tabular views of other objects:

```
freeze(newtab) ser1.freq
```

Table Views

label label information for the table object (p. 514).
sheet view the table (p. 529).

Table Procs

comment adds or removes a comment in a table cell (p. 510).
deletecol Remove columns from a table (p. 511).
deleterow Remove rows from a table (p. 512).
displayname set display name (p. 512).
insertcol insert additional columns into a table (p. 513).
insertrow insert additional rows into a table (p. 513).
save save table as CSV, tab-delimited ASCII text, RTF, or HTML file on disk (p. 515).
setfillcolor set the fill (background) color of a set of table cells (p. 516).
setfont set the font for the text in a set of table cells (p. 518).
setformat set the display format of a set of table cells (p. 519).
setheight set the row height in a set of table cells (p. 523).
setindent set the indentation for a set of table cells (p. 524).
setjust set the justification for a set of table cells (p. 524).

- [setlines](#)..... set the line characteristics and borders for a set of table cells (p. 525).
- [setmerge](#)..... merge or unmerge a set of table cells (p. 526).
- [settextcolor](#)..... set the text color in a set of table cells (p. 528).
- [setwidth](#)..... set the column width for a set of table cells (p. 529).
- [table](#)..... assign or change the title of a table (p. 530).

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. 774).
- [setcolwidth](#)..... set width of a table column (p. 776).
- [setline](#) place a horizontal line in table (p. 776).

Table Examples

```
table(5,5) mytable
%strval = mytable(2,3)
mytable(4,4) = "R2"
mytable(4,5) = @str(eql.@r2)
```

Table Entries

The following section provides an alphabetical listing of the commands associated with the “Table” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

comment	Table Procs
---------	-----------------------------

Adds or removes a comment in a table cell.

Syntax

```
table_name.comment(cell_arg) [comment_arg]
```

where *cell_arg*, which identifies the cell, can take one of the following forms:

<i>cell</i>	Cell identifier. You can reference 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 <i>number</i> (e.g., “R1C2”).
<i>row[, col]</i>	Row number, followed by column letter or number (e.g., “2,C”, or “2,3”), separated by “,”.

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:

```
tab1.comment(d2) "hello world"
tab1.comment(r2c4) "hello world"
tab1.comment(2,d) "hello world"
tab1.comment(2,4) "hello world"
```

To remove a comment, simply omit the *comment_arg*:

```
tab1.comment(d2)
```

clears the comment (if present) from the second row, fourth column.

Cross-references

For additional discussion of tables see [Chapter 20. “Working with Tables”](#) in the *User’s Guide I*. See also [Table::setlines \(p. 525\)](#) and [Table::setmerge \(p. 526\)](#).

deletecol	Table Procs
-----------	-----------------------------

Removes columns from a table.

Syntax

```
table_name.deletecol(col_loc) [num_cols]
```

where *col_loc* specifies the first column to be removed. The *col_loc* may either be the integer column number (e.g. “3”) or the column letter (e.g. “C”).

The *num_cols* specifies the number of columns to remove from the table. If *num_cols* is not provided, the default is one.

Examples

```
tab1.removecol(d) 2
```

removes two columns beginning at the “d” or fourth column.

Cross-references

For additional discussion of tables see [Chapter 20. “Working with Tables”](#) in the *User’s Guide I*. See also [Table::deleterow \(p. 512\)](#), [Table::insertcol \(p. 513\)](#) and [Table::insertrow \(p. 513\)](#).

deleterow	Table Procs
-----------	-----------------------------

Removes rows from a table.

Syntax

```
table_name.deleterow(row_loc) [num_rows]
```

where *row_loc* is an integer which specifies the first row to remove, and *num_rows* specifies the number of rows to remove from the table. If *num_rows* is not provided, the default is one.

Examples

```
tab1.deleterow(2) 5
```

removes five rows beginning with the second row.

Cross-references

For additional discussion of tables see [Chapter 20. “Working with Tables”](#) in the *User’s Guide I*. See also [Table::deletecol \(p. 511\)](#), [Table::insertcol \(p. 513\)](#) and [Table::insertrow \(p. 513\)](#).

displayname	Table Procs
-------------	-----------------------------

Display name for table objects.

Attaches a display name to a table object which may be used to label output in place of the standard table object name.

Syntax

```
table_name.displayname display_name
```

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in table object names.

Examples

```
hrs.displayname Hours Worked
hrs.label
```

The first line attaches a display name “Hours Worked” to the table object HRS, and the second line displays the label view of HRS, including its display name.

Cross-references

See “Labeling Objects” on page 72 of the *User’s Guide I* for a discussion of labels and display names.

See also `Table::label` (p. 514).

insertcol	Table Procs
-----------	-----------------------------

Insert additional columns in a table.

Syntax

```
table_name.insertcol(col_loc) [num_cols]
```

where *col_loc* specifies the column location to insert the new columns. The *col_loc* may either be the integer column number (e.g. “3”) or the column letter (e.g. “C”).

The *num_cols* specifies the number of columns to insert into the table. If *num_cols* is not provided, the default is one.

Examples

```
tbl.insertcol(d) 2
```

inserts two new columns beginning at the “d” or fourth column.

Cross-references

For additional discussion of tables see [Chapter 20. “Working with Tables”](#) in the *User’s Guide I*. See also `Table::insertrow` (p. 513), `Table::deletecol` (p. 511) and `Table::deleterow` (p. 512).

insertrow	Table Procs
-----------	-----------------------------

Insert additional rows in a table.

Syntax

```
table_name.insertrow(row_loc) [num_rows]
```

where *row_loc* is an integer which specifies the row location to insert the new rows, and *num_rows* specifies the number of rows to insert. If *num_rows* is not provided, the default is one.

Examples

```
tbl.insertrow(2) 5
```

inserts five new rows beginning at the second row.

Cross-references

For additional discussion of tables see [Chapter 20. “Working with Tables”](#) in the *User’s Guide I*. See also [Table::insertcol](#) (p. 513), [Table::deletecol](#) (p. 511) and [Table::deleterow](#) (p. 512).

label	Table Views Table Procs
-------	---

Display or change the label view of the table object, including the last modified date and display name (if any).

As a procedure, `label` changes the fields in the table label.

Syntax

```
table_name.label
table_name.label(options) [text]
```

Options

The first version of the command displays the label view of the table. The second version may be used to modify the label. 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 <i>text</i> .
s	Sets the source field to <i>text</i> .
u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of the table TAB1 with “Data from CPS 1988 March File”:

```
tab1.label(r)
tab1.label(r) Data from CPS 1988 March File
```

To append additional remarks to TAB1, and then to print the label view:

```
tab1.label(r) Log of hourly wage
tab1.label(p)
```

To clear and then set the units field, use:

```
tbl1$label(u) Millions of bushels
```

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels.

See also [Table::displayname \(p. 512\)](#).

save	Table Procs
------	-----------------------------

Save table to disk as a CSV, tab-delimited ASCII text, RTF, or HTML file.

Syntax

```
table_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.

If an explicit path is not specified, the file will be stored in the default directory, as set in the **File Locations** global options.

Options

t = <i>file_type</i> (default = “csv”)	Specifies the file type, where <i>file_type</i> may be one of: “csv” (CSV - comma-separated), “rtf” (Rich-text format), “txt” (tab-delimited text), or “html” (HTML - Hypertext Markup Language). Files will be saved with the “.csv”, “.rtf”, “.txt”, and “.htm” extensions, respectively.
s = <i>arg</i>	Scale size, where <i>arg</i> is from 0.05 to 2, representing the fraction of the original table size (only valid for HTML or RTF files).
r = <i>cell_range</i>	Range of table cells to be saved. See Table::setfill-color (p. 516) for the <i>cell_range</i> syntax. If a range is not provided, the entire table will be saved.
n = <i>string</i>	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

The command:

```
tbl.save mytable
```

saves TAB1 to a CSV file named “MYTABLE.CSV” in the default directory.

```
tbl.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”.

```
tbl.save(r=B2:C10, t=html, s=.5) mytable
```

saves from data from the second row, second column, to the tenth row, third column of TAB1 to a HTML file named MYTABLE.HTM at half of the original size.

```
tbl.save(f, n=".", r=B) mytable
```

saves the data in the second column in full precision to a CSV file named “MYTABLE.CSV”, and writes all NA values as “.”.

Cross-references

See [Chapter 15. “Graphs, Tables, Text, and Spools,” beginning on page 523](#) of the *User’s Guide I* for a discussion of graphs and tables.

setfillcolor	Table Procs
--------------	-----------------------------

Set the fill (background) color of the specified table cells.

Syntax

```
table_name.setfillcolor(cell_range) color_arg
```

where *cell_range* can take one of the following forms:

@all	Apply to all cells in the table.
cell	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 <i>number</i> (e.g., “R1C2”). Apply to cell.
row[,] col	Row number, followed by column letter or number (e.g., “2,C”, or “2,3”), separated by “,”. Apply to cell.
row	Row number (e.g., “2”). Apply to all cells in the row.

<i>col</i>	Column <i>letter</i> (e.g., “B”). Apply to all cells in the column.
<i>first_cell[:]</i> <i>last_cell</i> , <i>first_cell</i> [,] <i>last_cell</i>	Top left cell of the selection range (specified in “ <i>cell</i> ” format), followed by bottom right cell of the selection range (specified in “ <i>cell</i> ” format), separated by a “:” 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.
<i>first_cell_row</i> [,] <i>first_cell_col</i> [,] <i>last_cell_row</i> [,] <i>last_cell_col</i>	Top left cell of the selection range (specified in “ <i>row</i> [,] <i>col</i> ” format), followed by bottom right cell of the selection range (specified in “ <i>row</i> [,] <i>col</i> ” 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.

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 use custom colors.

The predefined colors are given by the keywords (with their RGB equivalents):

blue	@rgb(0, 0, 255)
red	@rgb(255, 0, 0)
green	@rgb(0, 128, 0)
black	@rgb(0, 0, 0)
white	@rgb(255, 255, 255)
purple	@rgb(128, 0, 128)
orange	@rgb(255, 128, 0)
yellow	@rgb(255, 255, 0)
gray	@rgb(128, 128, 128)
ltgray	@rgb(192, 192, 192)

Examples

To set a purple background color for the cell in the second row and third column of TAB1, you may use any of the following:

```
tab1.setfillcolor(C2) @rgb(128, 0, 128)
tab1.setfillcolor(2,C) @RGB(128, 0, 128)
tab1.setfillcolor(2,3) purple
tab1.setfillcolor(r2c3) purple
```

You may also specify a yellow color for the background of an entire column, or an entire row,

```
tbl.setfillcolor(C) @RGB(255, 255, 0)
tbl.setfillcolor(2) yellow
```

or for the background of the cells in a rectangular region:

```
tbl.setfillcolor(R2C3:R3C6) ltgray
tbl.setfillcolor(r2c3,r3c6) ltgray
tbl.setfillcolor(2,C,3,F) @rgb(192, 192, 192)
tbl.setfillcolor(2,3,3,6) @rgb(192, 192, 192)
```

Cross-references

For additional discussion of tables see [Chapter 20. “Working with Tables,” on page 675](#) of the *User’s Guide I*.

See [Table::settextcolor \(p. 528\)](#) and [Table::setFont \(p. 518\)](#) for details on changing text color and font, and [Table::setlines \(p. 525\)](#) for drawing lines between around and through cells. See [Table::setformat \(p. 519\)](#) for setting cell formats.

setFont	Table Procs
---------	-----------------------------

Set the font for text in the specified table cells.

Syntax

```
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 [Table::setfillcolor \(p. 516\)](#) for the syntax for *cell_range*.

The *font_args* may include one or more of the following:

<i>face_name</i>	A string that specifies the typeface name of the font, enclosed in double quotes.
<i>integer</i> [pt]	Integer font size, in points, followed by the “pt” identifier (e.g., “12pt”).
+ b / -b	[Add / remove] boldface.
+ i / -i	[Add / remove] italics.
+ s / -s	[Add / remove] strikethrough.
+ u / -u	[Add / remove] underscore.

Examples

```
tbl.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).

```
tbl.setfont(3,B,10,D) 8pt
```

changes all of text in the region to 8 point.

```
tbl.setfont(4,B) +b -i
```

removes the italic, and adds boldface to the B4 cell (row 4, column 2).

The commands:

```
tbl.setfont(b) -s +u 14pt
```

```
tbl.setfont(2) "Batang" 14pt +u
```

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 20. “Working with Tables”](#) of the *User’s Guide I*.

See [Table::settextcolor \(p. 528\)](#) and [Table::setfillcolor \(p. 516\)](#) for details on changing the text and cell background colors. See [Table::setformat \(p. 519\)](#) for setting the cell formats.

setformat	Table Procs
-----------	-----------------------------

Set the display format for cells in a table view.

Syntax

```
table_name.setformat(cell_range) 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 cells to be modified. It may take one of the following forms:

<code>@all</code>	Apply to all cells in the table.
<code>cell</code>	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 <i>number</i> (e.g., “R1C2”). Apply to cell.
<code>row[, col]</code>	Row number, followed by column letter or number (e.g., “2,C”, or “2,3”), separated by “,”. Apply to cell.
<code>row</code>	Row number (e.g., “2”). Apply to all cells in the row.
<code>col</code>	Column <i>letter</i> (e.g., “B”). Apply to all cells in the column.
<code>first_cell[:last_cell,</code> <code>first_cell[,last_cell]</code>	Top left cell of the selection range (specified in “ <code>cell</code> ” format), followed by bottom right cell of the selection range (specified in “ <code>cell</code> ” format), separated by a “:” 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.
<code>first_cell_row[,]</code> <code>first_cell_col[,]</code> <code>last_cell_row[,]</code> <code>last_cell_col</code>	Top left cell of the selection range (specified in “ <code>row[, col]</code> ” format), followed by bottom right cell of the selection range (specified in “ <code>row[, col]</code> ” 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.

To format numeric values, you should use one of the following format specifications:

<code>g[.precision]</code>	significant digits
<code>f[.precision]</code>	fixed decimal places
<code>c[.precision]</code>	fixed characters
<code>e[.precision]</code>	scientific/float
<code>p[.precision]</code>	percentage
<code>r[.precision]</code>	fraction

In order to set a format that groups digits into thousands, place a “t” after a specified format. For example, to obtain a fixed number of decimal places, with commas used to separate thousands, use “`ft[.precision]`”. To obtain a fixed number of characters with a period used to separate thousands, use “`ct[.precision]`”.

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 [“Date Formats” on page 707](#) of the *User’s Guide I*). The possible for-

mat 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 given by:

WF	(uses current EViews workfile date display format)
YYYY	“2002”
YYYY-Mon	“2002-Jan”
YYYYMon	“2002 Jan”
YYYY[M]MM	“2002[M]01”
YYYY:MM	“2002:01”
YYYY[Q]Q	“2002[Q]1”
YYYY:Q	“2002:Q
YYYY[S]S	“2002[S]1” (semi-annual)
YYYY:S	“2002:1”
YYYY-MM-DD	“2002-01-07”
YYYY Mon dd	“2002 Jan 7”
YYYY Month dd	“2002 January 7”
YYYY-MM-DD HH:MI	“2002-01-07 22:40”
YYYY-MM-DD HH:MI:SS	“2002-01-07 22:40:30”
YYYY-MM-DD HH:MI:SS.SSS	“2002-01-07 22:40:30.125”
Mon-YYYY	“Jan-2002”
Mon dd YYYY	“Jan 7 2002”
Mon dd, YYYY	“Jan 7, 2002”
Month dd YYYY	“January 7 2002”
Month dd, YYYY	“January 7, 2002”
MM/DD/YYYY	“01/07/2002”
mm/DD/YYYY	“1/07/2002”
mm/DD/YYYY HH:MI	“1/07/2002 22:40”
mm/DD/YYYY HH:MI:SS	“1/07/2002 22:40:30”
mm/DD/YYYY HH:MI:SS.SSS	“1/07/2002 22:40:30.125”
mm/dd/YYYY	“1/7/2002”
mm/dd/YYYY HH:MI	“1/7/2002 22:40”
mm/dd/YYYY HH:MI:SS	“1/7/2002 22:40:30”
mm/dd/YYYY HH:MI:SS.SSS	“1/7/2002 22:40:30.125”
dd/MM/YYYY	“7/01/2002”
dd/mm/YYYY	“7/1/2002”

DD/MM/YYYY	"07/01/2002"
dd Mon YYYY	"7 Jan 2002"
dd Mon, YYYY	"7 Jan, 2002"
dd Month YYYY	"7 January 2002"
dd Month, YYYY	"7 January, 2002"
dd/MM/YYYY HH:MI	"7/01/2002 22:40"
dd/MM/YYYY HH:MI:SS	"7/01/2002 22:40:30"
dd/MM/YYYY HH:MI:SS.SSS	"7/01/2002 22:40:30.125"
dd/mm/YYYY hh:MI	"7/1/2002 22:40"
dd/mm/YYYY hh:MI:SS	"7/1/2002 22:40:30"
dd/mm/YYYY hh:MI:SS.SSS	"7/1/2002 22:40:30.125"
hm:MI am	"10:40 pm"
hm:MI:SS am	"10:40:30 pm"
hm:MI:SS.SSS am	"10:40:30.125 pm"
HH:MI	"22:40"
HH:MI:SS	"22:40:30"
HH:MI:SS.SSS	"22:40:30.125"
hh:MI	"22:40"
hh:MI:SS	"22:40:30"
hh:MI:SS.SSS	"22:40:30.125"

Note that the “hh” formats display 24-hour time without leading zeros. In our examples above, there is no difference between the “HH” and “hh” formats for 10 p.m.

Also note that all of the “YYYY” formats above may be displayed using two-digit year “YY” format.

Examples

To set the format of a cell to fixed 5-digit precision, provide the format specification and a valid cell specification:

```
tab1.setformat(A2) f.5
```

You may use any of the date formats given above:

```
tab1.setformat(A3) YYYYMon
tab1.setformat(B1) "YYYY-MM-DD HH:MI:SS.SSS"
```

The cell specification may be described in a variety of ways:

```
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 20. “Working with Tables”](#) of the *User’s Guide I*.

See also [Table::setFont \(p. 518\)](#), [Table::setTextcolor \(p. 528\)](#) and [Table::setfillcolor \(p. 516\)](#) for details on changing the text font and text and cell background colors.

setheight	Table Procs
-----------	-----------------------------

Set the row height of rows in a table.

Syntax

```
table_name.setheight(row_range) height_arg
```

where *row_range* is either a single row number (*e.g.*, “5”), a colon delimited 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 values 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 command:

```
tab1.setheight(2:4) 1
```

sets the heights for rows 2 through 4.

Cross-references

For additional discussion of tables see [Chapter 20. “Working with Tables,”](#) beginning on [page 675](#) of the *User’s Guide I*.

See [Table::setWidth \(p. 529\)](#), [Table::setindent \(p. 524\)](#) and [Table::setjust \(p. 524\)](#) for details on setting table widths, indentation and justification.

setindent	Table Procs
-----------	-----------------------------

Set the display indentation for a table view.

Syntax

```
table_name.setindent(cell_range) 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 table is created.

The *cell_range* defines the cells to be modified. See [Table::setformat \(p. 519\)](#) for the syntax for *cell_range* specifications.

Examples

To set the justification, provide a valid cell specification:

```
tab1.setindent(@all) 2
tab1.setindent(2,B,10,D) 4
tab1.setindent(R2C2:R4C4) 2
```

Cross-references

For additional discussion of tables see [Chapter 20. “Working with Tables,” on page 675](#) of the *User’s Guide I*.

See [Table::setWidth \(p. 529\)](#) and [Table::setjust \(p. 524\)](#) for details on setting table widths and justification.

setjust	Table Procs
---------	-----------------------------

Set the display justification for cells in table views.

Syntax

```
table_name.setjust(cell_range) 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 *cell_range* defines the cells to be modified. See [Table::setformat \(p. 519\)](#) for the syntax for *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 taken from the original view when created by freezing a view, or as “middle bottom” for newly created tables.

Examples

To set the justification, 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 20. “Working with Tables,” on page 675](#) of the *User’s Guide I*.

See [Table::setWidth \(p. 529\)](#) and [Table::setindent \(p. 524\)](#) for details on setting table widths and indentation.

setlines	Table Procs
----------	-----------------------------

Sets line characteristics and borders for a set of table cells.

Syntax

```
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 [Table::setfillcolor \(p. 516\)](#) 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/of]f.

Examples

```
tbl.setlines(b2:d6) +o
```

draws borders around the outside of the rectangle defined by B2 and D6. Note that this command is equivalent to:

```
tbl.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.

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

puts a border around all four sides of the B2 cell.

```
tbl.setlines(2,b) -l -r +i
```

then removes both the left and the right border from the cell. In this example, “+ i” option has no effect; since the specification involves a single cell, there are no inner borders.

```
tbl.setlines(@all) -a
```

removes all borders from the table.

Cross-references

For additional discussion of tables see [Chapter 20. “Working with Tables,” on page 675](#) of the *User’s Guide I*.

See also [Table::settextcolor \(p. 528\)](#) for details on setting cell colors. See [Table::setmerge \(p. 526\)](#) for cell merging, and [Table::comment \(p. 510\)](#) for adding comments to cells.

setmerge	Table Procs
----------	-----------------------------

Merges/unmerges one or more table cells.

Syntax

```
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:

<code>first_cell[:]<i>last_cell</i>,</code> <code>first_cell[,]<i>last_cell</i></code>	Left (right) cell of the selection range (specified in “ <i>cell</i> ” format), followed by right (left) cell of the selection range (specified in “ <i>cell</i> ” format), separated by a “:” or “,” (e.g., “A2:C2”, “A2,C2”, or “R2C1:R2C3”, “R2C1,R2C3”). Merge all cells in the region defined by the first column and last column for the specified row.
<code>cell_row[,]</code> <code>first_cell_col[,]</code> <code>cell_row[,]</code> <code>last_cell_col</code>	Left (right) cell of the selection range (specified in “ <i>row[,]</i> <i>col</i> ” format), followed by right (left) cell of the selection range (specified in “ <i>row[,]</i> <i>col</i> ” format, with a fixed <i>row</i>), separated by a “,” (e.g., “2,A,2,C” or “2,1,2,3”). Merge all cells in the row defined by the first column and last column identifier.

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 left-most cell contents will be used; if merging from right to left, the right-most cell contents will be displayed.

If you specify a merge involving previously merged cells, EViews will unmerge all cells within the specified range.

Examples

```
tab1.setmerge(a2:d2)
```

```
tab1.setmerge(2,1,2,4)
```

merges the cells in row 2, columns 1 to 4, from left to right.

```
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:

```
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:

```
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 20. “Working with Tables,”](#) on page 675 of the *User’s Guide I*.

See also [Table::setlines](#) (p. 525).

settextcolor	Table Procs
---------------------	-----------------------------

Changes the text color of the specified table cells.

Syntax

```
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 [Table::setfillcolor](#) (p. 516) 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 20. “Working with Tables,”](#) on page 675 of the *User’s Guide I*.

See [Table::setFont](#) (p. 518) and [Table::setfillcolor](#) (p. 516) for details on changing the text font and cell background color.

setwidth	Table Procs
----------	-----------------------------

Set the column width for selected columns in a table.

Syntax

```
table_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.

```
tab1.setwidth(2:10) 20
```

sets the widths for columns 2 through 10 to 20 width units.

Cross-references

See [Table::setindent \(p. 524\)](#) and [Table::setjust \(p. 524\)](#) for details on setting table indentation and justification. For details on setting the row heights, see [Table::setheight \(p. 523\)](#).

For additional discussion of tables see [Chapter 20. “Working with Tables,” on page 675](#) of the *User’s Guide I*. See also [Chapter 15. “Graphs, Tables, Text, and Spools,” on page 523](#) of the *User’s Guide I* for a discussion and examples of table formatting in EViews.

sheet	Table Views
-------	-----------------------------

Display a table object.

Syntax

```
table_name.sheet(options)
```

Options

p	Print the spreadsheet view.
---	-----------------------------

Examples

```
tab1.sheet(p)
```

displays and prints table TAB1.

table	Table Declaration
-------	-----------------------------------

Declare a table object.

The `table` command declares and optionally sizes a table object.

Syntax

```
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 contain a single cell.

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 with new dimensions.

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 20. “Working with Tables,” on page 675](#) of the *User’s Guide I* describes table formatting using commands. See [Chapter 15. “Graphs, Tables, Text, and Spools,” on page 523](#) of the *User’s Guide I* for a general discussion and examples of table formatting in EViews.

See also [freeze \(p. 724\)](#).

title	Table Procs
-------	-----------------------------

Assign or change the title of a table.

Syntax

```
table_name.title title_arg
```

where *title_arg* is a case sensitive string which may contain spaces.

Examples

```
tab1.title Estimated Models
```

sets the TAB1 title to “Estimated Models.”

```
tab1.title
```

clears the TAB1 title.

Cross-references

See also [displayname](#) (p. 512) and [label](#) (p. 514).

Text

Text object.
Object for holding arbitrary text information.

Text Declaration

`text`declare text object (p. 535).

To declare a text object, use the keyword `text`, followed by the object name:

```
text mytext
```

Text Views

`label`label information for the text object (p. 534).
`text`view contents of text object (p. 535).

Text Procs

`displayname`.....label information for the text object (p. 533).

Text Examples

```
text mytext
[add text to the object]
mytext.text
```

Text Entries

The following section provides an alphabetical listing of the commands associated with the “Text” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

displayname	Text Procs
-------------	----------------------------

Display name for text objects.

Attaches a display name to a text object which may be used in place of the standard text object name.

Syntax

```
text_name.displayname display_name
```

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in text object names.

Examples

```
hrs.displayname Hours Worked
```

```
hrs.label
```

The first line attaches a display name “Hours Worked” to the text object HRS, and the second line displays the label view of HRS, including its display name.

Cross-references

See “[Labeling Objects](#)” on page 72 of the *User’s Guide I* for a discussion of labels and display names.

See also `Text::label` (p. 534).

label	Text Views Text Procs
-------	---

Display or change the label view of the text object, including the last modified date and display name (if any).

As a procedure, `label` changes the fields in the text object label.

Syntax

```
text_name.label
text_name.label(options) [text]
```

Options

The first version of the command displays the label view of the text object. The second version may be used to modify the label. 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 <i>text</i> .
s	Sets the source field to <i>text</i> .
u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of the text object LWAGE with “Data from CPS 1988 March File”:

```
lwage.label(r)
lwage.label(r) Data from CPS 1988 March File
```

To append additional remarks to LWAGE, and then to print the label view:

```
lwage.label(r) Log of hourly wage
lwage.label(p)
```

To clear and then set the units field, use:

```
lwage.label(u) Millions of bushels
```

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels.

See also [Text::displayname \(p. 533\)](#).

text	Text Declaration Text Views
-------------	--

Declare a text object when used as a command, or display text representation of the text object.

Syntax

```
text object_name
text_name.text(options)
```

When used as a command to declare a table object, follow the keyword with a name of the text object.

Options

p	Print the model text specification.
---	-------------------------------------

Examples

```
text notes1
```

declares a text object named NOTES1.

Cross-references

See [Chapter 15. “Graphs, Tables, Text, and Spools,” on page 523](#) of the *User’s Guide I* for a discussion of text objects in EViews.

Valmap

Valmap (value map).

Valmap Declaration

valmap.....declare valmap object ([p. 541](#)).

To declare a valmap use the keyword `valmap`, followed by a name

```
valmap mymap
```

Valmap Views

labellabel information for the valmap object ([p. 539](#)).

sheetview table of map definitions ([p. 540](#)).

statssummary of map definitions ([p. 540](#)).

usagelist of series and alphas which use the map ([p. 540](#)).

Valmap Procs

appendappend a definition to a valmap ([p. 538](#)).

displaynameset display name ([p. 538](#)).

Valmap Examples

```
valmap b
b.append 0 no
b.append 1 yes
```

declares a valmap B, and adds two map definitions, mapping 0 to “no” and 1 to “yes”.

```
valmap txtmap
txtmap append "NM" "New Mexico"
txtmap append CA California
txtmap append "RI" "Rhode Island"
```

declares the valmap TXTMAP and adds three definitions.

Valmap Entries

The following section provides an alphabetical listing of the commands associated with the “**Valmap**” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

append	Valmap Procs
--------	------------------------------

Append a specification line to a valmap.

Syntax

```
valmap_name.append text
```

Type the text to be added after the `append` keyword.

Examples

```
valmap b
b.append 0 no
b.append 1 yes
```

The first line declares a valmap object. The following lines set the specification for that valmap - 0's are mapped to “no” and 1's are mapped to “yes”.

Cross-references

For details, see [“Value Maps” on page 159](#) of the *User’s Guide I*.

displayname	Valmap Procs
-------------	------------------------------

Display name for a valmap objects.

Attaches a display name to a valmap which may be used to label output in place of the standard valmap object name.

Syntax

```
valmap_name.displayname display_name
```

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in valmap object names.

Examples

```
hrs.displayname Valmap for Hours Worked
hrs.label
```

The first line attaches a display name “Valmap for Hours Worked” to the valmap object HRS, and the second line displays the label view of HRS, including its display name.

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels and display names.

See also [valmap::label](#) (p. 539).

label	Valmap Views Valmap Procs
-------	---

Display or change the label view of a valmap, including the last modified date and display name (if any).

As a procedure, `label` changes the fields in the valmap label.

Syntax

```
valmap_name.label
valmap_name.label(options) [text]
```

Options

The first version of the command displays the label view of the valmap. The second version may be used to modify the label. 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 <i>text</i> .
s	Sets the source field to <i>text</i> .
u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of VMAP with “Data from CPS 1988 March File”:

```
vmmap.label(r)
vmmap.label(r) Data from CPS 1988 March File
```

To append additional remarks to VMAP, and then to print the label view:

```
vmmap.label(r) Log of hourly wage
vmmap.label(p)
```

To clear and then set the units field, use:

```
vmmap.label(u) Millions of bushels
```

Cross-references

See [“Labeling Objects”](#) on page 72 of the *User’s Guide I* for a discussion of labels.

See also [Valmap::displayname](#) (p. 538).

sheet	Valmap Views
-------	------------------------------

Spreadsheet view of a valmap object.

Syntax

`valmap_name.sheet(options)`

Options

p	Print the spreadsheet view.
---	-----------------------------

Examples

`vm1.sheet`

displays the spreadsheet view of the valmap object VM1.

stats	Valmap Views
-------	------------------------------

Statistics for valmap usage.

Displays a description of the composition of a valmap.

Syntax

`valmap_name.stats(options)`

Options

p	Print the table.
---	------------------

Examples

`map1.stats`

displays the summary descriptive view of the definitions in the valmap MAP1.

Cross-references

See [“Value Maps” on page 159](#) of the *User’s Guide I* for a discussion of value maps.

usage	Valmap Views
-------	------------------------------

Find series and alphas which use the valmap.

Display list of series and alpha objects which use the valmap.

Syntax

```
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 159](#) of the *User’s Guide I*.

See also [Series::map \(p. 373\)](#) and [Alpha::map \(p. 12\)](#).

valmap	Valmap Declaration
--------	------------------------------------

Declare a value map object.

Syntax

```
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”.

Cross-references

For additional details, see [“Value Maps” on page 159](#) of the *User’s Guide I*.

See also [Series::map \(p. 373\)](#) and [Alpha::map \(p. 12\)](#).

Var

Vector autoregression and error correction object.

Var Declaration

vardeclare var estimation object (p. 572).

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

ecestimate a vector error correction model (p. 553).

lsestimate an unrestricted VAR (p. 561).

Var Views

arlmserial correlation LM test (p. 546).

arrootsinverse roots of the AR polynomial (p. 547).

cointJohansen cointegration test (p. 548).

correlresidual autocorrelations (p. 550).

decompvariance decomposition (p. 551).

endogtable or graph of endogenous variables (p. 555).

impulseimpulse response functions (p. 555).

jberaresidual normality test (p. 558).

labellabel information for the var object (p. 559).

laglenlag order selection criteria (p. 560).

outputtable of estimation results (p. 565).

qstatsresidual portmanteau tests (p. 565).

representationstext describing var specification (p. 566).

residcorresidual correlation matrix (p. 566).

residcovresidual covariance matrix (p. 567).

residsresidual graphs (p. 567).

resultstable of estimation results (p. 568).

testexogexogeneity (Granger causality) tests (p. 570).

testlagslag exclusion tests (p. 571).

whiteWhite heteroskedasticity test (p. 573).

Var Procs

appendappend restriction text (p. 546).

cleartext clear restriction text (p. 548).
displayname set display name (p. 553).
makeoint make group of cointegrating relations (p. 562).
makeendog make group of endogenous series (p. 562).
makemodel make model from the estimated var (p. 563).
makeresids make residual series (p. 563).
makesystem make system from var (p. 564).
svar estimate structural factorization (p. 568).

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 .
@sddep(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.
@detresid 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).
- @svarooverid**.....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.makesresids
```

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.makesresids payres hholdres gdpres
empvar.makemodel(inmdl) cp fcp div fdiv r fr
```

To save coefficients in a scalar:

```
scalar coef1=empvar.b(1,2)
```

Var Entries

The following section provides an alphabetical listing of the commands associated with the “[Var](#)” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

append	Var Procs
---------------	---------------------------

Append a specification line to a var.

Syntax

```
var_name.append(options) text
```

Type the text to be added after the `append` keyword. *You must specify the restrictions type option.*

Options

One of the following options is required when using `append` as a var proc:

<code>svar</code>	Text for identifying restrictions for structural VAR.
<code>coint</code>	Text for restrictions on the cointegration relations and/or adjustment coefficients.

Examples

```
var v
v.append(coint) b(1,1)=1
v.ec(restrict) 1 4 x y
```

First a VEC, `V`, is declared, then a restriction is appended to `V`, finally `V` is estimated with that restriction imposed.

Cross-references

See also [Var::cleartext](#) (p. 548).

arlm	Var Views
-------------	---------------------------

Perform multivariate residual serial correlation LM test using an estimated Var.

Syntax

```
var_name.arlm(h, options)
```

You must specify the highest order of lag, *h*, for which to test.

Options

name = <i>arg</i>	Save LM statistics in named matrix object. The matrix has <i>h</i> rows and one column.
p	Print test output.

Examples

```
var var1.ls 1 6 lgdp lm1 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 348](#) of the *User’s Guide II* for other VAR diagnostics. See also [Var::qstats \(p. 565\)](#) for related multivariate residual autocorrelation Portmanteau tests.

arroots	Var Views
---------	---------------------------

Inverse roots of the characteristic AR polynomial.

Syntax

```
var_name.arroots(options)
```

Options

name = <i>arg</i>	Save roots in named matrix object. Each root is saved in a row of the matrix, with the first column containing the real, and the second column containing the imaginary part of the root.
graph	Plots the roots together with a unit circle. The VAR is stable if all of the roots are inside the unit circle.
p	Print table of AR roots.

Examples

```
var var1.ls 1 6 lgdp lm1 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.

```
var var1.ls 1 6 lgdp lm1 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 348](#) of the *User’s Guide II* for other VAR diagnostics.

cleartext	Var Procs
-----------	---------------------------

Clear restriction text from a var object.

Syntax

```
var_name.cleartext(arg)
```

You must specify the text type you wish to clear using one of the following arguments:

svar	Clear text of identifying restrictions for a structural VAR.
coint	Clear text of restrictions on the cointegration relations and/or adjustment coefficients.

Examples

```
var1.cleartext(svar)
var1.append(svar) @lr2(@u1) = 0
```

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 [Chapter 34. “Vector Autoregression and Error Correction Models,” on page 345](#) of the *User’s Guide II* for a discussion of VARs.

See also [Var::append \(p. 546\)](#).

coint	Var Views
-------	---------------------------

Johansen’s cointegration test.

Syntax

```
var_name.coint(test_option,n,option) [@ x1 x2 x3 ...]
```

The `coint` command tests for cointegration among the series in the var. By default, 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 the listed variables will be used in the test.

As of EViews 5, the output for cointegration tests now displays p -values for the rank test statistics. These p -values are computed using the response surface coefficients as estimated in MacKinnon, et. al. (1999). The 0.05 critical values are now based on the response surface coefficients from MacKinnon-Haug-Michelis. *Note: the reported critical values assume no exogenous variables other than an intercept and trend.*

Options

You must specify the test option followed by the number of lags n . You must choose one of the following six test options:

a	No deterministic trend in the data, and no intercept or trend in the cointegrating equation.
b	No deterministic trend in the data, and an intercept but no trend in the cointegrating equation.
c	Linear trend in the data, and an intercept but no trend in the cointegrating equation.
d	Linear trend in the data, and both an intercept and a trend in the cointegrating equation.
e	Quadratic trend in the data, and both an intercept and a trend in the cointegrating equation.
s	Summarize the results of all 5 options (a-e).

Other Options:

restrict	Impose restrictions as specified by the <code>append (coint)</code> proc.
m = <i>integer</i>	Maximum number of iterations for restricted estimation (only valid if you choose the restrict option).
c = <i>scalar</i>	Convergence criterion for restricted estimation. (only valid if you choose the restrict option).

<code>save = mat_name</code>	Stores test statistics as a named matrix object. The <code>save =</code> option stores a $(k + 1) \times 4$ matrix, where k is the number of endogenous variables in the VAR. The first column contains the eigenvalues, the second column contains the maximum eigenvalue statistics, the third column contains the trace statistics, and the fourth column contains the log likelihood values. The i -th row of columns 2 and 3 are the test statistics for rank $i - 1$. The last row is filled with NAs, except the last column which contains the log likelihood value of the unrestricted (full rank) model.
<code>cvtype = ol</code>	Display 0.05 and 0.01 critical values from Osterwald-Lenum (1992). This option reproduces the output from version 4. The default is to display critical values based on the response surface coefficients from MacKinnon-Haug-Michelis (1999). Note that the argument on the right side of the equals sign are letters, not numbers 0-1).
<code>cvsize = arg</code> (<i>default</i> = 0.05)	Specify the size of MacKinnon-Haug-Michelis (1999) critical values to be displayed. The size must be between 0.0001 and 0.9999; values outside this range will be reset to the default value of 0.05. This option is ignored if you set “ <code>cvtype = ol</code> ”.
<code>p</code>	Print output of the test.

Examples

```
var1.coint(c,12) @
```

carries 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

See [“Cointegration Testing” on page 363](#) of the *User’s Guide II* for details on the Johansen test.

See also [Var::ec \(p. 553\)](#).

correl	Var Views
--------	---------------------------

Display autocorrelation and partial correlations.

Displays the autocorrelation and partial correlation functions in the specified form, together with the Q -statistics and p -values associated with each lag.

Syntax

```
var_name.correl(n, options)
```

You must specify the largest lag n to use when computing the autocorrelations.

Options

graph	Display correlograms (graphs).
byser	Display autocorrelations in tabular form, by series.
bylag	Display autocorrelations in tabular form, by lag.
p	Print the correlograms.

Examples

```
v1.correl(24, byser)
```

Displays the correlograms of V1 in tabular form by series, for up to 24 lags.

Cross-references

See [“Autocorrelations \(AC\)” on page 325](#) and [“Partial Autocorrelations \(PAC\)” on page 326](#) of the *User’s Guide I* for a discussion of autocorrelation and partial correlation functions, respectively.

decomp	Var Views
--------	---------------------------

Variance decomposition in VARs.

Syntax

```
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

<i>g</i> (<i>default</i>)	Display combined graphs, with the decompositions for each variable shown in a graph.
m	Display multiple graphs, with each response-shock pair shown in a separate graph.
t	Show numerical results in table.

<code>imp = <i>arg</i></code> (<i>default</i> = “chol”)	Type of factorization for the decomposition: “chol” (Cholesky with d.f. correction), “mlechol” (Cholesky without d.f. correction), “struct” (structural). The structural factorization is based on the estimated structural VAR. To use this option, you must first estimate the structural decomposition; see Var:svar (p. 568). The option “imp = mlechol” is provided for backward compatibility with EViews 3.x and earlier.
<code>se = mc</code>	Monte Carlo standard errors. You must specify the number of replications with the “rep = ” option. Currently available only when you have specified the Cholesky factorization (using the “imp = chol” option).
<code>rep = <i>integer</i></code>	Number of Monte Carlo replications to be used in computing the standard errors. Must be used with the “se = mc” option.
<code>matbys = <i>name</i></code>	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.
<code>matbyr = <i>name</i></code>	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.
<code>p</code>	Print results.

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

```
var var1.ls 1 4 m1 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.

```
var1.decomp(10,t) gdp @ @ cpi gdp m1
```

performs the same variance decomposition as above using a different ordering.

Cross-references

See [“Variance Decomposition” on page 355](#) of the *User’s Guide II* for additional details.

See also [Var::impulse](#) (p. 555).

displayname	Var Procs
-------------	---------------------------

Display name for a var object.

Attaches a display name to a var object which may be used to label output in place of the standard var object name.

Syntax

```
var_name.displayname display_name
```

Display names are case-sensitive, and may contain a variety of characters, such as spaces, that are not allowed in var object names.

Examples

```
hrs.displayname Hours Worked
hrs.label
```

The first line attaches a display name “Hours Worked” to the var object HRS, and the second line displays the label view of HRS, including its display name.

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels and display names.

See also [Var::label](#) (p. 559).

ec	Var Methods
----	-----------------------------

Estimate a vector error correction model (VEC).

Syntax

```
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 should specify the trend option and the number of cointegrating equations n to use in parentheses, separated by a comma (the default is $n = 1$). You must choose the trend from the following five alternatives:

a	No deterministic trend in the data, and no intercept or trend in the cointegrating equation.
b	No deterministic trend in the data, and an intercept but no trend in the cointegrating equation.
c (default)	Linear trend in the data, and an intercept but no trend in the cointegrating equation.
d	Linear trend in the data, and both an intercept and a trend in the cointegrating equation.
e	Quadratic trend in the data, and both an intercept and a trend in the cointegrating equation.
restrict	Impose restrictions. See Var::append (p. 546) and Var::coint (p. 548) .
m = <i>integer</i>	Maximum number of iterations for restricted estimation (only valid if you choose the restrict option).
c = <i>scalar</i>	Convergence criterion for restricted estimation. (only valid if you choose the restrict option).

Options

p	Print the results view.
---	-------------------------

Examples

```
var macro1.ec 1 4 m1 gdp tb3
```

declares a var object MACRO1 and estimates a VEC with four lagged first differences, three endogenous variables and one cointegrating equation using the default trend option “c”.

```
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 “[Vector Error Correction \(VEC\) Models](#)” on page 377 of the *User’s Guide II* for a discussion of VECs.

See also [Var::var](#) (p. 572), [Var::coint](#) (p. 548) and [Var::append](#) (p. 546). [Var::ls](#) (p. 561) is used to estimate unrestricted VAR models.

endog	Var Views
-------	---------------------------

Displays a spreadsheet or graph view of the endogenous variables.

Syntax

```
var_name.endog(options)
```

Options

g	Multiple line graphs of the solved endogenous series.
p	Print the table of solved endogenous series.

Examples

```
var1.endog(g,p)
```

prints the graphs of the solved endogenous series.

Cross-references

See also [Var::makeendog](#) (p. 562) and [Var::var](#) (p. 572).

impulse	Var Views
---------	---------------------------

Display impulse response functions of var object with an estimated VAR or VEC.

Syntax

```
var_name.impulse(n, options) ser1 [ser2 ser3 ...] [@ shock_series [@ ordering_series]]
```

You must specify the number of periods n for which you wish to compute the impulse responses.

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 “@”.

Options

<code>g</code> (<i>default</i>)	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.
<code>m</code>	Display multiple graphs, with impulse response to each shock shown in separate graphs.
<code>t</code>	Tabulate the impulse responses.
<code>a</code>	Accumulate the impulse responses.
<code>imp = arg</code> (<i>default</i> = “chol”)	<p>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”).</p> <p>The structural factorization is based on the estimated structural VAR. To use this option, you must first estimate the structural decomposition; see Var : svar (p. 568).</p> <p>For user-specified impulses, you must specify the name of the vector/matrix containing the impulses using the “fname = ” option.</p> <p>The option “imp = mlechol” is provided for backward compatibility with EViews 3.x and earlier.</p>
<code>fname = name</code>	<p>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.</p>
<code>se = arg</code>	<p>Standard error calculations: “se = a” (analytic), “se = mc” (Monte Carlo).</p> <p>If selecting Monte Carlo, you must specify the number of replications with the “rep = ” option.</p> <p>Note the following:</p> <p>(1) Analytic standard errors are currently not available for (a) VECs and (b) structural decompositions identified by long-run restrictions. The “se = a” option will be ignored for these cases.</p> <p>(2) Monte Carlo standard errors are currently not available for (a) VECs and (b) structural decompositions. The “se = mc” option will be ignored for these cases.</p>

<code>rep = integer</code>	Number of Monte Carlo replications to be used in computing the standard errors. Must be used with the “se = mc” option.
<code>matbys = name</code>	Save responses ordered by shocks (impulses) in a 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. <i>The response and shock orderings correspond to the ordering of variables in the VAR.</i>
<code>matbyr = name</code>	Save responses ordered by response series in a 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. <i>The response and shock orderings correspond to the ordering of variables in the VAR.</i>
<code>smat = name</code>	Save responses ordered by shocks (impulses) in a named matrix (akin to the <code>matbys =</code> option). The shocks and responses are ordered according to the user-specified order given by the <code>@ shock_series</code> and <code>@ ordering_series</code> arguments
<code>rmat = name</code>	Save responses ordered by response series in a named matrix (akin to the <code>matbyr =</code> option). The shocks and responses are ordered according to the user-specified order given by the <code>@ shock_series</code> and <code>@ ordering_series</code> arguments.
<code>p</code>	Print the results.

Examples

```
var var1.ls 1 4 m1 gdp cpi
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.

```
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 34. “Vector Autoregression and Error Correction Models,”](#) on page 345 of the *User’s Guide II* for a discussion of variance decompositions in VARs.

See also `Var::decomp` (p. 551).

jbera	Var Views
-------	---------------------------

Multivariate residual normality test.

Syntax

```
var_name.jbera(options)
```

You must specify a factorization method using 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 Var::svar (p. 568).
name = arg	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 first column contains the third moments, the second column contains the χ^2_1 statistics for the third moments, the third column contains the fourth moments, and the fourth column holds the χ^2_1 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 last 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 χ^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 = 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 statistic is stored in the first column of the $(k + 1)$ row (which will be a missing value for all other options).

Examples

```
var var1.ls 1 6 lgdp lm1 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 34. “Vector Autoregression and Error Correction Models,”](#) on page 345 of the *User’s Guide II* for a discussion of the test and other VAR diagnostics.

label	Var Views Var Procs
-------	---------------------------------------

Display or change the label view of a var object, including the last modified date and display name (if any).

As a procedure, `label` changes the fields in the var object label.

Syntax

```
var_name.label
var_name.label(options) [text]
```

Options

The first version of the command displays the label view of the var object. The second version may be used to modify the label. 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 <i>text</i> .
s	Sets the source field to <i>text</i> .
u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of VAR1 with “Data from CPS 1988 March File”:

```
var1.label(r)
var1.label(r) Data from CPS 1988 March File
```

To append additional remarks to VAR1, and then to print the label view:

```
var1.label(r) Log of hourly wage
var1.label(p)
```

To clear and then set the units field, use:

```
var1.label(u) Millions of bushels
```

Cross-references

See “Labeling Objects” on page 72 of the *User’s Guide I* for a discussion of labels.

See also `Var::displayname` (p. 553).

laglen	Var Views
--------	---------------------------

VAR lag order selection criteria.

Syntax

```
var_name.laglen(m, options)
```

You must specify the maximum lag order *m* for which you wish to test.

Options

<code>vname = arg</code>	Save selected lag orders in named vector. See below for a description of the stored vector.
<code>mname = arg</code>	Save lag order criteria in named matrix. See below for a description of the stored matrix.
<code>p</code>	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. 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

```
var var1.ls 1 6 lgdp lm1 lcp1
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 34. “Vector Autoregression and Error Correction Models,”](#) on page 345 of the *User’s Guide II* for a discussion of the various criteria and other VAR diagnostics.

See also [Var::testlags](#) (p. 571).

ls	Var Methods
----	-----------------------------

Estimate VAR specification.

Syntax

```
var_name.ls(options) lag_pairs endog_list [@ exog_list]
```

ls 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. A constant is automatically added to the list of exogenous variables; to estimate a specification without a constant, you should use the option “noconst”.

Options

General options

noconst	Do not include a constant in exogenous regressors list for VARs.
p	Print basic estimation results.

Examples

```
var mvar.ls 1 3 m1 gdp
```

declares and estimates an unrestricted VAR named MVAR with two endogenous variables (M1 and GDP), a constant and 3 lags (lags 1 through 3).

```
mvar.ls(noconst) 1 3 m1 gdp
```

estimates the same VAR, but with no constant.

Cross-references

See [Chapter 34. “Vector Autoregression and Error Correction Models,”](#) on page 345 of the *User’s Guide II* for details.

See also [Var::ec](#) (p. 553) for estimation of error correction models.

makecoint[Var Procs](#)

Create group containing the estimated cointegrating relations from a VEC.

Syntax

```
var_name.makecoint [group_name]
```

The series contained in the group are given names of the form “COINTEQ##”, where ## are numbers such that “COINTEQ##” is the next available unused name.

If you provide a name for the group in parentheses after the keyword, EViews will quietly create the named group in the workfile. If you do not provide a name, EViews will open an untitled group window if the command is executed from the command line, otherwise no group will be created.

This proc will return an error message unless you have estimated an error correction model using the var object.

Examples

```
var vec1.ec(b,2) 1 4 y1 y2 y3  
vec1.makecoint gcount
```

The first line estimates a VEC with 2 cointegrating relations. The second line creates a group named GCOUNT 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).

Cross-references

See [Chapter 34. “Vector Autoregression and Error Correction Models,”](#) on page 345 of the *User’s Guide II* for details.

See also [Var::coint](#) (p. 548).

makeendog[Var Procs](#)

Make a group out of the endogenous series.

Syntax

```
var_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.

Examples

```
var1.makeendog grp_v1
```

creates a group named GRP_V1 that contains the endogenous series in VAR1.

Cross-references

See also [Var::endog \(p. 555\)](#) and [Model::makegroup \(p. 282\)](#).

makemodel	Var Procs
------------------	---------------------------

Make a model from a var object.

Syntax

```
var_name.makemodel(name) assign_statement
```

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

```
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 [Chapter 36. “Models,” on page 407](#) of the *User’s Guide II* for a discussion of specifying and solving models in EViews.

See also [Var::append \(p. 546\)](#), [Model::merge \(p. 283\)](#) and [Model::solve \(p. 287\)](#).

makeresids	Var Procs
-------------------	---------------------------

Create residual series.

Creates and saves residuals in the workfile from an estimated VAR.

Syntax

```
var_name.makeresids[res1 res2 res3]
```

Follow the VAR name with a period and the `makeresids` keyword, then provide a list of names to be given to the stored residuals. 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.

Options

<code>n = arg</code>	Create group object to hold the residual series.
----------------------	--

Examples

```
var macro_var.ls 1 4 y m1 r
macro_var.makesresids resay res_m1 riser
```

estimates an unrestricted VAR with four lags and endogenous variables Y, M1, and R, and stores the residuals as RES_Y, RES_M1, RES_R.

Cross-references

See [“Weighted Least Squares” on page 32](#) of the *User’s Guide II* for a discussion of standardized residuals after weighted least squares and [Chapter 30. “Discrete and Limited Dependent Variable Models,” on page 209](#) of the *User’s Guide II* for a discussion of standardized and generalized residuals in binary, ordered, censored, and count models.

makesystem	Var Procs
-------------------	---------------------------

Create system from a var.

Syntax

```
var_name.makesystem(options)
```

You may order the equations by series (*default*) or by lags.

Options

<code>bylag</code>	Specify system by lags (default is to order by variables).
<code>n = name</code>	Specify name for the object.

Examples

```
var1.makesystem(n=sys1)
```

creates a system named SYS1 from the var object VAR1

Cross-references

See [Chapter 33. “System Estimation,” on page 307](#) of the *User’s Guide II* for a discussion of system objects in EViews.

output	Var Views
---------------	---------------------------

Display estimation output.

`output` changes the default object view to display the estimation output (equivalent to using `Var::results` (p. 568)).

Syntax

```
var_name.output
```

Options

p	Print estimation output for estimation object
---	---

Examples

The `output` keyword may be used to change the default view of an estimation object. Entering the command:

```
var1.output
```

displays the estimation output for VAR1.

Cross-references

See `Var::results` (p. 568).

qstats	Var Views
---------------	---------------------------

Multivariate residual autocorrelation Portmanteau tests.

Syntax

```
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

<code>name = arg</code>	Save Q -statistics in the named matrix object. The matrix has two columns: the first column contains the unmodified Q -statistic; the second column contains the modified Q -statistics.
p	Print the Portmanteau test results.

Examples

```
var var1.ls 1 6 lgdp lml lcpi
show var1.qstats(10, name=q)
```

The first line declares and estimates a VAR. The second line displays the portmanteau tests for lags up to 10, and stores the Q -statistics in a matrix named Q .

Cross-references

See [“Diagnostic Views” on page 348](#) of the *User’s Guide II* for a discussion of the Portmanteau tests and other VAR diagnostics.

See [Var::arlm \(p. 546\)](#) for a related multivariate residual serial correlation LM test.

representations	Var Views
-----------------	---------------------------

Display text of specification for var objects.

Syntax

```
var_name.representation(options)
```

Options

p	Print the representation text.
---	--------------------------------

Examples

```
var1.representations
```

displays the specifications of the estimation object VAR1.

residcor	Var Views
----------	---------------------------

Residual correlation matrix.

Displays the correlations of the residuals from each equation in the var object.

Syntax

```
var_name.residcor(options)
```

Options

p	Print the correlation matrix.
---	-------------------------------

Examples

```
var1.residcor
```

displays the residual correlation matrix of VAR1.

Cross-references

See also [Var::residcov](#) (p. 567) and [Var::makeresids](#) (p. 563).

residcov	Var Views
-----------------	---------------------------

Residual covariance matrix.

Displays the covariances of the residuals from each equation in the var object.

Syntax

```
var_name.residcov(options)
```

Options

p	Print the covariance matrix.
---	------------------------------

Examples

```
var1.residcov
```

displays the residual covariance matrix of VAR1.

Cross-references

See also [Var::residcor](#) (p. 566) and [Var::makeresids](#) (p. 563).

resids	Var Views
---------------	---------------------------

Display residuals.

`resids` displays multiple graphs of the residuals. Each graph will contain the residuals for an equation in the VAR.

Syntax

```
var_name.resids(options)
```

Options

p	Print the table/graph.
---	------------------------

Examples

```
var var1.ls 1 3 m1 c
```

```
var1.resids
```

calculates a VAR with three lags, two endogenous variables and a constant term, and then displays a graph of the residuals.

Cross-references

See also [Var::makeresids](#) (p. 563).

results	Var Views
---------	---------------------------

Displays the results view of an estimated VAR.

Syntax

```
var_name.results(options)
```

Options

p	Print the view.
---	-----------------

Examples

```
var mvar.ls 1 4 8 8 m1 gdp tb3 @ @trend(70.4)
mvar.results(p)
```

prints the estimation results from the estimated VAR.

svar	Var Procs
------	---------------------------

Estimate factorization matrix for structural innovations.

Syntax

```
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 357](#) of the *User’s Guide II* for details on specifying restrictions.

Options

You must specify one of the following restriction type:

<code>rtype = text</code>	Text form restrictions. The restrictions must be specified by the <code>append</code> command to use this option.
<code>rtype = patsr</code>	Short-run pattern restrictions. You must provide the names of the patterned matrices by the “ <code>namea =</code> ” and “ <code>nameb =</code> ” options as described below.
<code>rtype = patlr</code>	Long-run pattern restrictions. You must provide the name of the patterned matrix by the “ <code>namelr =</code> ” option as described below.

Other Options:

<code>namea = arg,</code> <code>nameb = arg</code>	Names of the pattern matrices for A and B matrices. Must be used with “ <code>rtype = patsr</code> ”.
<code>namelr = arg</code>	Name of the pattern matrix for long-run impulse responses. Must be used with “ <code>rtype = patlr</code> ”.
<code>fsign</code>	Do not apply the sign normalization rule. Default is to apply the sign normalization rule whenever applicable. See “Sign Indeterminacy” on page 362 of the <i>User’s Guide II</i> for a discussion of the sign normalization rule.
<code>f0 = arg</code> (<i>default</i> = 0.1)	Starting values for the free parameters: “ <i>scalar</i> ” (specify fixed value for starting values), “ <i>s</i> ” (user specified starting values are taken from the C coefficient vector), “ <i>u</i> ” (draw starting values for free parameters from a uniform distribution on [0,1]), “ <i>n</i> ” (draw starting values for free parameters from standard normal).
<code>maxiter = integer</code>	Maximum number of iterations. Default is taken from global option setting.
<code>conv = number</code>	Convergence criterion. Default is taken from global option setting.
<code>trace = integer</code>	Trace iterations process every <i>integer</i> iterations (displays an untitled text object containing summary information).
<code>nostop</code>	Suppress “Near Singular Matrix” error message even if Hessian is singular at final parameter estimates.

Examples

```
var var1.ls 1 4 m1 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.

```
var var1.ls 1 8 dy u @
var1.append(svar) @lr1(@u1)=0
freeze(out1) var1.svar(rtype=text)
```

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 357](#) of the *User’s Guide II* for a discussion of structural VARs.

testexog	Var Views
----------	---------------------------

Perform exogeneity (Granger causality) tests on a VAR.

Syntax

```
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 the results in a $(k + 1) \times k$ matrix, where k is the number of endogenous variables in the VAR. In the first k rows, the i -th row, j -th column contains 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 is the number of lags included in the VAR.

In the last row, the j -th column 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 the number of lags included in the VAR.

Examples

```
var var1.ls 1 6 lgdp lml lcp1
```

```
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 348 of the *User’s Guide II* for a discussion of other VAR diagnostics.

See also `Var::testlags` (p. 571).

testlags	Var Views
----------	---------------------------

Perform lag exclusion (Wald) tests on a VAR.

Syntax

```
var_name.testlags(options)
```

Options

<code>name = arg</code>	Save the Wald test statistics in named matrix object. See below for a description of the statistics contained in the stored matrix.
<code>p</code>	Print the result of the test.

The “name = ” option stores results in 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. In 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 χ^2 distribution with k degrees of freedom under the exclusion null.

In the last column, the i -th row 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 lm1 lcp1
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 348 of the *User’s Guide II* for a discussion other VAR diagnostics.

See also `Var::laglen` (p. 560) and `Var::testexog` (p. 570).

var	Var Declaration
-----	---------------------------------

Declare a var (Vector Autoregression) object.

Syntax

```
var var_name
var var_name.ls(options) lag_pairs endog_list [@ exog_list]
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 and specification.

The `Var::ls` (p. 561) 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. A constant is automatically added to the list of exogenous variables; to estimate a specification without a constant, you should use the option “noconst”.

See `Var::ec` (p. 553) for the error correction specification of a VAR.

Options

noconst	Do not include a constant in the VAR specification (when combining declaration with <code>Var::ls</code> (p. 561) method).
p	Print the estimation result if the estimation procedure is specified.

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), a constant, 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 [Chapter 34. “Vector Autoregression and Error Correction Models,”](#) on page 345 of the *User’s Guide II* for a discussion of vector autoregressions.

See [Var::ls](#) (p. 561) for standard VAR estimation, and [Var::ec](#) (p. 553) for estimation of error correction models.

white	Var Views
-------	---------------------------

Performs White’s test for heteroskedasticity of residuals.

Carries out White’s multivariate test for heteroskedasticity of the residuals of the specified Var object. 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 that includes the cross-products.

Syntax

```
var_name.white(options)
```

Options

c	Include all possible nonredundant cross-product terms in the test regression.
name = <i>arg</i>	Save test statistics in named matrix object. See below for a description of the statistics stored in the matrix.
p	Print the test results.

The “name = ” option stores the results in 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 $T \times R^2 \chi^2$ statistic, and the fifth column is the p -value of the χ^2 statistic.

The numerator and denominator degrees of freedom of the F -statistic are stored in the third and fourth columns, respectively, of the $(r + 1)$ -st row, while the χ^2 degrees of freedom is stored in the fifth column of the $(r + 1)$ -st row.

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 χ^2 statistic.

Examples

```
var1.white
```

carries out the White test of heteroskedasticity.

Cross-references

See [“White's Heteroskedasticity Test” on page 158](#) of the *User's Guide II* for a discussion of White's test. For the multivariate version of this test, see [“White Heteroskedasticity Test” on page 352](#) of the *User's Guide II*.

References

- Doornik, Jurgen A. and Henrik Hansen (1994). “An Omnibus Test for Univariate and Multivariate Normality,” manuscript.
- MacKinnon, James G., Alfred A. Haug, and Leo Michelis (1999), “Numerical Distribution Functions of Likelihood Ratio Tests For Cointegration,” *Journal of Applied Econometrics*, 14, 563-577.
- Osterwald-Lenum, Michael (1992). “A Note with Quantiles of the Asymptotic Distribution of the Maximum Likelihood Cointegration Rank Test Statistics,” *Oxford Bulletin of Economics and Statistics*, 54, 461-472.
- Urzua, Carlos M. (1997). “Omnibus Tests for Multivariate Normality Based on a Class of Maximum Entropy Distributions,” in *Advances in Econometrics*, Volume 12, Greenwich, Conn.: JAI Press, 341-358.

Vector

Vector. (One dimensional array of numbers).

Vector Declaration

vectordeclare vector object (p. 587).

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 = vec1
```

Vector Views

corcompute variance measures for the data in the vector (p. 576).
covcompute variance measures for the data in the vector (p. 576).
labellabel information for the vector object (p. 580).
sheetspreadsheet view of the vector (p. 586).
statsdescriptive statistics (p. 587).

Vector Graph Views

Graph creation types are discussed in detail in “Graph Creation Commands” on page 601.

areaarea graph of the vector (p. 603).
barbar graph of data against the row index (p. 609).
boxplotboxplot graph (p. 613).
distplotdistribution graph (p. 615).
dotdot plot graph (p. 622).
lineline graph of the data against the row index (p. 630).
qqplotquantile-quantile graph (p. 636).
seasplotseasonal line graph (p. 651).
spikespike graph (p. 652).

Vector Procs

displaynameset display name (p. 579).
fillfill elements of the vector (p. 580).
readimport data from disk (p. 581).
setformatset the display format for the vector spreadsheet (p. 583).

- [setindent](#) set the indentation for the vector spreadsheet (p. 584).
- [setjust](#) set the justification for the vector spreadsheet (p. 585).
- [setwidth](#) set the column width for the vector spreadsheet (p. 586).
- [write](#) export data to disk (p. 588).

Vector Data Members

- (i) *i*-th element of the vector. Simply append “(i)” to the vector name (without a “.”).

Vector Entries

The following section provides an alphabetical listing of the commands associated with the “[Vector](#)” object. Each entry outlines the command syntax and associated options, and provides examples and cross references.

cor	Vector Views
---------------------	------------------------------

Compute variance measures for the vector. You may compute measures related to Pearson product-moment (ordinary) variance, rank variance, or Kendall’s tau.

Syntax

`vector_name.cor(options) [keywords [@partial z1 z2 z3...]]`

Note that `cor` is equivalent to the [cov](#) (p. 576) command with a different default setting. If you do not specify *keywords*, EViews will assume “cor” and compute the Pearson variance. The default setting is not particularly interesting since the Pearson correlation for the data in the vector is identically equal to 1.

See [cov](#) (p. 576) for details.

cov	Vector Views
---------------------	------------------------------

Compute variance measures for the vector. You may compute measures related to Pearson product-moment (ordinary) variance, rank variance, or Kendall’s tau.

Syntax

`vector_name.cov(options) [keywords [@partial z1 z2 z3...]]`

You should specify keywords indicating the statistics you wish to display from the list below, optionally followed by the keyword `@partial` and the name of a conditioning matrix. In the matrix view setting, the columns of the matrix should contain the conditioning information, and the number or rows should match the original matrix.

You may specify keywords from one of the four sets (Pearson correlation, Spearman correlation, Kendall's tau, Uncentered Pearson) corresponding the computational method you wish to employ. (You may not select keywords from more than one set.) Note that the Kendall's tau measures are not particularly interesting since they generally will be equal, or nearly equal, to 1.

If you do not specify *keywords*, EViews will assume “cov” and compute the Pearson variance. Note that `cov` is equivalent to the `cor` (p. 576) command with a different default setting.

Note that this view has been expanded considerably from EViews 5.1, but the syntax for the earlier version of the routine is fully compatible with the current version.

Pearson Correlation

<code>cov</code>	Product moment covariance.
<code>corr</code>	Product moment correlation.
<code>sscp</code>	Sums-of-squared cross-products.
<code>cases</code>	Number of cases.
<code>obs</code>	Number of observations.
<code>wgts</code>	Sum of the weights.

Spearman Rank Correlation

<code>rcov</code>	Spearman's rank covariance.
<code>rcorr</code>	Spearman's rank correlation.
<code>rsscp</code>	Sums-of-squared cross-products.
<code>cases</code>	Number of cases.
<code>obs</code>	Number of observations.
<code>wgts</code>	Sum of the weights.

Kendall's tau

<code>taub</code>	Kendall's tau-b.
<code>taua</code>	Kendall's tau-a.
<code>cases</code>	Number of cases.
<code>obs</code>	Number of observations.
<code>wgts</code>	Sum of the weights.

Uncentered Pearson

ucov	Product moment covariance.
ucorr	Product moment correlation.
usscp	Sums-of-squared cross-products.
cases	Number of cases.
obs	Number of observations.
wgts	Sum of the weights.

Note that `cases`, `obs`, and `wgts` are available for each of the methods.

Options

<code>wgt = name</code> (optional)	Name of series containing weights.
<code>wgtmethod = arg</code> (default = “sstdev”)	Weighting method (when weights are specified using “weight = ”): frequency (“freq”), inverse of variances (“var”), inverse of standard deviation (“stdev”), scaled inverse of variances (“svar”), scaled inverse of standard deviations (“sstdev”). Only applicable for ordinary (Pearson) calculations. Weights specified by “wgt = ” are frequency weights for rank correlation and Kendall’s tau calculations.
<code>df</code>	Compute covariances with a degree-of-freedom correction for the mean (for centered specifications), and any partial conditioning variables.
<code>outfmt = arg</code> (default = “single”)	Output format: single table (“single”), multiple table (“mult”), list (“list”), spreadsheet (“sheet”). Note that “outfmt = sheet” is only applicable if you specify a single statistic keyword.
<code>out = name</code>	Basename for saving output. All results will be saved in Sym matrices named using keys (“COV”, “CORR”, “SSCP”, “TAUA”, “TAUB”, “CONC” (Kendall’s concurrences), “DISC” (Kendall’s discordances), “CASES”, “OBS”, “WGTS”) appended to the basename (<i>e.g.</i> , the covariance specified by “out = my” is saved in the Sym matrix “MYCOV”).
<code>p</code>	Print the result.

Examples

```
vec1.cov corr stat prob
```

displays a table containing the Pearson correlation, t -statistic for testing for zero correlation, and associated p -value, for the vector VEC1.

```
vec1.cov taub taustat tauprob
```

computes the Kendall's tau-b, score statistic, and p -value for the score statistic.

```
vec1.cov(out=aa) cor
```

computes the Pearson correlation and saves the results in the symmetric matrix object AACORR.

Cross-references

See also [cor](#) (p. 576), [@cor](#) (p. 847), and [@cov](#) (p. 847).

displayname	Vector Procs
-------------	------------------------------

Set display name for vector.

Attaches a display name to a vector which may be used to label output in tables and graphs in place of the standard vector name.

Syntax

```
vector_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

```
v1.displayname Coef Results
v1.label
```

The first line attaches a display name “Coef Results” to the vector V1, and the second line displays the label view of V1, including its display name.

```
v1.displayname Means by State
plot v1
```

The first line attaches a display name “Means by State” to the vector V1. The line graph view of V1 will use the display name as the legend.

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels and display names.

See also [Vector::label](#) (p. 580) and [Graph::legend](#) (p. 161).

fill	Vector Procs
------	--------------

Fill a vector with the specified values.

Syntax

```
vector_name.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.*

Running out of values before the object is completely filled is not an error; the remaining cells or observations will be unaffected, unless the “1” (loop) option is specified. If, however, you list more values than the vector can hold, EViews will not modify any observations and will return an error message.

Options

1	Loop repeatedly over the list of values as many times as it takes to fill the vector.
o = integer (default = 1)	Fill the vector from the specified element. Default is the first element.

Examples

The following example declares a four element vector MC, initially filled with zeros. The second line fills MC with the specified values and the third line replaces from row 3 to the last row with -1.

```
vector(4) mc
mc.fill 0.1, 0.2, 0.5, 0.5
mc.fill(o=3, 1) -1
```

Cross-references

See [Chapter 18. “Matrix Language,” on page 627](#) of the *User’s Guide I* for a detailed discussion of vector and matrix manipulation in EViews.

label	Vector Views Vector Procs
-------	-----------------------------

Display or change the label view of the vector, including the last modified date and display name (if any).

Used as a procedure, `label` changes the fields in the vector label.

Syntax

```
vector_name.label
vector_name.label(options) [text]
```

Options

The first version of the command displays the label view of the vector. The second version may be used to modify the label. 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 <i>text</i> .
s	Sets the source field to <i>text</i> .
u	Sets the units field to <i>text</i> .
r	Appends <i>text</i> to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of LWAGE with “Data from CPS 1988 March File”:

```
lwage.label(r)
lwage.label(r) Data from CPS 1988 March File
```

To append additional remarks to LWAGE, and then to print the label view:

```
lwage.label(r) Log of hourly wage
lwage.label(p)
```

To clear and then set the units field, use:

```
lwage.label(u) Millions of bushels
```

Cross-references

See [“Labeling Objects” on page 72](#) of the *User’s Guide I* for a discussion of labels. See also [Vector::displayname](#) (p. 579).

read	Vector Procs
------	------------------------------

Import data from a foreign disk file into a vector.

May be used to import data into an existing workfile from a text, Excel, or Lotus file on disk.

Syntax

`vector_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. 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.

Options

File type options

<code>t = dat, txt</code>	ASCII (plain text) files.
<code>t = wk1, wk3</code>	Lotus spreadsheet files.
<code>t = xls</code>	Excel spreadsheet files.

If you do not specify the “t” option, EViews uses the file name extension to determine the file type. If you specify the “t” option, the file name extension will not be used to determine the file type.

Options for ASCII text files

<code>na = text</code>	Specify text for NAs. Default is “NA”.
<code>d = t</code>	Treat tab as delimiter (note: you may specify multiple delimiter options). The <i>default</i> is “d = c” only.
<code>d = c</code>	Treat comma as delimiter.
<code>d = s</code>	Treat space as delimiter.
<code>d = a</code>	Treat alpha numeric characters as delimiter.
<code>custom = symbol</code>	Specify symbol/character to treat as delimiter.
<code>mult</code>	Treat multiple delimiters as one.
<code>rect (default) / norect</code>	[Treat / Do not treat] file layout as rectangular.
<code>skipcol = integer</code>	Number of columns to skip. Must be used with the “rect” option.
<code>skiprow = integer</code>	Number of rows to skip. Must be used with the “rect” option.
<code>comment = symbol</code>	Specify character/symbol to treat as comment sign. Everything to the right of the comment sign is ignored. Must be used with the “rect” option.
<code>singlequote</code>	Strings are in single quotes, not double quotes.

dropstrings	Do not treat strings as NA; simply drop them.
negparen	Treat numbers in parentheses as negative numbers.
allowcomma	Allow commas in numbers (note that using commas as a delimiter takes precedence over this option).

Options for spreadsheet (Lotus, Excel) files

letter_number (default = "b2")	Coordinate of the upper-left cell containing data.
s = sheet_name	Sheet name for Excel 5–8 Workbooks.

Examples

```
v1.read(t=dat,na=.) a:\mydat.raw
```

reads data into vector V1 from an ASCII file MYDAT.RAW in the A: drive. The missing value NA is coded as a "." (dot or period).

```
v1.read(s=sheet2) "\\network\dr 1\cps91.xls"
```

reads the Excel file CPS91 into vector V1 from the network drive specified in the path.

Cross-references

See [“Importing Data” on page 95](#) of the *User’s Guide I* for a discussion and examples of importing data from external files.

For powerful, easy-to-use tools for reading data into a new workfile, see [“Creating a Workfile by Reading from a Foreign Data Source” on page 41](#) of the *User’s Guide I*, [pageload](#) (p. 750), and [wfoopen](#) (p. 802).

See also [Vector::write](#) (p. 588).

setformat	Vector Procs
-----------	------------------------------

Set the display format for cells in a vector spreadsheet view.

Syntax

```
vector_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.

For vectors, `setformat` operates on all of the cells in the vector.

You should use one of the following format specifications:

<code>g[.precision]</code>	significant digits
<code>f[.precision]</code>	fixed decimal places
<code>c[.precision]</code>	fixed characters
<code>e[.precision]</code>	scientific/float
<code>p[.precision]</code>	percentage
<code>r[.precision]</code>	fraction

In order to set a format that groups digits into thousands, place a “t” after a specified format. For example, to obtain a fixed number of decimal places, with commas used to separate thousands, use “ft[.precision]”. To obtain a fixed number of characters with a period used to separate thousands, use “ct[.precision]”.

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)”).

Examples

To set the format for all cells in the vector to fixed 5-digit precision, simply provide the format specification:

```
v1.setformat f.5
```

Other format specifications include:

```
v1.setformat f(.7)
v1.setformat e.5
```

Cross-references

See [Vector::setWidth \(p. 586\)](#), [Vector::setindent \(p. 584\)](#) and [Vector::setjust \(p. 585\)](#) for details on setting spreadsheet widths, indentation and justification.

setindent	Vector Procs
------------------	------------------------------

Set the display indentation for cells in vector spreadsheet views.

Syntax

```
view_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 spreadsheet (the EViews spreadsheet default font at the time the spreadsheet 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 spreadsheet view is created.

Examples

```
v1.setindent 2
```

sets the indentation for the vector spreadsheet view to 2.

Cross-references

See [Vector::setWidth \(p. 586\)](#) and [Vector::setjust \(p. 585\)](#) for details on setting spreadsheet widths and justification.

setjust	Vector Procs
---------	------------------------------

Set the display justification for cells in a vector spreadsheet view.

Syntax

```
vector_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 *format_arg* may be formed using the following:

top / middle / bottom]	Vertical justification setting.
auto / left / cen- ter / 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 taken from the Global Defaults for spreadsheet views.

Examples

```
v1.setjust middle
```

sets the vertical justification to the middle.

```
v1.setjust top left
```

sets the vertical justification to top and the horizontal justification to left.

Cross-references

See [Vector::setWidth \(p. 586\)](#) and [Vector::setindent \(p. 584\)](#) for details on setting spreadsheet widths and indentation.

setwidth	Vector Procs
----------	------------------------------

Set the column width in a vector spreadsheet view.

Syntax

`vector_name.setwidth width_arg`

where *width_arg* specifies the width unit value. The width unit is computed from representative characters in the default font for the current spreadsheet (the EViews spreadsheet default font at the time the spreadsheet 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

```
v1.setwidth 12
```

sets the width of the vector to 12 width units.

Cross-references

See [Vector::setindent](#) (p. 584) and [Vector::setjust](#) (p. 585) for details on setting spreadsheet indentation and justification.

sheet	Vector Views
-------	------------------------------

Spreadsheet view of vector object.

Syntax

`vector_name.sheet(options)`

Options

p	Print the spreadsheet view.
---	-----------------------------

Examples

```
v1.sheet(p)
```

displays and prints the spreadsheet view of vector V1.

stats	Vector Views
--------------	------------------------------

Descriptive statistics for the vector.

Computes and displays a table of means, medians, maximum and minimum values, standard deviations, and other descriptive statistics for the data in the vector object.

Syntax

```
vector_name.stats(options)
```

Options

p	Print the stats table.
---	------------------------

Examples

```
v1.stats(p)
```

displays and prints the descriptive statistics view of the vector V1.

Cross-references

See [“Descriptive Statistics & Tests” on page 306](#) and [page 379](#) of the *User’s Guide I* for a discussion of the descriptive statistics views of series and groups.

vector	Vector Declaration
---------------	------------------------------------

Declare a vector object.

The `vector` command declares and optionally initializes a (column) vector object.

Syntax

```
vector(size) vector_name [= assignment]
```

The keyword `vector` 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. If you do not provide a size, EViews will create a single element 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

```
vector vec1
vector(10) col3 = 3
```

```
rowvector(10) row3 = 3
vector vec3 = row3
```

VEC1 is declared as a single element vector initialized to 0. COL3 is a 10 element column vector containing the value 3. ROW3 is declared as a row vector of size 10 containing the value 3. 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 [Chapter 18. “Matrix Language,” on page 627](#) of the *User’s Guide I* for a discussion of matrices and vectors in EViews.

See also [Coef::coef \(p. 16\)](#) and [Rowvector::rowvector \(p. 343\)](#).

write	Vector Procs
-------	------------------------------

Write EViews data to a text (ASCII), Excel, or Lotus file on disk.

Creates a foreign format disk file containing data in a vector object. May be used to export EViews data to another program.

Syntax

`vector_name.write(options) [path\filename]`

Follow the name of the vector object by a period, the keyword, and the name for the output file. 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. The entire vector will be exported.

Note that EViews cannot, at present, write into an existing file. The file that you select will, if it exists, 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

<code>na = string</code>	Specify text string for NAs. Default is “NA”.
<code>d = arg</code>	Specify delimiter (<i>default</i> is tab): “s” (space), “c” (comma).

Spreadsheet (Lotus, Excel) files

<code>letter_number</code>	Coordinate of the upper-left cell containing data.
----------------------------	--

Examples

```
v1.write(t=txt,na=.) a:\dat1.csv
```

Writes the vector V1 into an ASCII file named DAT1.CSV on the A: drive. NAs are coded as “.” (dot).

```
v1.write(t=txt,na=.) dat1.csv
```

writes the same file in the default directory.

```
v1.write(t=xls) "\\network\drive a\results"
```

saves the contents of V1 in an Excel file “Results.xls” in the specified directory.

Cross-references

See [“Exporting to a Spreadsheet or Text File” on page 105](#) of the *User’s Guide I* for a discussion. See also [Vector::read \(p. 581\)](#).

Chapter 2. Object Summary for Commands

This chapter contains an alphabetical listing of the object commands, pairing each entry with a list of the EViews objects with which it may be used.

Object Summary

3slssystem (p. 475).
addgroup (p. 183), pool (p. 295).
addassignmodel (p. 271).
addinitmodel (p. 271).
addtextgraph (p. 143).
aligngraph (p. 143).
alphaalpha (p. 5).
anticovfactor (p. 97).
appendlogl (p. 233), model (p. 271), spool (p. 409), sspace (p. 427), system (p. 475), valmap (p. 537), var (p. 543).
archequation (p. 27), system (p. 475).
archtestequation (p. 27).
areacoef (p. 15), group (p. 183), matrix (p. 247), series (p. 355), sym (p. 453), vector (p. 575).
arlmvar (p. 543).
armaequation (p. 27).
arrootsvar (p. 543).
autoequation (p. 27).
axisgraph (p. 143).
bandgroup (p. 183), matrix (p. 247), sym (p. 453).
barcoef (p. 15), group (p. 183), matrix (p. 247), rowvector (p. 337), series (p. 355), sym (p. 453), vector (p. 575).
bdstestseries (p. 355).
binaryequation (p. 27).
blockmodel (p. 271).
boxplotcoef (p. 15), group (p. 183), matrix (p. 247), rowvector (p. 337), series (p. 355), sym (p. 453), vector (p. 575).
bpfseries (p. 355).
causegroup (p. 183).
cellipseequation (p. 27), logl (p. 233), pool (p. 295), sspace (p. 427), system (p. 475).

censored..... equation (p. 27).
checkderivs..... logl (p. 233).
chow equation (p. 27).
classify series (p. 355).
cleartext var (p. 543).
coef..... coef (p. 15).
coefcov..... equation (p. 27), logl (p. 233), pool (p. 295), sspace (p. 427), system (p. 475).
coint..... group (p. 183), pool (p. 295), var (p. 543).
comment spool (p. 409), table (p. 509).
control..... model (p. 271).
cor group (p. 183), matrix (p. 247), sym (p. 453), vector (p. 575).
correl..... equation (p. 27), group (p. 183), series (p. 355), var (p. 543).
correlsq equation (p. 27).
count..... equation (p. 27).
cov group (p. 183), matrix (p. 247), sym (p. 453), vector (p. 575).
cross group (p. 183).
datelabel..... graph (p. 143).
decomp var (p. 543).
define pool (p. 295).
delete pool (p. 295).
deletecol..... table (p. 509).
deleterow..... table (p. 509).
derivs equation (p. 27), system (p. 475).
describe..... pool (p. 295).
display spool (p. 409).
displayname alpha (p. 5), coef (p. 15), equation (p. 27), factor (p. 97), graph (p. 143), group (p. 183), link (p. 223), logl (p. 233), matrix (p. 247), model (p. 271), pool (p. 295), rowvector (p. 337), sample (p. 349), series (p. 355), spool (p. 409), sspace (p. 427), sym (p. 453), system (p. 475), table (p. 509), text (p. 533), valmap (p. 537), var (p. 543), vector (p. 575).
distplot..... coef (p. 15), group (p. 183), matrix (p. 247), rowvector (p. 337), series (p. 355), sym (p. 453), vector (p. 575).
dot coef (p. 15), group (p. 183), matrix (p. 247), rowvector (p. 337), series (p. 355), sym (p. 453), vector (p. 575).
draw..... graph (p. 143).
drawdefault graph (p. 143).
drop group (p. 183), pool (p. 295).

dtable.....group (p. 183).
ecvar (p. 543).
edfctest.....series (p. 355).
eigen.....factor (p. 97), sym (p. 453).
endog.....sspace (p. 427), system (p. 475), var (p. 543).
eqs.....model (p. 271).
equation.....equation (p. 27).
errbar.....group (p. 183), matrix (p. 247), rowvector (p. 337), sym (p. 453).
excludemodel (p. 271).
extract.....spool (p. 409).
facbreakequation (p. 27).
factnames.....factor (p. 97).
factorfactor (p. 97).
fetchpool (p. 295).
fill.....coef (p. 15), matrix (p. 247), rowvector (p. 337), series (p. 355),
sym (p. 453), vector (p. 575).
fiml.....system (p. 475).
fit.....equation (p. 27).
fitstatsfactor (p. 97).
fittedfactor (p. 97).
fixedtestequation (p. 27), pool (p. 295).
flattenspool (p. 409).
forecastequation (p. 27), sspace (p. 427).
freeze.....graph (p. 143), table (p. 509).
freq.....alpha (p. 5), group (p. 183), series (p. 355).
frmlalpha (p. 5), series (p. 355).
garch.....equation (p. 27), system (p. 475).
genralpha (p. 5), pool (p. 295), series (p. 355).
glslfactor (p. 97).
gmmequation (p. 27), system (p. 475).
grads.....equation (p. 27), logl (p. 233), sspace (p. 427), system (p. 475).
graphgraph (p. 143).
graphmodespool (p. 409).
groupgroup (p. 183).
hettestequation (p. 27).
hilogroup (p. 183), matrix (p. 247), sym (p. 453).
histequation (p. 27), series (p. 355).
horizindent.....spool (p. 409).
hpf.....series (p. 355).

impulse var (p. 543).
innov..... model (p. 271).
insert..... spool (p. 409).
insertcol table (p. 509).
insertrow table (p. 509).
ipf factor (p. 97).
jbera..... system (p. 475), var (p. 543).
label..... alpha (p. 5), coef (p. 15), equation (p. 27), factor (p. 97), graph
(p. 143), group (p. 183), link (p. 223), logl (p. 233), matrix
(p. 247), model (p. 271), pool (p. 295), rowvector (p. 337), sample
(p. 349), series (p. 355), spool (p. 409), sspace (p. 427), sym
(p. 453), system (p. 475), table (p. 509), text (p. 533), valmap
(p. 537), var (p. 543), vector (p. 575).
laglen var (p. 543).
leftmargin..... spool (p. 409).
legend graph (p. 143).
line coef (p. 15), group (p. 183), matrix (p. 247), series (p. 355), sym
(p. 453), vector (p. 575).
link link (p. 223).
linkto link (p. 223).
loadings..... factor (p. 97).
logit equation (p. 27).
logl..... logl (p. 233).
ls..... equation (p. 27), pool (p. 295), system (p. 475), var (p. 543).
makecoint..... var (p. 543).
makederivs equation (p. 27).
makeendog sspace (p. 427), system (p. 475), var (p. 543).
makefilter sspace (p. 427).
makegarch equation (p. 27), system (p. 475).
makegrads equation (p. 27), logl (p. 233), sspace (p. 427).
makegraph..... model (p. 271).
makegroup model (p. 271), pool (p. 295).
makelimits equation (p. 27).
makeloglike system (p. 475).
makemap..... alpha (p. 5).
makemodel..... equation (p. 27), logl (p. 233), pool (p. 295), sspace (p. 427), sys-
tem (p. 475), var (p. 543).
makepcomp group (p. 183).
makeregs equation (p. 27).

makesresidsequation (p. 27), pool (p. 295), system (p. 475), var (p. 543).
makescores.....factor (p. 97).
makesignals.....sspace (p. 427).
makestatessspace (p. 427).
makestatspool (p. 295).
makesystem.....pool (p. 295), var (p. 543).
mapalpha (p. 5), series (p. 355).
matrixmatrix (p. 247).
maxcorfactor (p. 97).
meansequation (p. 27).
merge.....graph (p. 143), model (p. 271).
mlfactor (p. 97), logl (p. 233), sspace (p. 427).
modelmodel (p. 271).
move.....spool (p. 409).
msa.....factor (p. 97).
msg.....model (p. 271).
namegraph (p. 143), spool (p. 409).
observedfactor (p. 97).
options.....graph (p. 143), spool (p. 409).
orderedequation (p. 27).
outputequation (p. 27), factor (p. 97), logl (p. 233), pool (p. 295), sspace
(p. 427), system (p. 475), var (p. 543).
override.....model (p. 271).
pace.....factor (p. 97).
partcorfactor (p. 97).
pcomp.....group (p. 183), matrix (p. 247).
pf.....factor (p. 97).
piegroup (p. 183), matrix (p. 247), rowvector (p. 337), sym (p. 453).
poolpool (p. 295).
predictequation (p. 27).
print.....spool (p. 409).
probit.....equation (p. 27).
qqplotcoef (p. 15), group (p. 183), matrix (p. 247), rowvector (p. 337),
series (p. 355), sym (p. 453), vector (p. 575).
qregequation (p. 27).
qrprocess.....equation (p. 27).
qrslope.....equation (p. 27).
qrsymm.....equation (p. 27).
qstatssystem (p. 475), var (p. 543).

ranhaus equation (p. 27), pool (p. 295).
read..... coef (p. 15), matrix (p. 247), pool (p. 295), rowvector (p. 337), sym
 (p. 453), vector (p. 575).
reduced factor (p. 97).
remove spool (p. 409).
representations equation (p. 27), pool (p. 295), var (p. 543).
resample..... group (p. 183), series (p. 355).
reset equation (p. 27).
residcor pool (p. 295), sspace (p. 427), system (p. 475), var (p. 543).
residcov..... pool (p. 295), sspace (p. 427), system (p. 475), var (p. 543).
resids equation (p. 27), factor (p. 97), pool (p. 295), sspace (p. 427), sys-
 tem (p. 475), var (p. 543).
results equation (p. 27), logl (p. 233), pool (p. 295), sspace (p. 427), sys-
 tem (p. 475), var (p. 543).
rls equation (p. 27).
rotate factor (p. 97).
rotatedclear factor (p. 97).
rotateout..... factor (p. 97).
rowvector rowvector (p. 337).
sample sample (p. 349).
save graph (p. 143), table (p. 509).
scalar scalar (p. 353).
scat group (p. 183), matrix (p. 247), rowvector (p. 337), sym (p. 453).
scatmat..... group (p. 183), matrix (p. 247), rowvector (p. 337), sym (p. 453).
scatpair group (p. 183), matrix (p. 247), rowvector (p. 337), sym (p. 453).
scenario..... model (p. 271).
scores..... factor (p. 97).
seas..... series (p. 355).
seasplot coef (p. 15), group (p. 183), matrix (p. 247), rowvector (p. 337),
 series (p. 355), sym (p. 453), vector (p. 575).
series..... series (p. 355).
set..... sample (p. 349).
setbpelem graph (p. 143).
setcell..... table (p. 509).
setcolwidth..... table (p. 509).
setconvert..... series (p. 355).
setelem..... graph (p. 143).
setfillcolor..... table (p. 509).
setfont..... table (p. 509).

setformatcoef (p. 15), group (p. 183), matrix (p. 247), rowvector (p. 337),
series (p. 355), sym (p. 453), table (p. 509), vector (p. 575).

setheighttable (p. 509).

setindentalpha (p. 5), coef (p. 15), group (p. 183), matrix (p. 247), rowvec-
tor (p. 337), series (p. 355), sym (p. 453), table (p. 509), vector
(p. 575).

setjustalpha (p. 5), coef (p. 15), group (p. 183), matrix (p. 247), rowvec-
tor (p. 337), series (p. 355), sym (p. 453), table (p. 509), vector
(p. 575).

setlinetable (p. 509).

setlinestable (p. 509).

setmergetable (p. 509).

setobslabelgraph (p. 143).

settextcolortable (p. 509).

setwidthcoef (p. 15), group (p. 183), matrix (p. 247), rowvector (p. 337),
series (p. 355), sym (p. 453), table (p. 509), vector (p. 575).

sheetalpha (p. 5), coef (p. 15), group (p. 183), matrix (p. 247), pool
(p. 295), rowvector (p. 337), series (p. 355), sym (p. 453), table
(p. 509), valmap (p. 537), vector (p. 575).

signalgraphssspace (p. 427).

smcfactor (p. 97).

smoothseries (p. 355).

solvemodel (p. 271).

solveoptmodel (p. 271).

sortgraph (p. 143), group (p. 183), series (p. 355).

speclogl (p. 233), model (p. 271), sspace (p. 427), system (p. 475).

spikecoef (p. 15), group (p. 183), matrix (p. 247), rowvector (p. 337),
series (p. 355), sym (p. 453), vector (p. 575).

spoolspool (p. 409).

sspacesspace (p. 427).

statbyseries (p. 355).

statefinalsspace (p. 427).

stategraphssspace (p. 427).

stateinitsspace (p. 427).

statscoef (p. 15), group (p. 183), matrix (p. 247), rowvector (p. 337),
series (p. 355), sym (p. 453), valmap (p. 537), vector (p. 575).

steplsequation (p. 27).

stochasticmodel (p. 271).

stomgroup (p. 183), series (p. 355).

stomna group (p. 183), series (p. 355).
store pool (p. 295).
structure factor (p. 97), sspace (p. 427).
sur system (p. 475).
svar var (p. 543).
sym sym (p. 453).
system system (p. 475).
table table (p. 509).
tablemode spool (p. 409).
template graph (p. 143).
testadd equation (p. 27), pool (p. 295).
testbtw group (p. 183).
testby series (p. 355).
testdrop equation (p. 27), pool (p. 295).
testexog var (p. 543).
testfit equation (p. 27).
testlags var (p. 543).
teststat series (p. 355).
text model (p. 271), text (p. 533).
textdefault graph (p. 143).
title table (p. 509).
topmargin spool (p. 409).
trace model (p. 271).
track model (p. 271).
tramoseats series (p. 355).
tsls equation (p. 27), pool (p. 295), system (p. 475).
ubreak equation (p. 27).
uls factor (p. 97).
unlink model (p. 271).
update model (p. 271).
updatecoefs equation (p. 27), logl (p. 233), pool (p. 295), sspace (p. 427), system (p. 475).
uroot group (p. 183), pool (p. 295), series (p. 355).
usage valmap (p. 537).
valmap valmap (p. 537).
var var (p. 543).
vars model (p. 271).
vector vector (p. 575).
vertindent spool (p. 409).

vertspacing.....spool (p. 409).
waldequation (p. 27), logl (p. 233), pool (p. 295), sspace (p. 427), system (p. 475).
whiteequation (p. 27), var (p. 543).
widthspool (p. 409).
wls.....system (p. 475).
writecoef (p. 15), matrix (p. 247), pool (p. 295), rowvector (p. 337), sym (p. 453), vector (p. 575).
wtslssystem (p. 475).
x11series (p. 355).
x12series (p. 355).
xyareagroup (p. 183), matrix (p. 247), sym (p. 453).
xybargroup (p. 183), matrix (p. 247), rowvector (p. 337), sym (p. 453).
xyline.....group (p. 183), matrix (p. 247), sym (p. 453).
xypairgroup (p. 183), matrix (p. 247), rowvector (p. 337), sym (p. 453).

Chapter 3. Graph Creation Commands

This chapter contains reference material for commands that display graph views of various EViews data objects. The chapter differs in structure from the earlier object reference ([Chapter 1. “Object View and Procedure Reference,” on page 3](#)) in that it describes the ways in which the commands may be used with multiple objects. For details on commands to customize existing graphs, see the the graph object reference: [“Graph” on page 143](#).

The remainder of the chapter consists of alphabetical listings of the graph view commands in three distinct formats:

- the first listing provides a basic summary of the available graph commands, with a reference to the detailed description for that command.
- the second listing repeats the summary of graph commands, pairing each entry with a list of the EViews objects with which it may be used.
- the third listing, which constitutes the main portion of this chapter, consists of a detailed description of each graph command, including basic syntax and options, as well as examples and cross-references.

Graph Creation Command Summary

The following view commands may be used to display graphs of various EViews data objects:

[area](#) area graph ([p. 603](#)).
[band](#) area band graph ([p. 606](#)).
[bar](#) bar graph ([p. 609](#)).
[boxplot](#) boxplot graph ([p. 613](#)).
[distplot](#) distribution graph ([p. 615](#)).
[dot](#) dot plot graph ([p. 622](#)).
[errbar](#) error bar graph ([p. 626](#)).
[hilo](#) high-low(-open-close) graph ([p. 628](#)).
[line](#) line-symbol graph ([p. 630](#)).
[pie](#) pie chart ([p. 633](#)).
[qqplot](#) quantile-quantile graph ([p. 636](#)).
[scat](#) scatterplot ([p. 640](#)).
[scatmat](#) matrix of scatterplots ([p. 644](#)).
[scatpair](#) scatterplot pairs graph ([p. 647](#)).
[seasplot](#) seasonal line graph ([p. 651](#)).
[spike](#) spike graph ([p. 652](#)).
[xyarea](#) XY area graph ([p. 656](#)).

xybar XY bar graph (p. 659).
xyline..... XY line graph (p. 661).
xypair XY line pairs graph (p. 664).

Graph Creation Object Summary

The graph creation commands may be used with the following EViews data objects:

area..... coef (p. 15), group (p. 183), matrix (p. 247), series (p. 355), sym (p. 453), vector (p. 575).
band group (p. 183), matrix (p. 247), sym (p. 453).
bar coef (p. 15), group (p. 183), matrix (p. 247), rowvector (p. 337), series (p. 355), sym (p. 453), vector (p. 575).
boxplot..... coef (p. 15), group (p. 183), matrix (p. 247), rowvector (p. 337), series (p. 355), sym (p. 453), vector (p. 575).
distplot..... coef (p. 15), group (p. 183), matrix (p. 247), rowvector (p. 337), series (p. 355), sym (p. 453), vector (p. 575).
dot coef (p. 15), group (p. 183), matrix (p. 247), rowvector (p. 337), series (p. 355), sym (p. 453), vector (p. 575).
errbar..... group (p. 183), matrix (p. 247), rowvector (p. 337), sym (p. 453).
hilo group (p. 183), matrix (p. 247), sym (p. 453).
line coef (p. 15), group (p. 183), matrix (p. 247), series (p. 355), sym (p. 453), vector (p. 575).
pie group (p. 183), matrix (p. 247), rowvector (p. 337), sym (p. 453).
qqplot coef (p. 15), group (p. 183), matrix (p. 247), rowvector (p. 337), series (p. 355), sym (p. 453), vector (p. 575).
scat group (p. 183), matrix (p. 247), rowvector (p. 337), sym (p. 453).
scatmat group (p. 183), matrix (p. 247), rowvector (p. 337), sym (p. 453).
scatpair group (p. 183), matrix (p. 247), rowvector (p. 337), sym (p. 453).
seasplot..... coef (p. 15), group (p. 183), matrix (p. 247), rowvector (p. 337), series (p. 355), sym (p. 453), vector (p. 575).
spike coef (p. 15), group (p. 183), matrix (p. 247), rowvector (p. 337), series (p. 355), sym (p. 453), vector (p. 575).
xyarea group (p. 183), matrix (p. 247), sym (p. 453).
xybar group (p. 183), matrix (p. 247), rowvector (p. 337), sym (p. 453).
xyline..... group (p. 183), matrix (p. 247), sym (p. 453).
xypair group (p. 183), matrix (p. 247), rowvector (p. 337), sym (p. 453).

Graph Creation Entries

The following section provides an alphabetical listing of the graph creation commands. Each entry outlines the command syntax and associated options, and includes examples and cross references.

area	Command Coef View Graph Command Group View Matrix View Series View Sym View Vector View
------	---

Display an area graph view.

Syntax

```
area(options) o1 [o2 o3 ...]
```

```
object_name.area(options) [categorical_spec(arg)]
```

where *o1*, *o2*, ..., are series or group objects. Following the `area` keyword, you may specify general graph characteristics using *options*. Available options include multiple graph handling, dual scaling, template application, data contraction, adding axis extensions, and rotation.

The optional *categorical_spec* allows you to specify a categorical graph (see “[Categorical Spec](#),” on page 668).

Options

Scale options

<code>a</code> (<i>default</i>)	Automatic single scale.
<code>d</code>	Dual scaling with no crossing. The first series or column is scaled on the left and all other series or columns are scaled on the right.
<code>x</code>	Dual scaling with possible crossing. See the “d” option.
<code>n</code>	Normalized scale (zero mean and unit standard deviation). May not be used with the “s” option.
<code>rotate</code>	Rotate the graph so the observation axis is on the left.
<code>ab = type</code>	Add axis border along data scale, where <i>type</i> may be “hist” or “h” (histogram), “boxplot” or “b”, “kernel” or “k”. (Note: axis borders are not available for panel graphs with “panel = ” options that involve summaries: mean, median, etc.)

Multiple series options (categorical graph settings will override these options)

m	Plot areas in multiple graphs (will override 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 or column listed and a line graph for all subsequent series or columns. May not be used with the “s” option.

Template and printing options

o = <i>template</i>	Use appearance options from the specified template. <i>template</i> may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
t = <i>graph_name</i>	Use appearance options and copy text and shading from the specified graph.
b / -b	[Apply / Remove] bold modifiers of the base template style specified using the “o = ” option above.
w / -w	[Apply / Remove] wide modifiers of the base template style specified using the “o = ” option above.
reset	Resets all graph options to the global defaults. May be used to remove existing customization of the graph.
p	Print the graph.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Graph data options

The following option is available in non-panel or categorical graph settings:

contract = <i>key</i>	Contract the data as specified by <i>key</i> , where <i>key</i> may be: “mean”, “median”, “max”, “min”, “sum”, “var” - variance, “sd” - standard deviation, “sumsq” - sum of the squared values, “skew” - skewness, “kurt” - kurtosis, “nas” - number of missing values, “obs” - number of observations, “unique” - error if the series is not identical for all observations in a given group, “first” - first observation in category using workfile order, “last” - last observation in category using workfile order, “quant(<i>quantile</i>)” - where <i>quantile</i> is a number between 0 and 1.
-----------------------	---

Panel options

The following option applies when graphing panel structured data:

<code>panel = arg</code> (default taken from global settings)	Panel data display: “stack” (stack the cross-sections), “individual” or “i” (separate graph for each cross-section), “combine” or “c” (combine cross-section graphs in a single frame), “mean” (plot means across cross-sections), “median” (plot median across cross-sections). (Note: more general versions of these panel graphs may be constructed as categorical graphs.)
--	---

Categorical graph options

These options only apply to categorical graphs (“[Categorical Spec](#),” on page 668) where the graph has one or more **within** factors and a contraction method other than raw data (see the **contract** option above).

<code>favorlegend</code>	Favor the use of legends over axis labels to describe categories.
<code>elemcommon = int</code>	Specifies the number of within factors for which the graph uses common area colors. For example, with multiple within dimensions, if “elemcommon = 1”, then only categories defined by the first within factor will have common colors. If “elemcommon = 2”, then categories defined by the first two within factors will have common colors. If “elemcommon = 0”, all areas will have different colors. The default is one less than the number of within factors.

Examples

Basic examples

```
area ser1 ser2 ser3
```

displays area graphs of SER1, SER2, and SER3.

```
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.

```
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.

```
g1.area(o=midnight, b, w)
```

creates an area graph of the group G1, using the settings of the predefined template “midnight,” applying the *bold* and *wide* modifiers.

Panel examples

```
ser1.area(panel=individual)
```

displays area graphs with a separate graph for each cross-section, while,

```
ser1.area(panel=mean)
```

displays an area graph of the means for each period computed across cross-sections.

Categorical spec examples

```
ser1.area across(firm, dispname)
```

displays a categorical area graph of SER1 using distinct values of FIRM to define the categories. The graphs in multiple frames with the display names used as labels.

```
ser1.area across(firm, dispname, iscale)
```

shows the same graph with individual scaling for each of the frames.

```
ser1.area within(firm, inctot)
```

displays a graph with the same categorization (along with a category for the total), but with all of the graphs in a single frame.

Cross-references

See [Chapter 13. “Graphing Data,” on page 415](#) of the *User’s Guide I* for a detailed discussion of graphs in EViews, and [“Templates” on page 536](#) of the *User’s Guide I* for a discussion of graph templates. See [Graph::graph \(p. 158\)](#) for graph declaration and other graph types.

band	Command Graph Command Group View Matrix View Sym View
-------------	---

Display an area band graph view (if possible).

An area band graph fills the area between pairs of series or columns of a matrix.

Syntax

```
band(options) o1 [o2 o3 ... ]  
object_name.band(options)
```

where *o1*, *o2*, ..., are series or group objects. Following the `band` keyword, you may specify general graph characteristics using *options*. Available options include axis settings and template application.

Options

Scale options

a (<i>default</i>)	Automatic single scale.
d	Dual scaling with no crossing. The first series or column is scaled on the left and all other series or columns are scaled on the right.
x	Dual scaling with possible crossing. See the “d” option.
n	Normalized scale (zero mean and unit standard deviation).
rotate	Rotate the graph so the observation axis is on the left.

Multiple series pair options

By default, EViews displays band graphs for series or columns in *object_name* taken in pairs, with the remainder series or column (if any) displayed as a line graph. You may modify this behavior using the “l” option:

l	Display band graph for the first pair of series or columns and a line graph for all subsequent series or columns.
---	---

Template and printing options

o = <i>template</i>	Use appearance options from the specified template. <i>template</i> may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
t = <i>graph_name</i>	Use appearance options and copy text and shading from the specified graph.
b / -b	[Apply / Remove] bold modifiers of the base template style specified using the “o = ” option above.
w / -w	[Apply / Remove] wide modifiers of the base template style specified using the “o = ” option above.
reset	Resets all graph options to the global defaults. May be used to remove existing customization of the graph.
p	Print the graph.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Panel options

The following option applies when graphing panel structured data:

panel = <i>arg</i> (default taken from global set- tings)	Panel data display: “stack” (stack the cross-sections), “individual” or “i” (separate graph for each cross-section), “combine” or “c” (combine cross-section graphs in a single frame), “mean” (plot means across cross-sections), “median” (plot median across cross-sections). (Note: more general versions of these panel graphs may be constructed as categorical graphs.)
--	---

Examples

Basic examples

```
band upper1 lower1
```

displays a band graph using UPPER1 and LOWER1.

```
group g1 upper1 lower1 upper2 lower2  
g1.band
```

plots a band graph with the UPPER1 and LOWER1 defining one band, and UPPER2 and LOWER2 defining as second band, both displayed in the same frame.

```
g1.band(o=midnight, 1)
```

plots the band graph defined by UPPER1 and LOWER1 along with line graphs for UPPER2 and LOWER2, using the settings of the predefined template “midnight.”

Panel examples

```
g1.band
```

shows the band graph for the stacked data in a panel workfile.

```
g1.band(panel=individual)
```

displays band graphs for each cross-section in separate frames, while,

```
g1.band(panel=mean)
```

constructs a band graph using the means for each period computed across cross-sections.

Cross-references

See [Chapter 13. “Graphing Data,” on page 415](#) of the *User’s Guide I* for a detailed discussion of graphs in EViews, and [“Templates” on page 536](#) of the *User’s Guide I* for a discussion of graph templates. See [Graph::graph \(p. 158\)](#) for graph declaration and other graph types.

bar	Command Coef View Graph Command Group View Matrix View Rowvector View Series View Sym View Vector View
-----	--

Display a bar graph.

(Note: when the individual bars in a bar graph become too thin to be distinguished, the graph will automatically be converted into an area graph; see [area](#) (p. 603).)

Syntax

```
bar(options) o1 [o2 o3 ... ]
object_name.bar(options) [categorical_spec(arg)]
```

where *o1*, *o2*, ..., are series or group objects. Following the `bar` keyword, you may specify general graph characteristics using *options*. Available options include multiple graph handling, dual scaling, template application, data contraction, adding axis extensions, and rotation.

The optional *categorical_spec* allows you to specify a categorical graph (see “[Categorical Spec](#),” on page 668).

Options

Scale options

a (default)	Automatic single scale.
d	Dual scaling with no crossing. The first series or column is scaled on the left and all other series or columns 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.
rotate	Rotate the graph so the observation axis is on the left.
ab = type	Add axis border along data scale, where <i>type</i> may be “hist” or “h” (histogram), “boxplot” or “b”, “kernel” or “k”. (Note: axis borders are not available for panel graphs with “panel = ” options that involve summaries: mean, median, etc.)

Multiple series options (categorical graph settings will override these options)

m	Plot bars in multiple graphs (will override the “s” option).
s	Stacked bar graph. Each bar represents the cumulative total of the series or columns listed. The difference between bars corresponds to the value of a series or column. May not be used with the “l” option.
l	Bar graph for the first series or column and a line graph for all subsequent series or columns. May not be used with the “s” option.

Template and printing options

o = <i>template</i>	Use appearance options from the specified template. <i>template</i> may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
t = <i>graph_name</i>	Use appearance options and copy text and shading from the specified graph.
b / -b	[Apply / Remove] bold modifiers of the base template style specified using the “o = ” option above.
w / -w	[Apply / Remove] wide modifiers of the base template style specified using the “o = ” option above.
reset	Resets all graph options to the global defaults. May be used to remove existing customization of the graph.
p	Print the graph.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Graph data options

The following option is available in non-panel or categorical graph settings:

contract = <i>key</i>	Contract the data as specified by <i>key</i> , where <i>key</i> may be: “mean”, “median”, “max”, “min”, “sum”, “var” - variance, “sd” - standard deviation, “sumsq” - sum of the squared values, “skew” - skewness, “kurt” - kurtosis, “nas” - number of missing values, “obs” - number of observations, “unique” - error if the series is not identical for all observations in a given group, “first” - first observation in category using workfile order, “last” - last observation in category using workfile order, “quant(<i>quantile</i>)” - where <i>quantile</i> is a number between 0 and 1.
-----------------------	---

Panel options

The following option applies when graphing panel structured data:

<code>panel = arg</code> (default taken from global settings)	Panel data display: “stack” (stack the cross-sections), “individual” or “i” (separate graph for each cross-section), “combine” or “c” (combine cross-section graphs in a single frame), “mean” (plot means across cross-sections), “median” (plot median across cross-sections). (Note: more general versions of these panel graphs may be constructed as categorical graphs.)
--	---

Categorical graph options

These options only apply to categorical graphs (“[Categorical Spec](#),” on page 668) where the graph has one or more **within** factors and a contraction method other than raw data (see the **contract** option above).

<code>favorlegend</code>	Favor the use of legends over axis labels to describe categories.
<code>elemcommon = int</code>	Specifies the number of within factors for which the graph uses common area colors. For example, with multiple within dimensions, if “elemcommon = 1”, then only categories defined by the first within factor will have common colors. If “elemcommon = 2”, then categories defined by the first two within factors will have common colors. If “elemcommon = 0”, all areas will have different colors. The default is one less than the number of within factors.

Examples

Basic examples

```
bar(p, rotate) oldsales newsales
```

displays and prints a rotated bar graph of the series OLDSALES and NEWSALES.

```
pop.bar
```

displays a bar graph of the series POP.

```
group mygrp oldsales newsales
mygrp.bar(s)
```

displays a stacked bar graph view of the series in the group MYGRP.

```
mygrp.bar(l, x, o=mybar1)
```

plots a bar graph of OLDSALES together with a line graph of NEWSALES. The bar graph is scaled on the left, while the line graph is scaled on the right. The graph uses options from graph MYBAR1 as a template.

```
mygrp.bar(o=midnight, b)
```

creates a bar graph of MYGRP, using the settings of the predefined template “midnight,” applying the *bold* modifier.

```
mygrp.bar(rotate, contract=mean)
```

displays a rotated bar graph of the means of OLDSALES and NEWSALES.

Panel examples

```
ser1.bar(panel=individual)
```

displays bar graphs for each cross-section in a separate frame, while,

```
ser1.bar(panel=median)
```

displays a bar graph of the medians of SER1 computed for each period across cross-sections.

Categorical spec examples

```
ser1.bar across(firm, dispname)
```

displays a categorical bar graph of SER1 using distinct values of FIRM to define the categories, and displaying the resulting graphs in multiple frames.

```
ser1.bar across(firm, dispname, iscale)
```

shows the same graph with individual scaling for each of the frames.

```
ser1.bar within(contract=mean, firm, inctot, label=value)
```

displays a graph of mean values of SER1 categorized by firm (along with an added category for the total), with all of the graphs in a single frame and the FIRM category value used as labels.

```
ser1.bar(contract=sum) across(firm, dispname) within(income, bin-  
type=quant, bincount=4)
```

constructs a categorical bar graph of the sum of SER1 values within a category. Different firms are displayed in different graph frames, using the display name as labels, with each frame containing bars depicting the sum of SER1 for each income quartiles.

```
ser1.bar(contract=mean, elemcommon=1) within(sex) within(union)
```

creates a bar graph of mean values of within categories based on both SEX and UNION. Categories for the distinct elements of UNION will be depicted using different bar colors, with the color assignment repeated for different values of SEX.

```
group mygrp oldsales newsales
```

```
mygrp.bar(contract=min) within(@series) within(age)
```

displays bar graphs of the minimum values for categories defined by distinct values of AGE (and the two series). All of the bars will be displayed in a single frame with the bars for OLDSALES grouped together followed by the bars for NEWSALES.


```
mygrp.bar(contract=median, elemcommon=2) across(firm)
  across(@series) across(age)
```

also adds an additional categorization using the FIRM identifiers. The observations for a given firm are grouped together. Within a firm, the bars for the OLDSALES and NEWSALES, which will be depicted using different colors, will be grouped within each age category. The color assignment to OLDSALES and NEWSALES will be repeated across firms and ages (note that @SERIES is treated as the last across factor).

Cross-references

See [Chapter 13. “Graphing Data,” on page 415](#) of the *User’s Guide I* for a detailed discussion of graphs in EViews, and [“Templates” on page 536](#) of the *User’s Guide I* for a discussion of graph templates. See [Graph::graph \(p. 158\)](#) for graph declaration and other graph types.

You may assign labels to the bars in (frozen) graph objects using the [Graph::options \(p. 164\)](#) command.

boxplot	Command Coef View Graph Command Group View Matrix View Rowvector View Series View Sym View Vector View
---------	--

Display boxplots for each series or column.

Syntax

```
boxplot(options) o1 [o2 o3 ... ]
object_name.boxplot(options) [categorical_spec(arg)]
```

where *o1*, *o2*, ..., are series or group objects. You may specify general options after the `boxplot` keyword.

The optional *categorical_spec* allows you to specify a categorical graph (see [“Categorical Spec,” on page 668](#)).

Options

<code>q = arg</code>	Set the quantile method, where <i>arg</i> can be: “r” - Rankit-Cleveland, “o” - Ordinary, “v” - van der Waerden, “b” - Blom, “t” - Tukey, “g” - Gumbel.
<code>rotate</code>	Rotate the graph so the observation axis is on the left.

Multiple series options (categorical graph settings will override these options)

<code>m</code>	Plot boxplots in multiple graphs.
----------------	-----------------------------------

Panel options

The following option applies when graphing panel structured data:

<code>panel = arg</code> (default taken from global set- tings)	Panel data display: “stack” (stack the cross-sections), “individual” or “i” (separate graph for each cross-section), “combine” or “c” (compute cross-section graphs in a single frame). (Note: more general versions of these panel graphs may be constructed as categorical graphs.)
--	--

Examples

Basic examples

```
wage.boxplot
```

displays boxplots for the series WAGE.

```
group g1 wage sex race  
g1.boxplot
```

displays boxplots for WAGES, SEX and RACE in a single graph frame.

```
g1.boxplot(m, rotate)
```

places the rotated boxplots for each series in a separate frame.

Panel examples

```
ser1.boxplot(panel=individual)
```

displays boxplots for each cross-section in a separate frame, while,

```
ser1.boxplot(panel=stack)
```

displays a single boxplot computed from the stacked panel data.

```
ser1.boxplot(panel=combined, rotate)
```

shows rotated boxplots computed for each period (across cross-sections) in a single frame.

Categorical spec examples

```
ser1.boxplot across(firm, dispname)
```

displays a categorical boxplot graph of SER1 using distinct values of FIRM to define the categories, and displaying the resulting graphs in multiple frames with common scaling. Each frame is labeled using the FIRM display name.

```
ser1.boxplot across(firm, dispname, iscale)
```

constructs the same graph with individual scaling.

```
ser1.boxplot within(firm, label=value)
```

constructs a boxplot for each value of FIRM and displays the results in a single frame. The individual boxplots are labeled using the value of FIRM associated with the category.

```
ser1.boxplot across(firm) within(income, bintype=quant,
                                bincount=4)
```

constructs a categorical boxplot with FIRM defining the across dimension, and INCOME defining the within dimension. Boxplots for each INCOME quartile of a given firm will be contained in a single frame, with different firms displayed in different frames.

```
grp1.boxplot within(sex) within(union)
```

creates an boxplot for within categories based on both SEX and UNION. Since we have not specified behavior for the implicit @SERIES in GRP1, each series in the group will be displayed in a separate frame, with individual scaling.

Cross-references

See [Chapter 13. “Graphing Data,” on page 415](#) of the *User’s Guide I* for a detailed discussion of graphs in EViews, and [“Templates” on page 536](#) of the *User’s Guide I* for a discussion of graph templates.

See [Graph::graph \(p. 158\)](#) for graph declaration and other graph types, and [Graph::set-bpelem \(p. 170\)](#) for a discussion of boxplot customization.

distplot	Command Coef View Graph Command Group View Matrix View Rowvector View Series View Sym View Vector View
----------	--

Display a distribution graph.

Syntax

```
distplot(options) o1 [o2 o3 ... ]
```

```
object_name.distplot(options) analytical_spec(arg) [categorical_spec(arg)]
```

where *o1*, *o2*, ..., are series or group objects.

When used as a command, `distplot` only allows you to display the default histogram view.

When used as an object view, you must specify the type of distribution graph you wish to create in the *analytical_spec*. You may select from: histogram, histogram polygon, histogram edge polygon, average shifted histogram, kernel density, theoretical distribution, empirical CDF, empirical survivor, empirical log survivor, or empirical quantile (see [“Analytical Spec,” on page 616](#)).

The optional *categorical_spec* allows you to specify a categorical graph (see [“Categorical Spec,” on page 668](#))

Options

Multiple series options

s	Plot in a single graph. (Categorical graph settings will override this option.)
---	---

Template and printing options

o = <i>template</i>	Use appearance options from the specified template. <i>template</i> may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workflow.
t = <i>graph_name</i>	Use appearance options and copy text and shading from the specified graph.
b / -b	[Apply / Remove] bold modifiers of the base template style specified using the “o = ” option above.
w / -w	[Apply / Remove] wide modifiers of the base template style specified using the “o = ” option above.
reset	Resets all graph options to the global defaults. May be used to remove existing customization of the graph.
p	Print the graph.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Panel options

The following option applies when graphing panel structured data.

panel = <i>arg</i> (default taken from global set- tings)	Panel data display: “stack” (stack the cross-sections), “individual” or “i” (separate graph for each cross-section). (Note: more general versions of these panel graphs may be constructed as categorical graphs.)
--	---

Analytical Spec

Specify the distribution graph you wish to create in the analytical spec. For a description of distribution graphs, see [“Analytical Graph Types,” on page 462](#) of the *User’s Guide I*. The analytical spec contains components of the form:

dist_type(dist_options)

where *dist_type* may be one of the following keywords:

hist	Histogram.
freqpoly	Histogram Polygon.
edgefreqpoly	Histogram Edge Polygon.
ash	Average Shifted Histogram.
kernel	Kernel Density
theory	Theoretical Distribution.
cdf	Empirical cumulative distribution function.
survivor	Empirical survivor function.
logsurvivor	Empirical log survivor function.
quantile	Empirical quantile function.

hist, **freqpoly**, **edgefreqpoly**, **ash**, **kernel**, and **theory** graphs may be combined in a single graph frame by providing multiple components.

Each distribution type has its own set of options, to be entered in *dist_options*:

Histogram, Histogram Polygon, Histogram Edge Polygon, and Avg. Shifted Histogram Options

scale = <i>arg</i>	<i>arg</i> specifies the scaling size, and may be “dens”, “freq”, or “relfreq”. (Note that the scaling setting is overridden if the histogram is displayed alongside a density, <i>e.g.</i> , kernel density or theoretical distribution, plot.)
binw = <i>arg</i>	<i>arg</i> specifies the bin width, and may be “views” (<i>default</i>), “sigma” (normal reference rule with $\hat{\sigma}$ as the measure of dispersion), “iqr” (normal reference rule based on the interquartile range), “silverman” (normal reference rule with Silverman’s robust measure of dispersion), “freedman” (Freedman-Diaconis).
anchor = <i>arg</i>	<i>arg</i> specifies the anchor position.
rightclosed	Right-closed bin intervals.
nshifts = <i>int</i> (<i>default</i> = 25)	Specifies the number of shift evaluations. (Only applies to average shifted histograms.)
fill	Fill the histogram. (Does not apply to the hist type.)
nofill	Don’t fill the histogram. (Does not apply to the hist type.)
leg = <i>arg</i>	Specify the legend display settings, where <i>arg</i> can be: “def” - default, “n” - none, “s” - short, “det” - detailed.

Histogram, Histogram Polygon, Histogram Edge Polygon, and Avg. Shifted Histogram Examples

```
inf.distplot hist
```

displays the default histogram view of the frequencies in each bin.

```
inf.distplot hist(scale=dens, anchor=100, binw=sigma)
```

constructs a density histogram computed using anchor position 100 and binwidth determined by the normal reference rule using $\hat{\sigma}$ as the measure of dispersion.

```
group g1 inf unemp
g1.distplot hist(scale=relfreq)
```

displays a relative frequency histogram for the series in INF and UNEMP, each in their own graph frame, while:

```
g1.distplot(s) histpoly
```

displays the two frequency histograms in the same graph frame.

```
g1.distplot freqpoly(fill)
```

constructs filled frequency polygons for the series in G1, displayed in individual frames.

```
inf.distplot edgefreqpoly(leg=detailed)
```

shows the edge frequency polygon for INF with detailed legend entries.

```
g1.distplot ash(scale=dens, rightclosed, nshifts=100)
```

constructs average shifted density histograms using 100 shifts, with right-closed bins.

Kernel Options

<code>k = arg</code> (<i>default</i> = “e”)	Kernel type: “e” (Epanechnikov), “r” (Triangular), “u” (Uniform), “n” (Normal-Gaussian), “b” (Biweight-Quartic), “t” (Triweight), “c” (Cosinus).
<code>b = number</code>	Specify a number for the bandwidth.
<code>b</code>	Bracket bandwidth.
<code>ngrid = integer</code> (<i>default</i> = 100)	Number of grid points to evaluate.
<code>x</code>	Exact evaluation.
<code>fill</code>	Fill the area.
<code>nofill</code>	Don’t fill the area.
<code>leg = arg</code>	Specify the legend display settings, where <i>arg</i> can be: “def” - default, “n” - none, “s” - short, “det”- detailed.

Kernel Examples

```
group gg weight height
```

```
gg.distplot kernel(ngrid=200, fill)
```

constructs kernel density estimates of HEIGHT and WEIGHT using 200 grid points and linear binning, and displays filled graphs in individual graph frames.

```
gg.displot(s) kernel(k=u, x)
```

computes the estimates using a uniform kernel with exact evaluation at each of the grid points, and displays the graphs in the same frame.

```
gg.displot kernel(leg=det)
```

displays the kernel plots along with detailed legend information.

Theory Options

<code>dist = arg</code>	<i>arg</i> can be: “normal”, “exp” - exponential, “logit” - logistic, “uniform” - uniform, “xman” - extreme max, “xmin” - extreme min, “chisq” - chi-squared, “pareto” - Pareto, “weibull” - Weibull, “gamma” - gamma, “tdist” - Student’s <i>t</i> -distribution.
<code>p1 = int</code>	Set first parameter.
<code>p2 = int</code>	Set second parameter.
<code>p3 = int</code>	Set third parameter.
<code>fill</code>	Fill the area.
<code>nofill</code>	Don’t fill the area.
<code>leg = arg</code>	Specify the legend display settings, where <i>arg</i> can be: “def” - default, “n” - none, “s” - short, “det” - detailed.
<code>m = int</code>	Set the iterations maximum. (Applies to logistic, extreme max, extreme min, chi-squared, Weibull, gamma or <i>t</i> -distributions.)
<code>c = int</code>	Sets the convergence criterion. (Applies to logistic, extreme max, extreme min, chi-squared, Weibull, gamma or <i>t</i> -distributions.)
<code>s</code>	Use user-specified starting values supplied in the C coefficient vector in the workfile (default uses EViews supplied starting values). (Applies to logistic, extreme max, extreme min, chi-squared, Weibull, gamma, or <i>t</i> -distributions.)

Theory Examples

```
gdp50.distplot theory(leg=det)
```

displays a normal density plot fitted to the data in GDP50 with detailed legend information.

```
gdp50.distplot theory(p1=0)
```

fits a normal density using GDP50, restricting the mean of the distribution to be zero.

```
group grol weight height
grol.distplot theory(dist=exp, fill)
```

constructs filled plots of the exponential densities fitted to the data in WEIGHT and HEIGHT, and displays them in separate frames.

```
grol.distplot(s) theory(dist=weibull, p1=5, c=1e-5)
```

fits weibull densities to the data in the series setting the first parameter to 5 and estimating the second with a convergence tolerance of 1e-5. The graphs are displayed in a single frame.

Empirical CDF, Survivor, Log Survivor, and Quantile Options

<code>q = arg</code>	Set the quantile method, where <i>arg</i> can be: “r” - Rankit-Cleveland, “o” - Ordinary, “v” - van der Waerden, “b” - Blom, “t” - Tukey, “g” - Gumbel.
<code>n</code> or <code>noci</code>	Do not include confidence intervals.
<code>ci = num</code> (<i>default</i> = 0.95)	Set confidence interval levels.
<code>leg = arg</code>	Specify the legend display settings, where <i>arg</i> can be: “def” - default, “n” - none, “s” - short, “det” - detailed.

Empirical CDF, Survivor, Log Survivor, and Quantile Examples

```
gdp50.distplot cdf
```

shows the cumulative distribution plot for GDP50, along with the default 95% confidence intervals.

```
gdp50.distplot survivor(noci)
```

displays the survivor plot for GDP50 without displaying confidence intervals.

```
group grol weight height
grol.distplot logsurvivor(ci=0.9, leg=det)
```

displays the log-survivor plots for WEIGHT and HEIGHT along with 90% confidence intervals, and a detailed legend. The plots will be displayed in individual graph frames.

```
grol.distplot(s) quantile
```

shows the quantile plots for WEIGHT and HEIGHT in the same graph frame.

Examples

Basic examples

```
distplot height weight length
```

displays default histograms for the three series.


```
group g1 age height weight length
```

```
g1.distplot hist(scale=dens, binw=sigma, leg=short) kernel theory
```

displays distribution plots for AGE, HEIGHT, WEIGHT, and LENGTH in separate frames, along with a short legend identifying each distribution plot. Each frame contains a histogram constructed using the $\hat{\sigma}$ -normal reference rule, a kernel density plot, and a plot of the theoretical normal distribution fitted to the data. (Note that the “scale = dens” option in the `hist` specification is redundant since combining a histogram with either the `kernel` or `theory` plot automatically sets the scaling.)

```
height.distplot theory theory(dist=weibull)
```

plots theoretical normal and weibull densities fit to the data in HEIGHT.

```
height.distplot quantile
```

displays a plot of the quantiles of height along with the confidence intervals.

```
g1.displot(s) cdf
```

plots the empirical CDF of the AGE, HEIGHT, WEIGHT, and LENGTH, and displays them in a single frame.

Panel examples

```
height.distplot(panel=individual) hist
```

displays histograms for each cross-section in separate frames while,

```
weight.distplot kern ash
```

displays a kernel density graph and average shifted histogram using the panel stacked WEIGHT data.

Categorical spec examples

```
height.distplot hist across(firm, dispname)
```

displays a categorical histogram graph of SER1 using distinct values of FIRM to define the categories, and displaying the resulting graphs in multiple frames.

```
height.distplot hist across(firm, dispname, iscale)
```

shows the same graph with individual scaling for each of the frames.

```
weight.distplot kernel ash within(firm, inctot, label=value)
```

displays kernel and average shifted histograms categorized by firm (with an added category for the total), with all of the graphs in a single frame and the category value used as labels.

```
length.distplot cdf across(firm, dispname) within(income, bin-  
type=quant, bincount=4)
```

constructs a categorical cdf graph with FIRM defining the across dimension, and INCOME defining the within dimension. Observations will be classified in the within dimension using the quartiles of INCOME.

Cross-references

For a description of distribution graphs, see [“Analytical Graph Types,” on page 462](#) of the *User’s Guide I*.
See [Chapter 13. “Graphing Data,” on page 415](#) of the *User’s Guide I* for a detailed discussion of graphs in EViews, and [“Templates” on page 536](#) of the *User’s Guide I* for a discussion of graph templates. See [Graph::graph \(p. 158\)](#) for graph declaration and other graph types.

dot	Command Coef View Graph Command Group View Matrix View Rowvector View Series View Sym View Vector View
-----	--

Display a dot plot graph view.

A dot plot is a symbol only version of the line and symbol graph that uses circles to represent the value of each observation.

Syntax

```
dot(options) o1 [o2 o3 ... ]  
object_name.dot(options) [categorical_spec(arg)]
```

where o1, o2, ..., are series or group objects.

Following the dot keyword, you may specify general graph characteristics using options. Available options include multiple graph handling, dual scaling, template application, data contraction, adding axis extensions, and rotation.

The optional categorical_spec allows you to specify a categorical graph (see [“Categorical Spec,” on page 668](#)).

Options

Scale options

a (default)	Automatic single scale.
d	Dual scaling with no crossing. The first series or column is scaled on the left and all other series or columns 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.
rotate	Rotate the graph so the observation axis is on the left.
ab = <i>type</i>	Add axis border along data scale, where <i>type</i> may be “hist” or “h” (histogram), “boxplot” or “b”, “kernel” or “k”. (Note: axis borders are not available for panel graphs with “panel = ” options that involve summaries: mean, median, etc.)

Multiple series options (categorical graph settings will override these options)

m	Plot dot plots in multiple graphs (will override the “s” option).
s	Stacked dot plot. Each dot represents the cumulative total of the series or columns listed. The difference between dots corresponds to the value of a series or column.

Template and printing options

o = <i>template</i>	Use appearance options from the specified template. <i>template</i> may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
t = <i>graph_name</i>	Use appearance options and copy text and shading from the specified graph.
b / -b	[Apply / Remove] bold modifiers of the base template style specified using the “o = ” option above.
w / -w	[Apply / Remove] wide modifiers of the base template style specified using the “o = ” option above.
reset	Resets all graph options to the global defaults. May be used to remove existing customization of the graph.
p	Print the graph.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Graph data options

The following option is available in non-panel or categorical graph settings:

<code>contract = key</code>	Contract the data as specified by <i>key</i> , where <i>key</i> may be: “mean”, “median”, “max”, “min”, “sum”, “var” - variance, “sd” - standard deviation, “sumsq” - sum of the squared values, “skew” - skewness, “kurt” - kurtosis, “nas” - number of missing values, “obs” - number of observations, “unique” - error if the series is not identical for all observations in a given group, “first” - first observation in category using workfile order, “last” - last observation in category using workfile order, “quant(<i>quantile</i>)” - where <i>quantile</i> is a number between 0 and 1.
-----------------------------	---

Panel options

The following option applies when graphing panel structured data:

<code>panel = arg</code> (default taken from global settings)	Panel data display: “stack” (stack the cross-sections), “individual” or “i” (separate graph for each cross-section), “combine” or “c” (combine cross-section graphs in a single frame), “mean” (plot means across cross-sections), “median” (plot median across cross-sections). (Note: more general versions of these panel graphs may be constructed as categorical graphs.)
--	---

Categorical graph options

These options only apply to categorical graphs ([“Categorical Spec,” on page 668](#)) where the graph has one or more **within** factors and a contraction method other than raw data (see the **contract** option above).

<code>favorlegend</code>	Favor the use of legends over axis labels to describe categories.
<code>elemcommon = int</code>	Specifies the number of within factors for which the graph uses common area colors. For example, with multiple within dimensions, if “elemcommon = 1”, then only categories defined by the first within factor will have common colors. If “elemcommon = 2”, then categories defined by the first two within factors will have common colors. If “elemcommon = 0”, all areas will have different colors. The default is one less than the number of within factors.

Examples

Basic examples

```
dot(rotate) oldsales newsales
```

displays rotated dotplots of OLDSALES and NEWSALES.

```
pop.dot
```

displays a dotplot graph of the series POP.

```
group mygrp oldsales newsales
mygrp.dot(m)
```

displays dotplots of each series in MYGRP, each in its own frame.

```
mygrp.dot(o=midnight, b)
```

creates a bar graph of MYGRP, using the settings of the predefined template “midnight”, applying the *bold* modifier.

```
mygrp.dot(rotate, contract=median)
```

displays a rotated dotplot of the medians of OLDSALES and NEWSALES.

Panel examples

```
ser1.dot(panel=individual)
```

displays dotplots for each cross-section in a separate frame, while,

```
ser1.dot(panel=mean)
```

displays a dotplot of the means for each period computed across cross-sections.

```
ser1.dot(panel=combine)
```

shows the dotplots for each cross-section in the same graph frame, with different symbols and colors for each cross-section.

Categorical spec examples

```
ser1.dot across(firm, dispname)
```

displays a categorical dotplot graph of SER1 using distinct values of FIRM to define the categories, and displaying the resulting graphs in multiple frames.

```
ser1.dot across(firm, dispname, iscale)
```

shows the same graph with individual scaling for each of the frames.

```
ser1.dot within(firm, inctot, label=value)
```

displays a graph categorized by firm (with an added category for the total), with all of the graphs in a single frame and the category value used as labels.

```
ser1.dot across(firm, dispname) within(income, bintype=quant,
    bincount=4)
```

constructs a categorical dotplot graph with FIRM defining the across dimension, and INCOME defining the within dimension. Observations will be classified in the within dimension using the quartiles of INCOME.

```
ser1.dot(contract=mean, elemcommon=1) within(sex) within(union)
```

creates a dotplot of mean values of within categories based on both SEX and UNION. Categories within the more slowly varying SEX factor will be drawn using the same symbol and color, while the distinct elements of UNION will employ different symbols and colors.

Cross-references

See [Chapter 13. “Graphing Data,” on page 415](#) of the *User’s Guide I* for a detailed discussion of graphs in EViews, and [“Templates” on page 536](#) of the *User’s Guide I* for a discussion of graph templates. See [Graph::graph \(p. 158\)](#) for graph declaration and other graph types.

errbar	Command Graph Command Group View Matrix View Rowvector View Sym View
--------	--

Display an error bar graph view (if possible).

If there are two series or columns, the error bar will show the high and low values in the bar. The optional third series or column will be plotted as a symbol.

Syntax

```
errbar(options) o1 o2 [o3 ...]  
object_name.errbar(options)
```

where *o1*, *o2*, ..., are series or group objects.

Options

rotate	Rotate the graph so the observation axis is on the left.
--------	--

Template and printing options

o = <i>template</i>	Use appearance options from the specified template. <i>template</i> may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
t = <i>graph_name</i>	Use appearance options and copy text and shading from the specified graph.
b / -b	[Apply / Remove] bold modifiers of the base template style specified using the “o = ” option above.
w / -w	[Apply / Remove] wide modifiers of the base template style specified using the “o = ” option above.
reset	Resets all graph options to the global defaults. May be used to remove existing customization of the graph.
p	Print the graph.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Panel options

The following option applies when graphing panel structured data:

<code>panel = arg</code> (default taken from global set- tings)	Panel data display: “stack” (stack the cross-sections), “individual” or “i” (separate graph for each cross-section), “combine” or “c” (combine cross-section graphs in a single frame), “mean” (plot means across cross-sections), “median” (plot median across cross-sections). (Note: more general versions of these panel graphs may be constructed as categorical graphs.)
--	---

Examples

Basic examples

```
errbar xlow xhigh xval
```

displays an error bar graph using the series XLOW, XHIGH, and XVAL.

```
group g1 xlow xhigh xval
```

```
g1.errbar
```

creates an error bar graph view of the three series in G1.

```
g1.errbar(o=midnight, w)
```

displays an errbar bar graph using the settings of the predefined template “midnight”, applying the *wide* modifier.

Panel examples

```
g1.errbar(panel=individual)
```

displays error bars for each cross-section in a separate frame, while,

```
g1.errbar(panel=mean)
```

displays error bars formed by computing the means for the series across cross-sections.

Cross-references

See [Chapter 13. “Graphing Data,” on page 415](#) of the *User’s Guide I* for a detailed discussion of graphs in EViews, and [“Templates” on page 536](#) of the *User’s Guide I* for a discussion of graph templates. See [Graph::graph \(p. 158\)](#) for graph declaration and other graph types.

hilo	Command Graph Command Group View Matrix View Sym View
------	---

Display a high-low[-open-close] graph view (if possible).

Syntax

```
hilo(options) o1 o2 [o3 ...]  
object_name.hilo(options)
```

where *o1*, *o2*, ..., are series or group objects. For a high-low[-open-close] graph, EViews uses the first series or column as the high series, the second series or column as the low series, and an optional third series or column as the close series. If four series or columns are provided, EViews will use them in the following order: high-low-open-close.

Note that if you wish to display a high-low-open graph, you should use an “NA”-series for the close values.

Options

rotate	Rotate the graph so the observation axis is on the left.
--------	--

Template and printing options

<i>o = template</i>	Use appearance options from the specified template. <i>template</i> may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
<i>t = graph_name</i>	Use appearance options and copy text and shading from the specified graph.
<i>b / -b</i>	[Apply / Remove] bold modifiers of the base template style specified using the “o = ” option above.
<i>w / -w</i>	[Apply / Remove] wide modifiers of the base template style specified using the “o = ” option above.
reset	Resets all graph options to the global defaults. May be used to remove existing customization of the graph.
p	Print the graph.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Panel options

The following option applies when graphing panel structured data:

<code>panel = arg</code> (default taken from global set- tings)	Panel data display: “stack” (stack the cross-sections), “individual” or “i” (separate graph for each cross-section), “combine” or “c” (combine cross-section graphs in a single frame), “mean” (plot means across cross-sections), “median” (plot median across cross-sections). (Note: more general versions of these panel graphs may be constructed as categorical graphs.)
--	---

Examples

Basic examples

```
hilo mshigh mslow msclose
```

displays a high-low-close graph using the series MSHIGH, MSLOW, and MSCLOSE.

```
group stockprice mshigh mslow msclose
stockprice.hilo(t=templt1)
```

displays a high-low-close graph of the series in STOCKPRICE, using the settings of the graph object TEMPLT1 as a template.

```
group g1 mshigh mslow msopen msclose
g1.hilo(p)
```

plots and prints the high-low-open-close graph of the four series in G1.

Panel examples

```
stockprice.hilo
```

displays the high-low-close graph for the stacked panel data.

```
stockprice.hilo(panel=individual)
```

displays high-low-close graphs for each cross-section in separate frames.

```
g1.hilo(panel=mean)
```

plots the high-low-open-close graph using the means for the series in every period computed across cross-sections.

Cross-references

See [Chapter 13. “Graphing Data,” on page 415](#) of the *User’s Guide I* for a detailed discussion of graphs in EViews, and [“Templates” on page 536](#) of the *User’s Guide I* for a discussion of graph templates. See [Graph::graph \(p. 158\)](#) for graph declaration and other graph types.

line	Command Coef View Graph Command Group View Matrix View Series View Sym View Vector View
------	---

Display a line graph view.

Syntax

```
line(options) o1 [o2 o3 ... ]
object_name.line(options) [categorical_spec(arg)]
```

where *o1*, *o2*, ..., are series or group objects. Following the `line` keyword, you may specify general graph characteristics using *options*. Available options include multiple graph handling, dual scaling, template application, data contraction, adding axis extensions, and rotation.

The optional *categorical_spec* allows you to specify a categorical graph (see “[Categorical Spec](#),” on page 668).

Options

Scale options

a (default)	Automatic single scale.
d	Dual scaling with no crossing. The first series or column is scaled on the left and all other series or columns 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.
rotate	Rotate the graph so the observation axis is on the left.
ab = type	Add axis border along data scale, where <i>type</i> may be “hist” or “h” (histogram), “boxplot” or “b”, “kernel” or “k”. (Note: axis borders are not available for panel graphs with “panel = ” options that involve summaries: mean, median, etc.).)
wf	Use workfile frequency for linked series.

Multiple series options (categorical graph settings will override these options)

m	Plot lines in multiple graphs (will override the “s” option).
s	Stacked line graph. Each line represents the cumulative total of the series or columns listed. The difference between lines corresponds to the value of a series or column.

Template and printing options

<code>o = template</code>	Use appearance options from the specified template. <i>template</i> may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
<code>t = graph_name</code>	Use appearance options and copy text and shading from the specified graph.
<code>b / -b</code>	[Apply / Remove] bold modifiers of the base template style specified using the “o = ” option above.
<code>w / -w</code>	[Apply / Remove] wide modifiers of the base template style specified using the “o = ” option above.
<code>reset</code>	Resets all graph options to the global defaults. May be used to remove existing customization of the graph.
<code>p</code>	Print the graph.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Graph data options

The following option is available in non-panel or categorical graph settings:

<code>contract = key</code>	Contract the data as specified by <i>key</i> , where <i>key</i> may be: “mean”, “median”, “max”, “min”, “sum”, “var” - variance, “sd” - standard deviation, “sumsq” - sum of the squared values, “skew” - skewness, “kurt” - kurtosis, “nas” - number of missing values, “obs” - number of observations, “unique” - error if the series is not identical for all observations in a given group, “first” - first observation in category using workfile order, “last” - last observation in category using workfile order, “quant(<i>quantile</i>)” - where <i>quantile</i> is a number between 0 and 1.
-----------------------------	---

Panel options

The following option applies when graphing panel structured data:

<code>panel = arg</code> (default taken from global settings)	Panel data display: “stack” (stack the cross-sections), “individual” or “i” (separate graph for each cross-section), “combine” or “c” (combine cross-section graphs in a single frame), “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 +/- 0.25 quantiles), “med10” (plot median and +/- 0.10 quantiles), “med05” (plot median +/- 0.05 quantiles), “med025” (plot median +/- 0.025 quantiles), “med005” (plot median +/- 0.005 quantiles), “medmxmn” (plot median, max and min). (Note: more flexible versions of the non-s.d. and on-quantile graphs may be constructed as categorical graphs.)
--	---

Categorical graph options

These options only apply to categorical graphs (“[Categorical Spec](#),” on page 668) where the graph has one or more **within** factors and a contraction method other than raw data (see the **contract** option above).

<code>favorlegend</code>	Favor the use of legends over axis labels to describe categories.
<code>elemcommon = int</code>	Specifies the number of within factors for which the graph uses common area colors. For example, with multiple within dimensions, if “elemcommon = 1”, then only categories defined by the first within factor will have common colors. If “elemcommon = 2”, then categories defined by the first two within factors will have common colors. If “elemcommon = 0”, all areas will have different colors. The default is one less than the number of within factors.

Examples

Basic examples

```
line gdp cons m1
```

displays line graphs of the series GDP, CONST, and M1.

```
group g1 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.

```
g1.line(m)
```

plots line graphs of the three series in group G1, with each plotted separately.

```
g1.line(o=midnight, b, w)
```

creates a line graph of the group G1, using the settings of the predefined template “midnight”, applying the *bold* and *wide* modifiers.

```
gdp.line(ab=boxplot)
```

displays the line graph with a boxplot displayed along the data dimension.

Panel examples

```
ser1.line(panel=individual)
```

displays area graphs with a separate graph for each cross-section, while,

```
ser1.line(panel=mean)
```

displays a line graph of the means for each period computed across cross-sections.

Categorical spec examples

```
ser1.line across(firm, dispname)
```

displays a categorical line graph of SER1 using distinct values of FIRM to define the categories, and displaying the resulting graphs in multiple frames using the display name in the labels.

```
ser1.line across(firm, dispname, iscale)
```

shows the same graph with individual scaling for each of the frames.

Cross-references

See [Chapter 13. “Graphing Data,” on page 415](#) of the *User’s Guide I* for a detailed discussion of graphs in EViews, and [“Templates” on page 536](#) of the *User’s Guide I* for a discussion of graph templates. See [Graph:graph \(p. 158\)](#) for graph declaration and other graph types.

pie	Command Graph Command Group View Matrix View Rowvector View Sym View
-----	---

Display a pie chart view.

In the default setting, there will be one pie for each date or observation number. Each series or column of data is shown as a wedge in a different color/pattern, where the width of the wedge equals the percentage contribution of the series or column to the total of all listed series or columns. Negative and missing values are treated as zeros.

Syntax

```
pie(options) o1 o2 [o3 ... ]
```

```
object_name.pie(options) [categorical_spec(arg)]
```

where *o1*, *o2*, ..., are series or group objects. You may specify general graph characteristics by including *options* following the `pie` keyword.

The optional *categorical_spec* allows you to specify a categorical graph (see [“Categorical Spec,” on page 668](#)).

Options

Template and printing options

<code>o = template</code>	Use appearance options from the specified template. <i>template</i> may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
<code>t = graph_name</code>	Use appearance options and copy text and shading from the specified graph.
<code>b / -b</code>	[Apply / Remove] bold modifiers of the base template style specified using the “o = ” option above.
<code>w / -w</code>	[Apply / Remove] wide modifiers of the base template style specified using the “o = ” option above.
<code>reset</code>	Resets all graph options to the global defaults. May be used to remove existing customization of the graph.
<code>p</code>	Print the graph.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Graph data options

The following option is available in non-panel or categorical graph settings:

<code>contract = key</code>	Contract the data as specified by <i>key</i> , where <i>key</i> may be: “mean”, “median”, “max”, “min”, “sum”, “var” - variance, “sd” - standard deviation, “sumsq” - sum of the squared values, “skew” - skewness, “kurt” - kurtosis, “nas” - number of missing values, “obs” - number of observations, “unique” - error if the series is not identical for all observations in a given group, “first” - first observation in category using workfile order, “last” - last observation in category using workfile order, “quant(<i>quantile</i>)” - where <i>quantile</i> is a number between 0 and 1.
-----------------------------	---

Panel options

The following option applies when graphing panel structured data.

<code>panel = arg</code> (default taken from global set- tings)	Panel data display: “stack” (stack the cross-sections), “individual” or “i” (separate graph for each cross-section), “combine” or “c” (combine cross-section graphs in a single frame), “mean” (plot means across cross-sections), “median” (plot median across cross-sections). (Note: more general versions of these panel graphs may be constructed as categorical graphs.)
--	---

Examples

Basic examples

```
pie const inv gov
```

displays pie charts for each period, each showing the relative sizes of CONS, INV, and GOV.

```
group g1 cons inv gov
g1.pie
```

displays the equivalent pie graph of the data in G1.

```
g1.pie(o=midnight, b, w)
```

displays the pie graph using the settings of the predefined template “midnight”, applying the *bold* and *wide* modifiers.

```
g1.pie(contract=mean)
```

displays a single pie graph with slices depicting the mean values for each series.

Panel examples

```
g1.pie(panel=individual)
```

displays pie graphs using the series in G1 with each cross-section displayed in a separate frame, while,

```
g1.pie(panel=mean)
```

displays a single pie graph showing, for each period, the pie graph formed using the means of the series computed across cross-sections.

Categorical examples

```
g1.pie(contract=mean) within(id)
```

constructs three pie graphs, one each for CONS, INV, and GOV, where the slices are determined by the relative sizes of the means of the respective series for each value of ID. There will be 10 slices for each pie.

```
g1.pie(contract=sum) within(id) within(@series)
```

displays a single pie graph with slices formed by the relative sizes of the sums of the series for each ID. If there are 10 distinct values of ID, the pie will have 30 slices.

for each value of ID using the sums of values of the series in the group G1 to determine the size of the pie slices. Each pie graph will be displayed in a separate frame. Alternately,

```
gl.pie(contract=mean) across(id) within(@series)
```

constructs one pie graph for each cross-section, where the slices are given by the mean values of CONS, INV, and GOV for the cross-section.

Cross-references

See [Chapter 13. “Graphing Data,” on page 415](#) of the *User’s Guide I* for a detailed discussion of graphs in EViews, and [“Templates” on page 536](#) of the *User’s Guide I* for a discussion of graph templates. See [Graph::graph \(p. 158\)](#) for graph declaration and other graph types.

qqplot	Command Coef View Graph Command Group View Matrix View Rowvector View Series View Sym View Vector View
--------	--

Display a quantile-quantile graph.

Plots the (empirical) quantiles of a series or matrix column against either the quantiles of a theoretical distribution or the empirical quantiles of other series or columns in the group or matrix. You may specify the theoretical distribution and/or the method used to compute the empirical quantiles as options.

Syntax

```
qqplot(options) o1 [o2 o3 ... ]  
object_name.qqplot(options) analytical_spec(arg) [categorical_spec(arg)]
```

where *o1*, *o2*, ..., are series or group objects.

When used as a command, `qqplot` displays the theoretical qq-plot against a fitted normal distribution.

When used to display the view of an object, you must specify a theoretical or empirical quantile graph in the *analytical_spec* (see [“Analytical Spec,” on page 637](#)).

The optional *categorical_spec* allows you to specify a categorical graph (see [“Categorical Spec,” on page 668](#)).

Options

Multiple series pair options (categorical graph settings will override these options)

s	Plot in a single graph (applies only to theoretical Q-Q graphs).
mult = mat_type	Multiple series or column handling: where <i>mat_type</i> may be: “pairs” or “p” - pairs, “mat” or “m” - scatterplot matrix, “lower” or “l” - lower triangular matrix.

Template and printing options

<code>o = template</code>	Use appearance options from the specified template. <i>template</i> may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
<code>t = graph_name</code>	Use appearance options and copy text and shading from the specified graph.
<code>b / -b</code>	[Apply / Remove] bold modifiers of the base template style specified using the “o = ” option above.
<code>w / -w</code>	[Apply / Remove] wide modifiers of the base template style specified using the “o = ” option above.
<code>reset</code>	Resets all graph options to the global defaults. May be used to remove existing customization of the graph.
<code>p</code>	Print the graph.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Panel options

The following option applies when graphing panel structured data.

<code>panel = arg</code> (default taken from global settings)	Panel data display: “stack” (stack the cross-sections), “individual” or “i” (separate graph for each cross-section). (Note: more general versions of these panel graphs may be constructed as categorical graphs.)
--	---

Analytical Spec

Specify the type of quantile-quantile graph you wish to create in the analytical spec. For a description of quantile-quantile graphs, see [“Analytical Graph Types,” on page 462](#) of the *User’s Guide I*. The analytical spec should be in the form:

`qq_type(type_options)`

where *qq_type* may be one of the following keywords:

<code>theory</code>	Theoretical quantile-quantile plot.
<code>empirical</code>	Empirical quantile-quantile plot (requires at least two series or columns of a matrix)

You may provide multiple theoretical qq-plot elements, but may not have more than one empirical qq-plot, nor may you mix the two.

Each type has its own set of options, to be entered in *type_options*:

Theoretical Options

<code>dist = arg</code>	<i>arg</i> can be: “normal”, “exp” - exponential, “logit” - logistic, “uniform” - uniform, “xman” - extreme max, “xmin” - extreme min, “chisq” - chi-squared, “pareto” - Pareto, “weibull” - Weibull, “gamma” - gamma, “tdist” - Student’s <i>t</i> -distribution.
<code>p1 = int</code>	Set first parameter.
<code>p2 = int</code>	Set second parameter.
<code>p3 = int</code>	Set third parameter.
<code>q = arg</code>	Set the quantile method, where <i>arg</i> can be: “r” - Rankit-Cleveland, “o” - Ordinary, “v” - van der Waerden, “b” - Blom, “t” - Tukey, “g” - Gumbel.
<code>noline</code>	Don’t display a fit line.
<code>m = int</code>	Set the iterations maximum. (Applies to logistic, extreme max, extreme min, chi-squared, Weibull, gamma, or <i>t</i> -distributions.)
<code>c = int</code>	Sets the convergence criterion. (Applies to logistic, extreme max, extreme min, chi-squared, Weibull, gamma, or <i>t</i> -distributions.)
<code>s</code>	Use user-specified starting values, supplied in the C coefficient vector in the workfile (default uses EViews supplied starting values). (Applies to logistic, extreme max, extreme min, chi-squared, Weibull, gamma, or <i>t</i> -distributions.)
<code>leg = arg</code>	Specify the legend display settings, where <i>arg</i> can be: “def” - default, “n” - none, “s” - short, “det” - detailed.

Empirical Options

<code>q = arg</code>	Set the quantile method, where <i>arg</i> can be: “r” - Rankit-Cleveland, “o” - Ordinary, “v” - van der Waerden, “b” - Blom, “t” - Tukey, “g” - Gumbel.
<code>noline</code>	Don’t display a regression line.
<code>leg = arg</code>	Specify the legend display settings, where <i>arg</i> can be: “def” - default, “n” - none, “s” - short, “det” - detailed.

Examples*Theoretical examples*

```
qqplot(s) inf unemp
```

displays theoretical qq-plots for INF and UNEMP against fitted normal distributions in a single frame.

```
group g1 inf unemp
g1.qqplot theory
```

displays theoretical qqplots of INF and UNEMP compared with normal distributions fitted to the data in each series. The graphs include fit lines and are displayed in separate frames.

```
g1.qqplot(s) theory(dist=exp)
```

compares INF and UNEMP with fitted exponential distributions, and displays the graphs in a single frame.

```
g1.qqplot(s) theory(dist=exp, p1=5)
```

plots the series against the quantiles of an exponential distribution with parameter 5 in a single frame.

Empirical Examples

```
group g2 ser1 ser2 ser3 ser4
g2.qqplot empirical
```

displays empirical qqplots for pairs of series in G2. The default behavior is to plot the first series in the group (SER1) against the remaining series (SER2, SER3, and SER4). The graphs include fit lines and are displayed in separate graph frames.

```
g1.qqplot(mult=pair) empirical(noline)
```

displays qqplots of SER1 versus SER2 and SER3 versus SER4 in separate graph frames, without a regression line.

Categorical examples

```
g1.qqplot theory within(age)
```

displays theoretical qq-plots with the series in G1 treated as the within factor and @SERIES treated as the across factor. The qq-plots for each series in G1 will be displayed in separate frames, with multiple qq-plots for each AGE category shown in each frame.

```
g1.qqplot(mult=p) empirical across(age)
```

displays empirical qq-plots for categories of AGE in separate graph frames.

Cross-references

For a description of quantile-quantile graphs, see [“Analytical Graph Types,”](#) on page 462 of the *User’s Guide I*.

See [Chapter 13. “Graphing Data,”](#) on page 415 of the *User’s Guide I* for a detailed discussion of graphs in EViews, and [“Templates”](#) on page 536 of the *User’s Guide I* for a discussion of graph templates. See [Graph::graph](#) (p. 158) for graph declaration and other graph types.

scat	Command Graph Command Group View Matrix View Rowvector View Sym View
------	--

Display a scatterplot (if possible).

A scatterplot graph plots the values of one series or column against another using symbols.

There must be at least two series or columns to create a scatterplot. By default, the first series or column 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, or to construct graphs using all possible pairs (or the lower triangular set of pairs).

Scatterplots are simply XY-line plots with symbols turned on and lines turned off (see [Graph::setelem](#) (p. 171)).

Syntax

```
scat(options) o1 o2 [o3 ... ]
object_name.scat(options) [auxiliary_spec(arg)] [categorical_spec(arg)]
```

where *o1*, *o2*, ..., are series or group objects.

Following the `scat` keyword, you may specify general graph characteristics using *options*. Available options include plotting the data in pairs or in multiple graphs, template application, and adding axis extensions.

The optional *auxiliary_spec* allows you to add fit lines to the scatterplot (regression lines, kernel fit, nearest neighbor fit, orthogonal regression, and confidence ellipses; see “[Auxiliary Spec,](#)” on page 671).

The optional *categorical_spec* allows you to specify a categorical graph (see “[Categorical Spec,](#)” on page 668).

Options

Scale options

a (<i>default</i>)	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).
d	Dual scaling with no crossing.

x	Dual scaling with possible crossing.
n	Normalized scale (zero mean and unit standard deviation). May not be used with the “s” option.
ab = <i>type</i>	Add axis border along data scales, where <i>type</i> may be “hist” or “h” (histogram), “boxplot” or “b”, “kernel” or “k”. (Note: axis borders are not available for panel graphs with “panel = ” options that involve summaries: mean, median, <i>etc.</i>)

Multiple series pair options (categorical graph settings will override these options)

m	Place scatterplots in multiple graphs.
mult = <i>mat_type</i>	Multiple series or column handling: where <i>mat_type</i> may be: “pairs” or “p” - pairs, “mat” or “m” - scatterplot matrix, “lower” or “l” - lower triangular matrix. (Using the “mat” or “lower” options is the same as using the scatmat (p. 644) command; using the “pairs” option is the same as using scatpair (p. 647).)
s	Stacked scatterplot graph. Each symbol represents the cumulative total of the series or columns listed. The difference between symbols corresponds to the value of a series or column.

Template and printing options

o = <i>template</i>	Use appearance options from the specified template. <i>template</i> may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
t = <i>graph_name</i>	Use appearance options and copy text and shading from the specified graph.
b / -b	[Apply / Remove] bold modifiers of the base template style specified using the “o = ” option above.
w / -w	[Apply / Remove] wide modifiers of the base template style specified using the “o = ” option above.
reset	Resets all graph options to the global defaults. May be used to remove existing customization of the graph.
p	Print the graph.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Note that use of the template option will override the symbol setting.

Graph data options

The following option is available in categorical graph settings:

<code>contract = key</code>	Contract the data as specified by <i>key</i> , where <i>key</i> may be: “mean”, “median”, “max”, “min”, “sum”, “var” - variance, “sd” - standard deviation, “sumsq” - sum of the squared values, “skew” - skewness, “kurt” - kurtosis, “nas” - number of missing values, “obs” - number of observations, “unique” - error if the series is not identical for all observations in a given group, “first” - first observation in category using workfile order, “last” - last observation in category using workfile order, “quant(<i>quantile</i>)” - where <i>quantile</i> is a number between 0 and 1.
-----------------------------	---

Panel options

The following option applies when graphing panel structured data.

<code>panel = arg</code> (default taken from global settings)	Panel data display: “stack” (stack the cross-sections), “individual” or “i” (separate graph for each cross-section), “combine” or “c” (combine cross-section graphs in a single frame), “mean” (plot means across cross-sections), “median” (plot median across cross-sections).
--	--

Categorical graph options

These options only apply to categorical graphs ([“Categorical Spec,” on page 668](#)) where the graph has one or more **within** factors and a contraction method other than raw data (see the **contract** option above).

<code>favorlegend</code>	Favor the use of legends over axis labels to describe categories.
<code>elemcommon = int</code>	Specifies the number of within factors for which the graph uses common area colors. For example, with multiple within dimensions, if “elemcommon = 1”, then only categories defined by the first within factor will have common colors. If “elemcommon = 2”, then categories defined by the first two within factors will have common colors. If “elemcommon = 0”, all areas will have different colors. The default is one less than the number of within factors.

Examples

Basic examples

```
scat(m) age height weight length
```

displays scatterplots with AGE on the horizontal and HEIGHT, WEIGHT and LENGTH on the vertical axis in multiple frames.

```
group g1 age height weight length
g1.scat
```

displays the same scatterplots in a single frame.

```
g1.scat(m, ab=hist)
```

displays the same information in multiple frames with histograms along the data axes.

```
g1.scat(mult=pairs) linefit
```

plots AGE against HEIGHT and WEIGHT against LENGTH (along with a regression fit line) in a single graph frame.

```
g1.scat(s, t=scat2)
```

displays a stacked scatterplot, using the graph object SCAT2 as a template.

```
g1.scat(d, ab=kernel)
```

shows a scatterplot with dual scales and no crossing, with kernel density plots along the borders.

Panel examples

```
g1.scat(panel=combined)
```

displays a scatterplot for the series in G1 in a single frame with observations for different cross-sections identified using different symbols and colors.

```
g1.scat(panel=individual)
```

draws each cross-section scatter in a different graph frame.

```
g1.scat(panel=stacked)
```

displays the same plot, but with observations drawn with common color and symbol.

```
g1.scat(panel=stacked, contract=mean) linefit kernfit
```

constructs a scatterplot using the mean values computed across cross-sections (for a given period) and displays it in a single graph frame, along with regression and kernel regression fits. The “panel= -stacked” option instructs EViews to display the observations using a single symbol type and color, and to fit lines using all of the data depicted in the graph.

Categorical examples

```
group cgrp income consumption
```

```
cgrp.scats within(sex)
```

displays a scatterplot categorized by values of sex, with both categories displayed in the same graph frame using different symbol types and colors.

```
cgrp.scats within(sex) kernfit linefit
```

displays the same graph along with linear and kernel regression fits for each category.

```
cgrp.scats(contract=mean) nnfit within(state)
```

computes mean values for the series in CGRP for each STATE category, and displays the results in a single graph frame along with a line depicting the linear regression fit to the mean values.

```
cgrp.scats across(state) within(sex) nnfit
```

displays scatterplots for data with each STATE value in different frames. Within each frame, the data for each value of SEX are depicted using different symbol types and colors, and a nearest neighbor regression is fit to observations in each category.

Cross-references

See [Chapter 13. “Graphing Data,” on page 415](#) of the *User’s Guide I* for a detailed discussion of graphs in EViews, and [“Templates” on page 536](#) of the *User’s Guide I* for a discussion of graph templates. See [Graph::graph \(p. 158\)](#) for graph declaration and other graph types.

For a description of the available fit lines, see [“Auxiliary Graph Types,” on page 480](#).

See [xyline \(p. 661\)](#) for XY graphs.

scatmat	Command Graph Command Group View Matrix View Rowvector View Sym View
---------	---

Display a matrix of scatterplots.

The `scatmat` view forms pairs using all possible pairwise combinations for the series or columns and constructs a plot for each pair, using specialized positioning and axis labeling.

Scatterplots are simply XY-line plots with symbols turned on and lines turned off (see [Graph::setelem \(p. 171\)](#)). The `scatmat` graph type is equivalent to using `scat` ([p. 640](#)) with the “mult = mat” or “mult = lower” option indicating that the data should be graphed using the full or lower-triangular matrix of pairs.

Syntax

```
scatmat(options) o1 o2 [o3 ... ]  
object_name.scatsmat(options) [auxiliary_spec(arg)]
```

where *o1*, *o2*, ..., are series or group objects.

Following the `scatmat` keyword, you may specify general graph characteristics using *options*. Available options include template application and adding axis extensions.

The optional *auxiliary_spec* allows you to add fit lines to the scatterplot (regression lines, kernel fit, nearest neighbor fit, orthogonal regression, and confidence ellipses; see [“Auxiliary Spec,”](#) on page 671).

Options

Scale options

<code>a</code> (<i>default</i>)	Automatic single scale.
<code>d</code>	Dual scaling with no crossing.
<code>x</code>	Dual scaling with possible crossing.
<code>n</code>	Normalized scale (zero mean and unit standard deviation).
<code>ab = type</code>	Add axis border along data scales, where <i>type</i> may be “hist” or “h” (histogram), “boxplot” or “b”, “kernel” or “k”. (Note: axis borders are not available for panel graphs with “panel = ” options that involve summaries: mean, median, etc.)

Multiple graph options

<code>l</code>	Plot lower triangular scatterplot matrix.
----------------	---

Template and printing options

<code>o = template</code>	Use appearance options from the specified template. <i>template</i> may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
<code>t = graph_name</code>	Use appearance options and copy text and shading from the specified graph.
<code>b / -b</code>	[Apply / Remove] bold modifiers of the base template style specified using the “o = ” option above.
<code>w / -w</code>	[Apply / Remove] wide modifiers of the base template style specified using the “o = ” option above.
<code>reset</code>	Resets all graph options to the global defaults. May be used to remove existing customization of the graph.
<code>p</code>	Print the graph.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Note that use of the template option will override the symbol setting.

Panel options

The following option applies when graphing panel structured data.

panel = <i>arg</i> (default taken from global set- tings)	Panel data display: “stack” (stack the cross-sections), “individual” or “i” (separate graph for each cross-section), “combine” or “c” (combine cross-section graphs in a single frame), “mean” (plot means across cross-sections), “median” (plot median across cross-sections). (Note: more general versions of these panel graphs may be constructed as categorical graphs.)
--	---

Examples

Basic examples

```
scatmat weight height age
```

displays a 3×3 matrix of scatter plots for all pairs of the three series

```
group g1 weight height age  
g1.scatmat
```

displays the same graph using the named group G1.

```
g1.scatmat(1)
```

shows the portion of the matrix below the diagonal.

```
g1.scatmat(1, ab=hist, o=midnight)
```

displays the lower triangular matrix with histograms along the borders using the graph settings in the pre-defined template “midnight.”

Panel examples

```
g1.scatmat(panel=combined)
```

displays a scatterplot matrix using the series in G1 with observations for different cross-sections identified using different symbols and colors.

```
g1.scatmat(panel=stacked)
```

displays the same matrix, but with a common color and symbol.

```
g1.scatmat(panel=individual, 1) linefit
```

displays a lower-triangular scatterplot matrix with regression fit for each cross-section, each in an individual frame.

Cross-references

See [Chapter 13. “Graphing Data,” on page 415](#) of the *User’s Guide I* for a detailed discussion of graphs in EViews, and [“Templates” on page 536](#) of the *User’s Guide I* for a discussion of graph templates. See [Graph::graph \(p. 158\)](#) for graph declaration and other graph types.

For a description of the available fit lines, see [“Auxiliary Graph Types,” on page 480](#) of the *User’s Guide I*.

See [xyline \(p. 661\)](#) for XY graphs.

scatpair	Command Graph Command Group View Matrix View Rowvector View Sym View
----------	--

Display a scatterplot pairs graph (if possible).

The data will be plotted 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. If the number of series or columns is odd, the last one will be ignored.

Scatterplots are simply XY plots with symbols turned on and lines turned off (see [Graph::setelem \(p. 171\)](#)). The `scatpair` graph type is equivalent to using `scat` ([p. 640](#)) with the “mult = pairs” option indicating that the data should be graphed in pairs.

Syntax

```
scatpair(options) o1 o2 [o3 ... ]
object_name.scatpair(options) [auxiliary_spec(arg)]
```

where *o1*, *o2*, ..., are series or group objects.

Following the `scatpair` keyword, you may specify general graph characteristics using *options*. Available options include plotting the data in multiple graphs, template application, and adding axis extensions.

The optional *auxiliary_spec* allows you to add fit lines to the scatterplot (regression lines, kernel fit, nearest neighbor fit, orthogonal regression, and confidence ellipses; see [“Auxiliary Spec,” on page 671](#)).

Options

Scale options

a (default)	Automatic single scale.
d	Dual scaling with no crossing.

x	Dual scaling with possible crossing.
n	Normalized scale (zero mean and unit standard deviation).
ab = <i>type</i>	Add axis border along data scales, where <i>type</i> may be “hist” or “h” (histogram), “boxplot” or “b”, “kernel” or “k”. (Note: axis borders are not available for panel graphs with “panel = ” options that involve summaries: mean, median, etc.)

Multiple series pair options

m	Place scatterplots in multiple graphs.
---	--

Template and printing options

o = <i>template</i>	Use appearance options from the specified template. <i>template</i> may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workflow.
t = <i>graph_name</i>	Use appearance options and copy text and shading from the specified graph.
b / -b	[Apply / Remove] bold modifiers of the base template style specified using the “o = ” option above.
w / -w	[Apply / Remove] wide modifiers of the base template style specified using the “o = ” option above.
reset	Resets all graph options to the global defaults. May be used to remove existing customization of the graph.
p	Print the graph.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Note that use of the template option will override the symbol setting.

Graph data options

The following option is available in categorical graph settings:

<code>contract = key</code>	Contract the data as specified by <i>key</i> , where <i>key</i> may be: “mean”, “median”, “max”, “min”, “sum”, “var” - variance, “sd” - standard deviation, “sumsq” - sum of the squared values, “skew” - skewness, “kurt” - kurtosis, “nas” - number of missing values, “obs” - number of observations, “unique” - error if the series is not identical for all observations in a given group, “first” - first observation in category using workfile order, “last” - last observation in category using workfile order, “quant(<i>quantile</i>)” - where <i>quantile</i> is a number between 0 and 1.
-----------------------------	---

Panel options

The following option applies when graphing panel structured data.

<code>panel = arg</code> (default taken from global settings)	Panel data display: “stack” (stack the cross-sections), “individual” or “i” (separate graph for each cross-section), “combine” or “c” (combine cross-section graphs in a single frame), “mean” (plot means across cross-sections), “median” (plot median across cross-sections). (Note: more general versions of these panel graphs may be constructed as categorical graphs.)
--	---

Examples

Basic examples

```
scatpair weight height age length
```

displays a combined scatterplot with AGE on the horizontal and HEIGHT on the vertical axis, and with WEIGHT on the horizontal and LENGTH on the vertical axis.

```
group g1 weight height age length
g1.scatpair
```

displays the same graph using the named group G1.

```
g1.scatpair(m, ab=kern)
```

displays each scatterplot in a separate frame with kernel density plots along the borders.

```
g1.scatpair(t=scat2)
```

displays the pairwise scatterplots, using the graph object SCAT2 as a template.

```
g1.scatpair(d)
```

shows a scatterplot for the pairs with dual scales and no crossing.

Panel examples

```
g1.scatpair kernfit
```

shows the scatterplot of the stacked panel data for pairs of series in G1. The scatterplot will be drawn with a common symbol type and color for all observations, and the kernel fit will use all of the observations.

```
g1.scatpair(panel=individual) linefit
```

displays, in individual frames, scatterplot pairs with fitted regression lines for each of the cross-sections.

```
g1.scatpair(panel=combined) linefit
```

displays the cross-section scatterplots and regression lines in a single graph frame. Different symbols and colors will be used for each cross-section series pair in the graph.

```
g1.scatpair(panel=stacked, contract=mean) nnfit kernfit
```

displays a scatterplot matrix of the mean values for each period (computed across cross-sections) in a single graph frame, along with nearest neighbor and kernel regression fits for the means.

Categorical examples

```
group cgrp income consumption interest savings  
cgrp.scatpair(d) within(sex)
```

displays a scatterplot pair graph (CONSUMPTION versus INCOME; and SAVINGS and INTEREST) categorized by values of sex, with observations displayed in the same graph frame using different symbols and colors to denote cross-sections, and dual scaling.

```
cgrp.scatpair(d) within(sex) kernfit linefit
```

displays the same scatterplot but with linear regression and kernel regression fits for the observations in each category for each pair of series.

```
cgrp.scatpair(d) across(state) within(sex) nnfit
```

displays scatterplots for observations in each STATE in different frames. Within each frame, observations are depicted using different symbols and colors to denote SEX, and a nearest neighbor regression is fit to observations in each category.

```
cgrp.scatpair(d, contract=mean) nnfit within(state)
```

computes mean values for the series in CGRP for each STATE, and displays paired scatterplots of the means, along with a line depicting the nearest neighbor regression fit to the means, in a single graph frame.

Cross-references

See [Chapter 13. “Graphing Data,” on page 415](#) of the *User’s Guide I* for a detailed discussion of graphs in EViews, and [“Templates” on page 536](#) of the *User’s Guide I* for a discussion of graph templates. See [Graph::graph \(p. 158\)](#) for graph declaration and other graph types.

For a description of the available fit lines, see [“Auxiliary Graph Types,” on page 480](#) of the *User’s Guide I*.

See [xyline \(p. 661\)](#) for XY graphs.

seasplot	Command Coef View Graph Command Group View Matrix View Rowvector View Series View Sym View Vector View
-----------------	--

Display a seasonal line graph view.

`seasplot` displays a paneled line graph view of a series or column ordered by season. This view is only available for workfiles with quarterly, monthly, or semi-annual frequencies.

Syntax

```
seasplot(options) o1 [o2 o3 ... ]
```

```
object_name.seasplot(options)
```

where *o1*, *o2*, ..., are series or group objects.

Options

m	Plot seasons using multiple overlayed lines.
---	--

Template and printing options

<code>o = template</code>	Use appearance options from the specified template. <i>template</i> may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
<code>t = graph_name</code>	Use appearance options and copy text and shading from the specified graph.
<code>b / -b</code>	[Apply / Remove] bold modifiers of the base template style specified using the “o = ” option above.
<code>w / -w</code>	[Apply / Remove] wide modifiers of the base template style specified using the “o = ” option above.
<code>reset</code>	Resets all graph options to the global defaults. May be used to remove existing customization of the graph.
<code>p</code>	Print the bar graph.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Examples

```
seasplot ipnsa ipnsb
```

displays a paneled seasonal plot of the series IPNSA and IPNSB.

```
freeze(gra_ip) ipnsa.seasplot
```

creates a graph object named GAR_IP that contains the paneled seasonal line graph view of the series IPNSA.

```
freeze(gra_ip2) ipnsa.seasplot(m)
```

creates GRA_IP2 containing the multiple line seasonal graph view of the series.

Cross-references

See [“Seasonal Graphs” on page 461](#) of the *User’s Guide I* for a brief discussion of seasonal line graphs.

See [Chapter 13. “Graphing Data,” on page 415](#) of the *User’s Guide I* for a detailed discussion of graphs in EViews, and [“Templates” on page 536](#) of the *User’s Guide I* for a discussion of graph templates. See [Graph::graph \(p. 158\)](#) for graph declaration and other graph types.

See also [Series::seas \(p. 375\)](#), [Series::x11 \(p. 400\)](#) and [Series::x12 \(p. 402\)](#).

spike	Command Coef View Graph Command Group View Matrix View Rowvector View Series View Sym View Vector View
-------	--

Display a spike graph view.

Syntax

```
spike(options) o1 [o2 o3 ... ]  
object_name.spike(options) [categorical_spec(arg)]
```

where *o1*, *o2*, ..., are series or group objects.

Following the `spike` keyword, you may specify general graph characteristics using *options*. Available options include multiple graph handling, dual scaling, template application, data contraction, adding axis extensions, and rotation.

The optional *categorical_spec* allows you to specify a categorical graph (see [“Categorical Spec,” on page 668](#)).

Options

Scale options

a (<i>default</i>)	Automatic single scale.
d	Dual scaling with no crossing. The first series or column is scaled on the left and all other series or columns are scaled on the right.
x	Dual scaling with possible crossing. See the “d” option.
n	Normalized scale (zero mean and unit standard deviation).
rotate	Rotate the graph so the observation axis is on the left.
ab = <i>type</i>	Add axis border along data scales, where <i>type</i> may be “hist” or “h” (histogram), “boxplot” or “b”, “kernel” or “k”. (Note: axis borders are not available for panel graphs with “panel = ” options that involve summaries: mean, median, etc.).)

Multiple series options (*categorical graph settings will override these options*)

m	Plot spikes in multiple graphs.
l	Spike graph for the first series or column listed and a line graph for all subsequent series or columns.

Template and printing options

o = <i>template</i>	Use appearance options from the specified template. <i>template</i> may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
t = <i>graph_name</i>	Use appearance options and copy text and shading from the specified graph.
b / -b	[Apply / Remove] bold modifiers of the base template style specified using the “o = ” option above.
w / -w	[Apply / Remove] wide modifiers of the base template style specified using the “o = ” option above.
reset	Resets all graph options to the global defaults. May be used to remove existing customization of the graph.
p	Print the spike graph.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Graph data options

The following option is available in non-panel or categorical graph settings:

<code>contract = key</code>	Contract the data as specified by <i>key</i> , where <i>key</i> may be: “mean”, “median”, “max”, “min”, “sum”, “var” - variance, “sd” - standard deviation, “sumsq” - sum of the squared values, “skew” - skewness, “kurt” - kurtosis, “nas” - number of missing values, “obs” - number of observations, “unique” - error if the series is not identical for all observations in a given group, “first” - first observation in category using workfile order, “last” - last observation in category using workfile order, “quant(<i>quantile</i>)” - where <i>quantile</i> is a number between 0 and 1.
-----------------------------	---

Panel options

The following option applies when graphing panel structured data:

<code>panel = arg</code> (default taken from global settings)	Panel data display: “stack” (stack the cross-sections), “individual” or “i” (separate graph for each cross-section), “combine” or “c” (combine cross-section graphs in a single frame), “mean” (plot means across cross-sections), “median” (plot median across cross-sections). (Note: more general versions of these panel graphs may be constructed as categorical graphs.)
--	---

Categorical graph options

These options only apply to categorical graphs, which are described below and specified by the **within** and **across** categorical spec. The graph must have one or more **within** factors and a contraction method other than raw data (see the **contract** option above).

<code>favorlegend</code>	Favor the use of legends over axis labels to describe categories.
<code>elemcommon = int</code>	Specifies the number of within factors for which the graph uses common area colors. For example, with multiple within dimensions, if “elemcommon = 1”, then only categories defined by the first within factor will have common colors. If “elemcommon = 2”, then categories defined by the first two within factors will have common colors. If “elemcommon = 0”, all areas will have different colors. The default is one less than the number of within factors.

Examples

Basic examples

```
spike(rotate, m) pop oldsales newsales
```

displays a rotated spike graph of the series POP, OLDSALES, and NEWSALES, with each series in a separate frame.

```
pop.spike
```

displays a spike graph of the series POP.

```
group mygrp oldsales newsales
mygrp.spike(l, x, o=mytpt)
```

plot a spike graph of OLDSALES together with a line graphs of NEWSALES. The spike graph is scaled on the left, while the line graph is scaled on the right. The graph uses options from the graph MYTPT as a template.

```
mygrp.spike(o=midnight, b)
```

creates a spike graph of MYGRP, using the settings of the predefined template “midnight.”

```
mygrp.spike(rotate, contract=mean)
```

displays a rotated spike graph of the means of the series in MYGRP.

Panel examples

```
ser1.spike(panel=individual)
```

displays spike graphs for each cross-section in a separate frame, while,

```
ser1.spike(panel=median)
```

displays a spike graph of the medians for each period computed across cross-sections.

Categorical spec examples

```
ser1.spike across(firm, dispname)
```

displays a categorical spike graph of SER1 using distinct values of FIRM to define the categories, and displaying the resulting graphs in multiple frames.

```
ser1.spike across(firm, dispname, iscale)
```

shows the same graph with individual scaling for each of the frames.

```
ser1.spike within(contract=mean, firm, inctot, label=value)
```

displays a spike graph of mean values of SER1 categorized by firm (along with an added category for the total), with all of the graphs in a single frame and the FIRM category value used as labels.

```
ser1.spike(contract=sum) across(firm, dispname) within(income,
  bintype=quant, bincount=4)
```

constructs a categorical spike graph of the sum of SER1 values within a category. Different firms are displayed in different graph frames, using the display name as labels, with each frame containing spikes depicting the sum of SER1 for each income quartiles.

```
group mygrp oldsales newsales
```

```
mygrp.spike(contract=min) within(@series) within(age)
```

displays spike graphs of the minimum values for categories defined by distinct values of AGE (and the two series). All of the spike will be displayed in a single frame with the spikes for OLDSALES grouped together followed by the spikes for NEWSALES.

Cross-references

See [Chapter 13. “Graphing Data,” on page 415](#) of the *User’s Guide I* for a detailed discussion of graphs in EViews, and [“Templates” on page 536](#) of the *User’s Guide I* for a discussion of graph templates. See [Graph::graph \(p. 158\)](#) for graph declaration and other graph types.

xyarea	Command Graph Command Group View Matrix View Sym View
--------	---

Display an XY area graph view (if possible).

An XY area graph plots the values of one series or column against another. It is similar to a XY line, but with the region between the line and the zero horizontal axis filled.

(Note that XY area graphs are typically employed only when data along the horizontal axis are ordered.)

There must be at least two series or columns to create an XY area graph. By default, the first series or column will be located along the horizontal axis, with 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, or to construct graphs using all possible pairs (or the lower triangular set of pairs).

Syntax

```
xyarea(options) o1 o2 [o3 ... ]  
object_name.xyarea(options)
```

where *o1*, *o2*, ..., are series or group objects.

Options

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).
d	Dual scaling with no crossing.

x	Dual scaling with possible crossing.
n	Normalized scale (zero mean and unit standard deviation).
ab = <i>type</i>	Add axis border along data scale, where <i>type</i> may be “hist” or “h” (histogram), “boxplot” or “b”, “kernel” or “k”. (Note: axis borders are not available for panel graphs with “panel = ” options that involve summaries: mean, median, <i>etc.</i>)

Multiple series pair options (categorical graph settings will override these options)

m	Plot areas in multiple graphs.
s	Stacked graph. Each line represents the cumulative total of the series or columns listed. The difference between lines corresponds to the value of a series or column.

Template and printing options

o = <i>template</i>	Use appearance options from the specified template. <i>template</i> may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
t = <i>graph_name</i>	Use appearance options and copy text and shading from the specified graph.
b / -b	[Apply / Remove] bold modifiers of the base template style specified using the “o = ” option above.
w / -w	[Apply / Remove] wide modifiers of the base template style specified using the “o = ” option above.
reset	Resets all graph options to the global defaults. May be used to remove existing customization of the graph.
p	Print the graph.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Panel options

The following option applies when graphing panel structured data:

<code>panel = arg</code> (default taken from global set- tings)	Panel data display: “stack” (stack the cross-sections), “individual” or “i” (separate graph for each cross-section), “combine” or “c” (combine cross-section graphs in a single frame), “mean” (plot means across cross-sections), “median” (plot median across cross-sections). (Note: more general versions of these panel graphs may be constructed as categorical graphs.)
--	---

Examples

Basic examples

```
xyarea income sales
```

displays an XY-area graph with INCOME on the horizontal and SALES on the vertical axis.

```
group g1 income sales
```

```
g1.xyarea
```

plots the same graph using the named object G1.

```
g1.xyarea(ab=boxplot, t=gr1)
```

displays the graph with boxplots along the axes, using the template settings from the graph GR1.

Panel examples

```
g1.xyarea
```

displays an XY-area graph for the stacked panel data.

```
g1.xyarea(panel=individual)
```

displays XY-area graphs for each cross-section in separate graph frames.

```
g1.xyarea(panel=mean)
```

computes means for each period across cross-sections, then displays the XY-area graph for the mean data in a single graph frame. Note that only in a very narrow set of circumstances is this latter command likely to yield a sensible graph.

Cross-references

[scat](#) (p. 640) and [xyline](#) (p. 661) are specialized forms of XY graphs.

See [Chapter 13. “Graphing Data,” on page 415](#) of the *User’s Guide I* for a detailed discussion of graphs in EViews, and [“Templates” on page 536](#) of the *User’s Guide I* for a discussion of graph templates. See [Graph: :graph](#) (p. 158) for graph declaration and other graph types.

xybar	Command Graph Command Group View Matrix View Rowvector View Sym View
-------	--

Display an XY bar graph view (if possible).

An XY bar graph displays the data in sets of three series or columns as a vertical bar. For a given observation, the values in the first two series or columns define a region along the horizontal axis, while the value in the third series or column defines the vertical height of the bar.

XY bar graphs may, for example, be used to construct variable width histograms.

Syntax

```
xybar(options) o1 o2 [o3 ... ]
object_name.xybar(options)
```

where *o1*, *o2*, ..., are series or group objects.

Options

n	Normalized scale (zero mean and unit standard deviation).
---	---

Template and printing options

o = <i>template</i>	Use appearance options from the specified template. <i>template</i> may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
t = <i>graph_name</i>	Use appearance options and copy text and shading from the specified graph.
b / -b	[Apply / Remove] bold modifiers of the base template style specified using the “o = ” option above.
w / -w	[Apply / Remove] wide modifiers of the base template style specified using the “o = ” option above.
reset	Resets all graph options to the global defaults. May be used to remove existing customization of the graph.
p	Print the graph.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Panel options

The following option applies when graphing panel structured data:

<code>panel = arg</code> (default taken from global set- tings)	Panel data display: “stack” (stack the cross-sections), “individual” or “i” (separate graph for each cross-section), “combine” or “c” (combine cross-section graphs in a single frame in single graph frame), “mean” (plot means across cross-sections), “median” (plot median across cross-sections). (Note: more general versions of these panel graphs may be constructed as categorical graphs.)
--	---

Examples

Basic examples

```
xyarea lowbin highbin height
```

plots an XY-bar graph using LOWBIN and HIGHBIN to define the bin ranges and HEIGHT to draw the corresponding bar height.

```
group g1 lowbin highbin height  
g1.xybar
```

plots the same graph using the named object G1.

```
g1.xybar (t=t1)
```

displays the graph using the template settings from the graph object T1.

Panel examples

```
g1.xybar (panel=individual)
```

displays an XY-bar graph for each cross-section in an individual graph frame.

```
g1.xybar (panel=mean)
```

displays an XY-bar graph for the data formed by taking means across cross-sections for each period. Note that only in a very narrow set of circumstances is this latter command likely to yield a sensible graph.

Cross-references

[scat](#) (p. 640), [xyarea](#) (p. 656), [xyline](#) (p. 661), and [xypair](#) (p. 664) are specialized forms of XY graphs.

See [Chapter 13. “Graphing Data,”](#) on page 415 of the *User’s Guide I* for a detailed discussion of graphs in EViews, and [“Templates”](#) on page 536 of the *User’s Guide I* for a discussion of graph templates. See [Graph::graph](#) (p. 158) for graph declaration and other graph types.

xyline	Command Graph Command Group View Matrix View Sym View
--------	---

Display an XY line graph view (if possible).

There must be at least two series or columns to create an XY line graph. By default, the first series or column will be located along the horizontal axis, with 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, or to construct graphs using all possible pairs (or the lower triangular set of pairs).

XY line graphs are simply XY plots with lines turned on and symbols turned off (see [Graph::setelem \(p. 171\)](#)).

Syntax

```
xyline(options) o1 o2 [o3 ... ]
object_name.xyline(options) [auxiliary_spec(arg)] [categorical_spec(arg)]
```

where *o1*, *o2*, ..., are series or group objects.

Following the `xyline` keyword, you may specify general graph characteristics using *options*. Available options include plotting the data in pairs or in multiple graphs, template application, and adding axis extensions.

The optional *auxiliary_spec* allows you to add fit lines to the scatterplot (regression lines, kernel fit, nearest neighbor fit, orthogonal regression, and confidence ellipses; see [“Auxiliary Spec,” on page 671](#)).

The optional *categorical_spec* allows you to specify a categorical graph (see [“Categorical Spec,” on page 668](#)).

Options

Scale options

a (<i>default</i>)	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).
d	Dual scaling with no crossing.

x	Dual scaling with possible crossing.
n	Normalized scale (zero mean and unit standard deviation). May not be used with the “s” option.
ab = <i>type</i>	Add axis border along data scales, where <i>type</i> may be “hist” or “h” (histogram), “boxplot” or “b”, “kernel” or “k”. (Note: axis borders are not available for panel graphs with “panel = ” options that involve summaries: mean, median, etc.)

Multiple series pair options (categorical graph settings will override these options)

m	Plot XY lines in multiple graphs.
mult = <i>mat_type</i>	Multiple series or column handling: where <i>mat_type</i> may be: “pairs” or “p” - pairs, “mat” or “m” - scatterplot matrix, “lower” or “l” - lower triangular matrix. (Using the “pairs” options is the same as using the xypair (p. 664) command.)
s	Stacked XY line graph. Each line represents the cumulative total of the series or columns listed. The difference between lines corresponds to the value of a series or column.

Template and printing options

o = <i>template</i>	Use appearance options from the specified template. <i>template</i> may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
t = <i>graph_name</i>	Use appearance options and copy text and shading from the specified graph.
b / -b	[Apply / Remove] bold modifiers of the base template style specified using the “o = ” option above.
w / -w	[Apply / Remove] wide modifiers of the base template style specified using the “o = ” option above.
reset	Resets all graph options to the global defaults. May be used to remove existing customization of the graph.
p	Print the graph.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Note that use of the *template* option will override the *lines* setting.

Graph data options

The following option is available in categorical graph settings:

<code>contract = key</code>	Contract the data as specified by <i>key</i> , where <i>key</i> may be: “mean”, “median”, “max”, “min”, “sum”, “var” - variance, “sd” - standard deviation, “sumsq” - sum of the squared values, “skew” - skewness, “kurt” - kurtosis, “nas” - number of missing values, “obs” - number of observations, “unique” - error if the series is not identical for all observations in a given group, “first” - first observation in category using workfile order, “last” - last observation in category using workfile order, “quant(<i>quantile</i>)” - where <i>quantile</i> is a number between 0 and 1.
-----------------------------	---

Panel options

The following option applies when graphing panel structured data.

<code>panel = arg</code> (default taken from global settings)	Panel data display: “stack” (stack the cross-sections), “individual” or “i” (separate graph for each cross-section), “combine” or “c” (combine cross-section graphs in a single frame), “mean” (plot means across cross-sections), “median” (plot median across cross-sections). (Note: more general versions of these panel graphs may be constructed as categorical graphs.)
--	---

Examples

Basic examples

```
xyline age height weight length
```

displays XY-line plots with AGE on the horizontal and HEIGHT, WEIGHT and LENGTH on the vertical axis.

```
group g1 age height weight length
g1.xyline
```

displays the same graph using the named object G1.

```
g1.xyline(m, ab=hist)
```

displays the same information in multiple frames with histograms along the borders.

```
g1.xyline(s, t=scat2)
```

displays a stacked XY-line graph, using the graph object SCAT2 as a template.

```
g1.xyline(d)
```

shows XY-line plots with dual scales and no crossing.

Panel examples

```
g1.xyline(panel=combined)
```

displays XY-line for series in G1 in a single frame with lines for different cross-sections for a given pair identified using different symbols and colors.

```
g1.xyline(panel=individual)
```

displays the graphs for each of the cross-sections in a different frame.

```
g1.xyline(panel=stacked)
```

displays the same plot, but with lines drawn from the beginning of the stacked panel to the end.

Categorical examples

```
group cgrp income consumption
```

```
cgrp.xyline within(sex)
```

displays a scatterplot categorized by values of sex, with both categories displayed in the same graph frame using different symbols and colors.

```
cgrp.xyline(contract=mean) within(state)
```

computes mean values for the series in CGRP for each STATE category, and displays the results in a single graph frame using a single line to connect the mean values.

```
cgrp.xyline across(state) within(sex)
```

displays line plots for data with each STATE value in different frames. Within each frame, the data for each value of SEX are drawn as a separate line.

Cross-references

[scat](#) (p. 640) is a specialized form of an XY graph.

See [Chapter 13. “Graphing Data,” on page 415](#) of the *User’s Guide I* for a detailed discussion of graphs in EViews, and [“Templates” on page 536](#) of the *User’s Guide I* for a discussion of graph templates. See [Graph::graph](#) (p. 158) for graph declaration and other graph types.

xypair	Command Graph Command Group View Matrix View Rowvector View Sym View
--------	---

Display an XY pairs graph (if possible).

The data will be plotted 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. If the number of series or columns is odd, the last one will be ignored.

XY line graphs are simply XY plots with lines turned on and symbols turned off (see [Graph::setelem \(p. 171\)](#)). The `xypair` graph type is equivalent to using `xyline` ([p. 661](#)) with the “mult = pairs” option indicating that the data should be graphed in pairs.

Syntax

```
xypair(options) o1 o2 [o3 ... ]
object_name.xypair(options) [auxiliary_spec(arg)]
```

Following the `xypair` keyword, you may specify general graph characteristics using *options*. Available options include plotting the data in multiple graphs, template application, and adding axis extensions.

The optional *auxiliary_spec* allows you to add fit lines to the scatterplot (regression lines, kernel fit, nearest neighbor fit, orthogonal regression, and confidence ellipses; see [“Auxiliary Spec,” on page 671](#)).

Options

Scale options

a (<i>default</i>)	Automatic single scale.
d	Dual scaling with no crossing.
x	Dual scaling with possible crossing.
n	Normalized scale (zero mean and unit standard deviation).
ab = <i>type</i>	Add axis border along data scales, where <i>type</i> may be “hist” or “h” (histogram), “boxplot” or “b”, “kernel” or “k”. (Note: axis borders are not available for panel graphs with “panel = ” options that involve summaries: mean, median, <i>etc.</i>)

Multiple series pair options

m	Plot XY lines in multiple graphs.
---	-----------------------------------

Template and printing options

<code>o = template</code>	Use appearance options from the specified template. <i>template</i> may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
<code>t = graph_name</code>	Use appearance options and copy text and shading from the specified graph.
<code>b / -b</code>	[Apply / Remove] bold modifiers of the base template style specified using the “o = ” option above.
<code>w / -w</code>	[Apply / Remove] wide modifiers of the base template style specified using the “o = ” option above.
<code>reset</code>	Resets all graph options to the global defaults. May be used to remove existing customization of the graph.
<code>p</code>	Print the graph.

The options which support the “-” may be preceded by a “+” or “-” indicating whether to turn on or off the option. The “+” is optional.

Note that use of the template option will override the pair and line settings.

Graph data options

The following option is available in categorical graph settings:

<code>contract = key</code>	Contract the data as specified by <i>key</i> , where <i>key</i> may be: “mean”, “median”, “max”, “min”, “sum”, “var” - variance, “sd” - standard deviation, “sumsq” - sum of the squared values, “skew” - skewness, “kurt” - kurtosis, “nas” - number of missing values, “obs” - number of observations, “unique” - error if the series is not identical for all observations in a given group, “first” - first observation in category using workfile order, “last” - last observation in category using workfile order, “quant(<i>quantile</i>)” - where <i>quantile</i> is a number between 0 and 1.
-----------------------------	---

Panel options

The following option applies when graphing panel structured data.

<code>panel = arg</code> (default taken from global set- tings)	Panel data display: “stack” (stack the cross-sections), “individual” or “i” (separate graph for each cross-section), “combine” or “c” (combine cross-section graphs in a single frame), “mean” (plot means across cross-sections), “median” (plot median across cross-sections). (Note: more general versions of these panel graphs may be constructed as categorical graphs.)
--	---

Basic examples

```
xypair age height weight length
```

displays XY-line plots with AGE on the horizontal and HEIGHT on the vertical axis, and WEIGHT on the horizontal and LENGTH on the vertical axis.

```
group g1 age height weight length
g1.xypair
```

plots the same graph using the named object G1.

```
g1.xypair(m, ab=boxplot)
```

displays the same information in multiple frames with boxplots along the axes.

```
g1.xypair(t=scat2)
```

displays the XY-line pair graphs, using the graph object SCAT2 as a template.

```
g1.xypair(d, ab=hist)
```

shows the paired XY-line plots with dual scales and no crossing, and histograms along the borders.

Panel examples

```
g1.xypair(panel=combined)
```

displays XY-line graphs in a single frame, with different lines types and colors for different cross-sections pairs.

```
g1.xypair(panel=individual)
```

displays the graphs for each of each cross-section in a different frame.

```
g1.xypair(panel=stacked)
```

constructs a single frame graph with lines drawn from the beginning of the stacked panel to the end.

```
g1.xypair(panel=mean)
```

constructs line graphs for pairs of series using the mean values computed across cross-sections (for a given period), and displays them in a single frame.

Categorical examples

```
group cgrp income consumption sales revenue
cgrp.xypair within(sex)
```

displays a paired data line graphs categorized by values of sex, with both categories displayed in the same graph frame using different line types and colors.

```
cgrp.xypair(contract=mean) within(state)
```

computes mean values for the series in CGRP for each STATE category, and displays the results in a single graph frame.

```
cgrp.xypair across(state) within(sex)
```

displays line plots for data with each STATE value in different frames. Within each frame, the data for each value of SEX are drawn as a separate line.

Cross-references

[scat](#) (p. 640) and [xyline](#) (p. 661) are specialized forms of XY graphs.

See [Chapter 13. “Graphing Data,” on page 415](#) of the *User’s Guide I* for a detailed discussion of graphs in EViews, and [“Templates” on page 536](#) of the *User’s Guide I* for a discussion of graph templates. See [Graph::graph](#) (p. 158) for graph declaration and other graph types.

Optional Graph Components

The following sections describe optional components that may be used as part of a graph specification:

- A categorical spec may be added to most graph commands to create a categorical graph.
- An auxiliary spec may be added to an XY graph command ([scat](#) (p. 640), [scatmat](#) (p. 644), [scatpair](#) (p. 647), [xyarea](#) (p. 656), [xybar](#) (p. 659), [xyline](#) (p. 661), [xypair](#) (p. 664)) to add fit lines (or confidence ellipses) to the graph.

Categorical Spec

Adding a categorical spec to a graph commands produces a categorical graph. For example, adding a categorical spec to a bar graph generates a categorical bar graph using the factors defined by the spec; adding a categorical spec to an XY-line graph creates a categorical XY-line graph.

The categorical spec is used to specify the factors used in categorization. It may include one or more **within** and **across** factors of the following form:

```
within(factor_name[, factor_options])
```


or

```
across(factor_name[, factor_options])
```

where *factor_name* is the name of a series used to define a category along with the *factor_options*. Multiple factors of a given type should be listed in order from most slowly to fastest varying.

Categorical graphs are not supported for matrix object views. Note also that use of a categorical specification will override any panel options.

Factor options

<code>incna</code>	include NA category
<code>inctot</code>	include total category
<code>iscale, cscale</code>	individual/common scale for this factor. The default is individual for the “@series” factor, and common for all others.
<code>iscalex, cscalex</code>	individual/common X axis scale for this factor. The default is individual for the “@series” factor, and common for all others.
<code>iscaley, cscaley</code>	individual/common Y axis scale for this factor. The default is individual for the “@series” factor, and common for all others.
<code>bintype = type</code>	bin type, where <i>type</i> can be: “auto” (default), “quant” - quantile binning, “value” - value binning, “none” - forces no binning.
<code>bincount = int</code>	<i>int</i> is the number of quantile bins or maximum number of value bins.
<code>dispname</code>	use display name in labels
<code>label = key</code>	<i>key</i> can be: “auto” (default), “value” - factor value only, “both” - factor name and value.
<code>ncase = key</code>	sets the capitalization for factor names in labels, where <i>key</i> can be: “upper”, “lower”, “title”. The default is to preserve case.
<code>vcase = key</code>	sets the capitalization for factor values in labels, where <i>key</i> can be: “upper”, “lower”, “title”. The default is to preserve case.

Categorical spec examples

```
profit.boxplot across (firm)
```

displays a categorical boxplot graph of PROFITS using distinct values of FIRM to define the categories, and displaying the graphs in multiple frames.

```
profit.boxplot across(firm, dispname, iscale)
```

shows the same graph with individual scaling for each of the frames, using the displayname in labels.

```
profit.boxplot within(firm, inctot, label=value)
```

displays a boxplot graph categorized by firm (with an added category for the total), with all of the graphs in a single frame and the category value used as labels.

```
ser1.bar(contract=sum) across(firm, dispname) within(income, bin-  
type=quant, bincount=4)
```

constructs a categorical bar graph of the sum of SER1 values within a category. Different firms are displayed in different graph frames, using the display name as labels, with each frame containing bars depicting the sum of SER1 for each income quartiles.

```
ser1.bar(contract=mean, elemcommon=1) within(sex) within(union)
```

creates a bar graph of mean values of within categories based on both SEX and UNION. Categories for the distinct elements of UNION will be depicted using different bar colors, with the color assignment repeated for different values of SEX.

By default, the multiple series in a group are treated as the first (most slowly varying) across factor. To control the treatment of this implicit factor, you may use the “@series” keyword in a `within` or `across` specification; if the factor is not the first one of its type listed, it will be treated as the last factor. Thus:

```
g1.boxplot within(sex) within(union)
```

creates an boxplot for within categories based on both SEX and UNION. Since we have not specified behavior for the implicit series factor in GRP1, the series in the group will be treated as the first across factor and will be displayed in a separate frame.

```
g1.qqplot theory within(age)
```

displays theoretical qq-plots with the series in G1 treated as the within factor and @SERIES treated as the across factor. The qq-plots for each series in G1 will be displayed in separate frames, with multiple qq-plots for each AGE category shown in each frame.

```
g1.distplot hist kernel across(sex) across(@series) across(age)
```

displays histograms and kernel density plots where the implicit factor is the last across factor.

```
group mygrp oldsales newsales
```

```
mygrp.bar(contract=min) within(@series) within(age)
```

displays bar graphs of the minimum values for categories defined by distinct values of AGE (and the two series). All of the bars will be displayed in a single frame with the bars for OLDSALES grouped together followed by the bars for NEWSALES.

```
mygrp.bar(contract=median, elemcommon=2) across(firm)
      across(@series) across(age)
```

also adds an additional categorization using the FIRM identifiers. The observations for a given firm are grouped together. Within a firm, the bars for the OLDSALES and NEWSALES, which will be depicted using different colors, will be grouped within each age category. The color assignment to OLDSALES and NEWSALES will be repeated across firms and ages (note that @SERIES is treated as the last across factor).

Auxiliary Spec

You may add one or more fit lines or confidence ellipses to your XY graph using an auxiliary spec. (Note that auxiliary specs are not allowed with stacked XY graphs.)

For a description of the available fit line types, see “Auxiliary Graph Types,” on page 480 of the *User’s Guide I*.

The auxiliary spec should be in the form:

fitline_type(type_options)

where *fitline_type* is one of the following keywords:

linefit	Add a regression line.
kernfit	Add a kernel fit line.
nnfit	Add a nearest neighbor fit line.
orthreg	Add an orthogonal regression line.
cellipse	Add a confidence ellipse.

Each fit line type has its own set of options, to be entered in *type_options*:

Linefit Options

yl	Take the natural log of first series or column, <i>y</i> .
yi	Take the inverse of <i>y</i> .
yp = <i>number</i>	Take <i>y</i> to the power of the specified number.
yb = <i>number</i>	Take the Box-Cox transformation of <i>y</i> with the specified parameter.
xl	Take the natural log of <i>x</i> .
xi	Take the inverse of <i>x</i> .
xp = <i>number</i>	Take <i>x</i> to the power of the specified number.

<code>xb = number</code>	Take the Box-Cox transformation of x with the specified parameter.
<code>xd = integer</code>	Fit a polynomial of x up to the specified power.
<code>m = integer</code>	Set number of robustness iterations.
<code>leg = arg</code>	Specify the legend display settings, where <i>arg</i> can be: “def” - default, “n” - none, “s” - short, “det”- detailed.

If the polynomial degree of x leads to singularities in the regression, EViews will automatically drop high order terms to avoid collinearity.

Linefit Examples

```
group g1 x y z w
g1.scatpair linefit (yl,xl)
```

displays a scatterplot of Y against X and W against Z, together with the fitted values from a regression of log Y on log X and log W on log Z.

```
g1.scat linefit linefit (yb=0.5,m=10)
```

shows scatterplots of Y, Z, and W along the vertical axis and X along the horizontal axis, and superimposes both a simple linear regression fit and a fit of the Box-Cox transformation of the vertical axis variable against X, with 10 iterations of bisquare weights,.

Kernfit Options

<code>k = arg</code> (<i>default</i> = “e”)	Kernel type: “e” (Epanechnikov), “r” (Triangular), “u” (Uniform), “n” (Normal-Gaussian), “b” (Biweight-Quartic), “t” (Triweight), “c” (Cosinus).
<code>b = number</code>	Specify a number for the bandwidth.
<code>b</code>	Bracket bandwidth.
<code>ngrid = integer</code> (<i>default</i> = 100)	Number of grid points to evaluate.
<code>x</code>	Exact evaluation of the polynomial fit.
<code>d = integer</code> (<i>default</i> = 1)	Degree of polynomial to fit. Set “d = 0” for Nadaraya-Watson regression.
<code>leg = arg</code>	Specify the legend display settings, where <i>arg</i> can be: “def” - default, “n” - none, “s” - short, “det”- detailed.

Kernfit Examples

```
group gg weight height length volume
gg.scat kernfit kernfit (d=2, b)
```

displays scatterplots with HEIGHT, LENGTH, and VOLUME on the vertical axis and WEIGHT on the horizontal axis, along with the default kernel regression fit, and a second-degree polynomial fit with bracketed bandwidths.

```
gg.scatmat kernfit(ngrid=200)
```

displays a scatterplot matrix of the series in GG and fits a kernel regression of the Y-axis variable on the X-axis variable using 200 grid points.

Nnfit Options

<code>d = integer</code> (default = 1)	Degree of polynomial to fit.
<code>b = fraction</code> (default = 0.3)	Bandwidth as a fraction of the total sample. The larger the fraction, the smoother the fit.
<code>b</code>	Bracket bandwidth span.
<code>s</code>	Symmetric neighbors. Default is nearest neighbors.
<code>u</code>	No local weighting. Default is local weighting using tricube weights.
<code>m = integer</code>	Set number of robustness iterations.
<code>x</code>	Exact (full) sampling. Default is Cleveland subsampling.
<code>neval = integer</code> (default = 100)	Approximate number of data points at which to compute the fit (if performing Cleveland subsampling).
<code>leg = arg</code>	Specify the legend display settings, where <i>arg</i> can be: “def” - default, “n” - none, “s” - short, “det” - detailed.

Nnfit Examples

```
group gr1 gdp90 cons90 gdp70 cons70
gr1.scatpair nnfit(x,m=3)
```

displays the nearest neighbor fit of CONS90 on GDP90 and of CONS70 on GDP70 with exact (full) sampling and 3 robustness iterations. Each local regression fits the default linear regression, with tricube weighting and a bandwidth of span 0.3.

```
gr1.scatpair nnfit nnfit(neval=50,d=2,m=3)
```

computes both the default nearest neighbor fit and a custom fit that fits a quadratic at 50 data points, using tricube robustness weights with 3 robustness iterations.

Orthreg Options

<code>leg = arg</code>	Specify the legend display settings, where <i>arg</i> can be: “def” - default, “n” - none, “s” - short, “det” - detailed.
------------------------	---

Orthreg Examples

```
group gg weight height length volume
gro1.scatmat(1) orthreg
```

displays the orthogonal regression fit for each pair in the lower-triangle scatterplot matrix.

Cellipse Options

<code>size = arg</code>	Specify the confidence levels.
<code>c</code>	Use χ^2 distribution to compute the confidence ellipses. The default is to use the F -distribution.
<code>leg = arg</code>	Specify the legend display settings, where <i>arg</i> can be: “def” - default, “n” - none, “s” - short, “det” - detailed.

Cellipse Examples

```
group gro1 age income cons taxes
gro1.scat cellipse
```

displays the 95% confidence ellipse around the means of the plots of INCOME, CONS, and TAXES against AGE.

```
gro1.scat cellipse(size=0.95 0.85 0.75)
```

displays the 95%, 85%, and 75% confidence ellipses, computed using the chi-square distribution

```
vector(3) sizes
sizes.fill 0.95, 0.85, 0.75
gro1.scat cellipse(size=sizes)
```

displays the same graph.

Chapter 4. Command Reference

Commands

The following list summarizes the EViews basic commands.

A list of views and procedures available for each object may be found in [Chapter 1. “Object View and Procedure Reference,” on page 3](#) of the *Command Reference*. Commands for working with matrix objects are listed in [Chapter 7. “Matrix Language Reference,” on page 839](#), and EViews programming expressions are described in [Chapter 8. “Programming Language Reference,” beginning on page 869](#).

Command Actions

doexecute action without opening window ([p. 714](#)).
freezecreate view object ([p. 724](#)).
printprint view ([p. 762](#)).
showshow object window ([p. 777](#)).

Global Commands

cdchange default directory ([p. 689](#)).
exitexit the EViews program ([p. 715](#)).
outputredirect printer output ([p. 740](#)).
paramset parameter values ([p. 761](#)).
rndseedset the seed of the random number generator ([p. 770](#)).
smplset current workfile sample ([p. 780](#)).

Object Creation Commands

alphaalpha series ([p. 6](#)).
coefcoefficient vector ([p. 16](#)).
equationequation object ([p. 47](#)).
factorfactor analysis object ([p. 103](#)).
frmlnumeric or alpha series object with a formula for auto-updating
([p. 725](#)).
genrnumeric or alpha series object ([p. 726](#)).
graphgraph object—create using a graph command or by merging exist-
ing graphs ([p. 158](#)).
groupgroup object ([p. 202](#)).
linkseries or alpha link object ([p. 225](#)).
logllikelihood object ([p. 240](#)).
matrixmatrix object ([p. 258](#)).

model..... model object (p. 284).
pool pool object (p. 318).
rowvector rowvector object (p. 343).
sample sample object (p. 351).
scalar scalar object (p. 353).
series numeric series (p. 375).
spool spool object (p. 421).
sspace sspace object (p. 446).
sym..... sym object (p. 471).
system..... system object (p. 503).
table..... table object (p. 530).
text text object (p. 535).
valmap valmap object (p. 541).
var var estimation object (p. 572).
vector vector object (p. 587).

Object Container, Data, and File Commands

ccopy copy series from DRI database (p. 688).
cfetch fetch series from DRI database (p. 691).
clabel display DRI series description (p. 692).
close close object, program, or workfile (p. 693).
copy copy objects within and between workfiles, workfile pages, and databases (p. 696).
db open or create a database (p. 707).
dbcopy make copy of a database (p. 708).
dbcreate create a new database (p. 708).
dbdelete delete a database (p. 709).
dbopen open a database (p. 709).
dbpack pack a database (p. 711).
dbrebuild..... rebuild a database (p. 711).
dbrename rename a database (p. 712).
dbrepair repair a database (p. 712).
driconvert..... convert the entire DRI database to an EViews database (p. 714).
expand expand workfile range (p. 715).
fetch..... fetch objects from databases or databank files (p. 717).
hconvert..... convert an entire Haver Analytics database to an EViews database (p. 731).
hfetch..... fetch series from a Haver Analytics database (p. 732).
hlabel..... obtain label from a Haver Analytics database (p. 733).

loadload a workfile (p. 735).
openopen a program or text (ASCII) file (p. 737).
pageappend.....append observations to workfile page (p. 742).
pagecontract.....contract workfile page (p. 744).
pagecopy.....copy contents of a workfile page (p. 744).
pagecreate.....create a workfile page (p. 746).
pagedelete.....delete a workfile page (p. 750).
pageloadload one or more pages into a workfile from a workfile or a foreign data source (p. 750).
pagerename.....rename a workfile page (p. 751).
pagesavesave page into a workfile or a foreign data source (p. 752).
pageselectmake specified page active (p. 753).
pagestackreshape the workfile page by stacking observations (p. 753).
pagestructapply a workfile structure to the page (p. 756).
pageunstack.....reshape the workfile page by unstacking observations into multiple series (p. 759).
rangereset the workfile range (p. 766).
readimport data from a foreign disk file into series (p. 766).
savesave workfile to disk (p. 773).
sortsort the workfile (p. 782).
storestore objects in database and databank files (p. 787).
unlink.....break links in series objects (p. 796).
wfcreatecreate a new workfile (p. 801).
wfopenopen workfile or foreign source data as a workfile (p. 802).
wfsavesave workfile to disk as a workfile or a foreign data source (p. 810).
wfselectchange active workfile page (p. 811).
workfilecreate or change active workfile (p. 812).
write.....write series to a disk file (p. 812).

Object Utility Commands

closeclose window of an object, program, or workfile (p. 693).
copy.....copy objects (p. 696).
delete.....delete objects (p. 713).
renamerename object (p. 769).

Object Assignment Commands

dataenter data from keyboard (p. 706).
frmlassign formula for auto-updating to a numeric or alpha series object (p. 725).
genrcreate numeric or alpha series object.(p. 726).

[randint](#)..... assign random integer values to object (p. 770).
[randseed](#) set random number generator seed (p. 770).

Graph Creation Commands

Graph creation is discussed in detail in “[Graph Creation Commands](#)” on page 601.

[area](#)..... area graph (p. 603).
[band](#) area band graph (p. 606).
[bar](#) bar graph (p. 609).
[boxplot](#)..... boxplot graph (p. 613).
[distplot](#)..... distribution graph (p. 615).
[dot](#) dot plot graph (p. 622).
[errbar](#)..... error bar graph (p. 626).
[hilo](#) high-low(-open-close) graph (p. 628).
[line](#) line-symbol graph (p. 630).
[pie](#) pie chart (p. 633).
[qqplot](#) quantile-quantile graph (p. 636).
[scat](#) scatterplot (p. 640).
[scatmat](#) matrix of all pairwise scatterplots (p. 644).
[scatpair](#) scatterplot pairs graph (p. 647).
[seasplot](#) seasonal line graph (p. 651).
[spike](#) spike graph (p. 652).
[xyarea](#) XY area graph (p. 656).
[xybar](#) XY bar graph (p. 659).
[xyline](#)..... XY line graph (p. 661).
[xypair](#) XY pairs graph (p. 664).

Table Commands

[setcell](#)..... format and fill in a table cell (p. 774).
[setcolwidth](#)..... set width of a table column (p. 776).
[setline](#) place a horizontal line in table (p. 776).

Programming Commands

[open](#)..... open a program file (p. 737).
[output](#) redirects print output to objects or files (p. 740).
[poff](#) turns off automatic printing in programs (p. 875).
[pon](#) turns on automatic printing in programs (p. 875).
[program](#) create a new program (p. 763).
[run](#)..... read data from a foreign disk file (p. 772).
[spawn](#) spawn a new process (p. 783).
[statusline](#)..... open a file (p. 785).

ticreset the timer (p. 790).
tocdisplay elapsed time (since timer reset) in seconds (p. 791).

Interactive Use Commands

archestimate autoregressive conditional heteroskedasticity (ARCH and GARCH) models (p. 680).
archtestLM test for the presence of ARCH in the residuals (p. 684).
autoBreusch-Godfrey serial correlation Lagrange Multiplier (LM) test (p. 685).
binarybinary dependent variable models (includes probit, logit, gompit) models (p. 686).
causepairwise Granger causality tests (p. 687).
censoredestimate censored and truncated regression (includes tobit) models (p. 689).
chowChow breakpoint and forecast tests for structural change (p. 691).
cointJohansen cointegration test (p. 694).
corcorrelation matrix (p. 702).
countcount data modeling (includes poisson, negative binomial and quasi-maximum likelihood count models) (p. 703).
covcovariance matrix (p. 704).
crosscross correlogram (p. 706).
facbreakfactor breakpoint test for stability (p. 715).
factestestimate a factor analysis model (p. 716).
fitstatic forecast from an equation (p. 720).
forecastdynamic forecast from an equation (p. 722).
gmmgeneralized method of moments estimation (p. 727).
histhistogram and descriptive statistics (p. 732).
hpfHodrick-Prescott filter (p. 734).
logitlogit (binary) estimation (p. 735).
lslinear and nonlinear least squares regression (includes weighted least squares and ARMAX) (p. 735).
orderedordinal dependent variable models (includes ordered probit, ordered logit, and ordered extreme value models) (p. 739).
probitprobit (binary) estimation (p. 763).
qregestimate a quantile regression specification (p. 764).
resetRamsey's RESET test for functional form (p. 769).
seasseasonal adjustment for quarterly and monthly time series (p. 774).
smoothexponential smoothing (p. 778).
solvesolve a model (p. 782).

stats	descriptive statistics (p. 785).
stepls	estimation by stepwise least squares (p. 786).
testadd	likelihood ratio test for adding variables to equation (p. 789).
testdrop	likelihood ratio test for dropping variables from equation (p. 790).
tsls	linear and nonlinear two-stage least squares (TSLS) regression models (includes weighted TSLS, and TSLS with ARMA errors) (p. 792).
ubreak	Andrews-Quandt test for unknown breakpoint (p. 795).
uroot	unit root test (p. 796).
varest	specify and estimate a VAR or VEC (p. 800).

Command Entries

The following section provides an alphabetical listing of commands. Each entry outlines the command syntax and associated options, and provides examples and cross references.

arch	Commands
-------------	--------------------------

Estimate generalized autoregressive conditional heteroskedasticity (GARCH) models.

Syntax

`arch(p,q,options) y [x1 x2 x3] [@ p1 p2 [@ t1 t2]]`

`arch(p,q,options) y = expression [@ p1 p2 [@ t1 t2]]`

The ARCH command 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, list them after the mean equation using an “@”-sign to separate the mean from the variance equation.

When estimating component ARCH models, you may specify exogenous variance regressors for the permanent and transitory components. 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 regressor.

Options

General Options

<code>egarch</code>	Exponential GARCH.
<code>parch[= <i>arg</i>]</code>	Power ARCH. If the optional <i>arg</i> is provided, the power parameter will be set to that value, otherwise the power parameter will be estimated.
<code>cgarch</code>	Component (permanent and transitory) ARCH.
<code>asy = <i>integer</i></code> (<i>default</i> = 1)	Number of asymmetric terms in the Power ARCH or EGARCH model. The maximum number of terms allowed is 9.
<code>thrsh = <i>integer</i></code> (<i>default</i> = 0)	Number of threshold terms for GARCH and Component models. The maximum number of terms allowed is 9. For Component models, “thrsh” must take a value of 0 or 1.
<code>archm = <i>arg</i></code>	ARCH-M (ARCH in mean) specification with the conditional standard deviation (“archm = sd”), the conditional variance (“archm = var”), or the log of the conditional variance (“archm = log”) entered as a regressor in the mean equation.
<code>tdist [= <i>number</i>]</code>	Estimate the model assuming that the residuals follow a conditional Student’s <i>t</i> -distribution (the default is the conditional normal distribution). Providing the optional number greater than two will fix the degrees of freedom to that value. If the argument is not provided, the degrees of freedom will be estimated.
<code>ged [= <i>number</i>]</code>	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.
<code>h</code>	Bollerslev-Wooldridge robust quasi-maximum likelihood (QML) covariance/standard errors. Not available when using the “tdist” or “ged” options.
<code>z</code>	Turn of backcasting for both initial MA innovations and initial variances.
<code>b</code>	Use Berndt-Hall-Hall-Hausman (BHHH) as maximization algorithm. The default is Marquardt.
<code>m = <i>integer</i></code>	Set maximum number of iterations.
<code>c = <i>scalar</i></code>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.

<code>s</code>	Use the current coefficient values in “C” as starting values (see also param (p. 761)).
<code>s = number</code>	Specify a number between zero and one to determine starting values as a fraction of preliminary LS estimates (out of range values are set to “s = 1”).
<code>showopts / -showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
<code>deriv = keyword</code>	Set derivative method. The argument <i>keyword</i> should be a one-letter string (“f” or “a” corresponding to fast or accurate numeric derivatives, respectively).
<code>coef = arg</code>	Specify the name of the coefficient vector of a system’s variance component; the default behavior is to use the “C” coefficient vector.
<code>backcast = n</code>	Backcast weight to calculate value used as the presample conditional variance. Weight needs to be greater than 0 and less than or equal to 1; the default value is 0.7. Note that a weight of 1 is equivalent to no backcasting, i.e. using the unconditional residual variance as the presample conditional variance.
<code>p</code>	Print estimation results.

EGARCH options

<code>parch [= arg]</code>	Power ARCH. If the optional <i>arg</i> is provided, the power parameter will be set to that value, otherwise the power parameter will be estimated.
<code>cgarch</code>	Component (permanent and transitory) ARCH.
<code>asy = int (default = 1)</code>	Number of asymmetric terms in the Power ARCH or EGARCH model. The maximum number of terms allowed is 9.
<code>thrsh = int (default = 0)</code>	Number of threshold terms for GARCH and Component models. The maximum number of terms allowed is 9. For Component models, thrsh must take a value of 0 or 1.
<code>archm = arg</code>	ARCH-M (ARCH in mean) specification, where <i>arg</i> is entered as a regressor in the mean equation and may be: “sd” (the conditional standard deviation), “var” (the conditional variance), “log” (the log of the conditional variance).

<code>tdist [= n]</code>	Estimate the model assuming that the residuals follow a conditional Student's t-distribution (default is the conditional normal distribution). Providing the optional number greater than two will fix the degrees of freedom to that value. If the argument is not provided, the degrees of freedom will be estimated.
<code>ged [= n]</code>	Estimate the model assuming that the residuals follow a conditional GED (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.
<code>vt</code>	variance target of the constant term.
<code>integrated</code>	Restrict GARCH model to be integrated, i.e. IGARCH.
<code>z</code>	Turn of backcasting for initial MA innovations.
<code>deriv = key</code>	Set derivative method. The <i>key</i> should be: “f” (fast numeric derivatives) or “a” (accurate numeric derivatives).

Saved results

Most of the results saved for the `ls` command are also available after ARCH estimation; see [ls \(p. 735\)](#) for details.

Examples

```
arch(4, 0, m=1000, h) sp500 c
```

estimates an ARCH(4) model with a mean equation consisting of the series SP500 regressed on a constant. The procedure will perform up to 1000 iterations, and will report Bollerslev-Wooldridge robust QML standard errors upon completion.

The commands:

```
c = 0.1
arch(thrsh=1, s, mean=var) @pch(nys) c ar(1)
```

estimate a TARCH(1, 1)-in-mean specification with the mean equation relating the percent change of NYS to a constant, an AR term of 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:

```
arch(1, 2, asy=0, parch=1.5, ged=1.2) dlog(ibm)=c(1)+c(2)*
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 consists of 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.

Cross-references

See [Chapter 29. “ARCH and GARCH Estimation,” on page 185](#) of the *User’s Guide II* for a discussion of ARCH models. See also [Equation::garch \(p. 53\)](#) and [Equation::makegarch \(p. 66\)](#).

archtest	Commands
----------	--------------------------

Test for autoregressive conditional heteroskedasticity (ARCH).

Carries out Lagrange Multiplier (LM) tests for ARCH in the residuals.

Note that a more general version of the ARCH test is available using [Equation::archtest \(p. 34\)](#).

Syntax

`archtest(options)`

Options

You must specify the order of ARCH for which you wish to test. The number of lags to be included in the test equation should be provided in parentheses after the `arch` keyword.

Other Options:

<code>p</code>	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 eq1.arch sp500 c
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 when performing an `archtest` after an `arch` estimation, EViews uses the standardized residuals (the residual of the mean equation divided by the estimated conditional standard deviation) to form the test.

Cross-references

See [“ARCH LM Test” on page 158](#) of the *User’s Guide II* for further discussion of testing ARCH and [Chapter 29. “ARCH and GARCH Estimation,” on page 185](#) of the *User’s Guide II* for a general discussion of working with ARCH models in EViews.

auto	Commands
------	--------------------------

Compute serial correlation LM (Lagrange multiplier) test.

Carries out Breusch-Godfrey Lagrange Multiplier (LM) tests for serial correlation in the estimation residuals from the default equation.

Syntax

`auto(order, options)`

You must specify the order of serial correlation for which you wish to test. You should specify the number of lags in parentheses after the `auto` keyword, followed by any additional options.

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 you may use the commands:

```
ls output c labor capital
auto(4)
```

The commands:

```
output(t) c:\result\artest.txt
equation eq1.ls cons c y y(-1)
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

See [“Serial Correlation LM Test” on page 65](#) of the *User’s Guide II* for further discussion of the Breusch-Godfrey test.

binary	Commands
--------	----------

Estimate binary dependent variable models.

Estimates models where the binary dependent variable Y is either zero or one (probit, logit, gompit).

Syntax

```
binary(options) y x1 [x2 x3 ...]  
binary(options) specification
```

Options

d = <i>arg</i> (default = "n")	Specify likelihood: normal likelihood function, probit ("n"), logistic likelihood function, logit ("l"), Type I extreme value likelihood function, Gompit ("x").
q (default)	Use quadratic hill climbing as the maximization algorithm.
r	Use Newton-Raphson as the maximization algorithm.
b	Use Berndt-Hall-Hall-Hausman (BHHH) for maximization algorithm.
h	Quasi-maximum likelihood (QML) standard errors.
g	GLM standard errors.
m = <i>integer</i>	Set maximum number of iterations.
c = <i>scalar</i>	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 = <i>number</i>	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 using a constant, WAGE, EDU, and KIDS, and computing QML standard errors, you may use the command:

```
binary(d=l,h) y c wage edu kids
```

Note that this estimation uses the default global optimization options. The commands:

```
param c(1) .1 c(2) .1 c(3) .1
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:

```
coef beta_probit = @coefs
matrix cov_probit = @coefcov
```

store the estimated coefficients and coefficient covariances in the coefficient vector BETA_PROBIT and matrix COV_PROBIT.

Cross-references

See [“Binary Dependent Variable Models” on page 209](#) of the *User’s Guide II* for additional discussion.

cause	Commands
-------	----------

Granger causality test.

Performs pairwise Granger causality tests between (all possible) pairs of the listed series or group of series.

Syntax

```
cause(n, options) ser1 ser2 ser3
```

Following the keyword, list the series or group of series for which you wish to test for Granger causality.

Options

You must specify the number of lags *n* to use for the test by providing an integer in parentheses after the 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

To compute Granger causality tests of whether GDP Granger causes M1 and whether M1 Granger causes GDP, you may enter the command:

```
cause(4) gdp m1
```

The regressors of each test are a constant and four lags of GDP and M1.

```
cause(12,p) m1 gdp r
```

prints the result of six pairwise Granger causality tests for the three series. 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).

Cross-references

See [“Granger Causality” on page 410](#) of the *User’s Guide I* for a discussion of Granger’s approach to testing hypotheses about causality.

See also [Var::var \(p. 572\)](#).

ccopy	Commands
--------------	--------------------------

Copy one or more series from the DRI Basic Economics database to EViews data bank (.DB) files.

You must have the DRI database installed on your computer to use this feature.

Syntax

```
ccopy series_name
```

Type the name of the series or wildcard expression for series you want to copy after the `ccopy` keyword. The data bank files will be stored in the default directory with the same name as the series names in the DRI database. You can supply path information to indicate the directory for the data bank files.

Examples

The command:

```
ccopy lhur
```

copies the DRI series LHUR to LHUR.DB file in the default path directory.

```
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 \(p. 691\)](#), [clabel \(p. 692\)](#), [store \(p. 787\)](#), [fetch \(p. 717\)](#).

cd	Commands
----	----------

Change default directory.

The `cd` command changes the current default working directory. The current working directory is displayed in the “Path = ...” message in the bottom right of the EViews window.

Syntax

`cd path_name`

Examples

To change the default directory to “SAMPLE DATA” in the A: drive, you may issue the command:

```
cd "a:\sample data"
```

Notice that the directory name is surrounded by double quotes. If your name does not contain spaces, you may omit the quotes. The command:

```
cd a:\test
```

changes the default directory to A:\TEST.

Subsequent save operations will save into the default directory, unless you specify a different directory at the time of the operation.

Cross-references

See [Chapter 3. “Workfile Basics,” on page 37](#) of the *User’s Guide I* for further discussion of basic operations in EViews.

See also [wfsave \(p. 810\)](#), [pagesave \(p. 752\)](#), and [save \(p. 773\)](#).

censored	Commands
----------	----------

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

`censored(options) y x1 [x2 x3]`

`censored(options) specification`

Options

<i>l = number</i> (<i>default = 0</i>)	Set value for the left censoring limit.
<i>r = number</i> (<i>default = none</i>)	Set value for the right censoring limit.
<i>l = series_name, i</i>	Set series name of the indicator variable for the left censoring limit.
<i>r = series_name, i</i>	Set series name of the indicator variable for the right censoring limit.
<i>t</i>	Estimate truncated model.
<i>d = arg</i> (<i>default = "n"</i>)	Specify error distribution: normal ("n"), logistic ("l"), Type I extreme value ("x").
<i>q (default)</i>	Use quadratic hill climbing as the maximization algorithm.
<i>r</i>	Use Newton-Raphson as the maximization algorithm.
<i>b</i>	Use Berndt-Hall-Hausman for maximization algorithm.
<i>h</i>	Quasi-maximum likelihood (QML) standard errors.
<i>g</i>	GLM standard errors.
<i>m = integer</i>	Set maximum number of iterations.
<i>c = scalar</i>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.
<i>s</i>	Use the current coefficient values in C as starting values.
<i>s = number</i>	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").
<i>showopts / -showopts</i>	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
<i>p</i>	Print results.

Examples

The command:

```
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, no right censoring, and the quadratic hill climbing algorithm.

Cross-references

See [Chapter 30. “Discrete and Limited Dependent Variable Models,”](#) on page 209 of the *User’s Guide II* for discussion of censored and truncated regression models.

cfetch	Commands
--------	--------------------------

Fetch a series from the DRI Basic Economics database into a workfile.

`cfetch` reads one or more series from the DRI Basic Economics Database into the active workfile. *You must have the DRI database installed on your computer to use this feature.*

Syntax

`cfetch series_name`

Examples

```
cfetch 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 106 of the *User’s Guide I*.

See also [ccopy](#) (p. 688), [clabel](#) (p. 692), [store](#) (p. 787), [fetch](#) (p. 717).

chdir	Commands
-------	--------------------------

Change default directory.

See [cd](#) (p. 689).

chow	Commands
------	--------------------------

Chow test for stability.

Carries out Chow breakpoint or Chow forecast tests for parameter constancy.

Syntax

```
chow(options) obs1 [obs2 obs3 ...] @ x1 x2 x3
```

You must provide the breakpoint observations (using dates or observation numbers) to be tested. To specify more than one breakpoint, separate the breakpoints by a space. For the

Chow breakpoint test, if the equation is specified by list and contains no linear terms, you may specify a subset of the regressors to be tested for a breakpoint after an “@” sign.

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 1970Q1 1980Q1
```

perform a regression of M1 on a constant, GDP, and CPI with first order autoregressive errors, and employ a Chow breakpoint test to determine whether the parameters before the 1970’s, during the 1970’s, and after the 1970’s are “stable”.

To regress the log of SPOT on a constant, the log of P_US, and the log of P_UK, and to carry out the Chow forecast test starting from 1973, enter the commands:

```
ls log(spot) c log(p_us) log(p_uk)
chow(f) 1973
```

To test whether only the constant term and the coefficient on the log of P_US prior to and after 1970 are “stable” enter the commands:

```
chow 1970 @ c log(p_us)
```

Cross-references

See [“Chow’s Breakpoint Test” on page 165](#) of the *User’s Guide II* for further discussion.

See also [Equation::facbreak](#) (p. 48) and [Equation::rls](#) (p. 84).

clabel	Commands
--------	--------------------------

Display a DRI Basic Economics database series description.

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.

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.*

Syntax

```
clabel series_name
```

Examples

```
clabel lhur
```

displays the description of the DRI series LHUR on the status line.

Cross-references

See also [ccopy](#) (p. 688), [cfetch](#) (p. 691), [read](#) (p. 766), [fetch](#) (p. 717).

close	Commands
-------	--------------------------

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

```
close object_name
```

Examples

```
close gdp graph1 table2
```

closes the three objects GDP, GRAPH1, and TABLE2.

```
lwage.hist
```

```
close lwage
```

opens the LWAGE window and displays the histogram view of LWAGE, then closes the window.

Cross-references

See [Chapter 1. “Introduction,” on page 5](#) of the *User’s Guide I* for a discussion of basic control of EViews.

coint	Commands
-------	----------

Johansen’s cointegration test.

Syntax

```
coint(test_option,n,option) y1 y2 [y3 ...]  
coint(test_option,n,option) y1 y2 [y3 ...] [@ x1 x2 x3 ...]
```

Enter the `coint` keyword followed by a list of series or group names for you wish to test for cointegration. Each name should be separated by a space. To use exogenous variables, such as seasonal dummy variables, in the test, list the names after an “@”-sign.

As of EViews 5, the output for cointegration tests displays *p*-values for the rank test statistics. These *p*-values are computed using the response surface coefficients as estimated in MacKinnon, et. al. (1999). The 0.05 critical values are now based on the response surface coefficients from MacKinnon-Haug-Michelis. *Note: the reported critical values assume no exogenous variables other than an intercept and trend.*

Options

You must specify the test option followed by the number of lags *n*. You must choose one of the following six test options:

a	No deterministic trend in the data, and no intercept or trend in the cointegrating equation.
b	No deterministic trend in the data, and an intercept but no trend in the cointegrating equation.
c	Linear trend in the data, and an intercept but no trend in the cointegrating equation.
d	Linear trend in the data, and both an intercept and a trend in the cointegrating equation.
e	Quadratic trend in the data, and both an intercept and a trend in the cointegrating equation.
s	Summarize the results of all 5 options (a-e).

Other Options:

<code>restrict</code>	Impose restrictions as specified by the <code>append (coint)</code> proc.
<code>m = integer</code>	Maximum number of iterations for restricted estimation (only valid if you choose the <code>restrict</code> option).
<code>c = scalar</code>	Convergence criterion for restricted estimation. (only valid if you choose the <code>restrict</code> option).
<code>save = mat_name</code>	Stores test statistics as a named matrix object. The <code>save =</code> option stores a $(k + 1) \times 4$ matrix, where k is the number of endogenous variables in the VAR. The first column contains the eigenvalues, the second column contains the maximum eigenvalue statistics, the third column contains the trace statistics, and the fourth column contains the log likelihood values. The i -th row of columns 2 and 3 are the test statistics for rank $i - 1$. The last row is filled with NAs, except the last column which contains the log likelihood value of the unrestricted (full rank) model.
<code>cvtype = ol</code>	Display 0.05 and 0.01 critical values from Osterwald-Lenum (1992). This option reproduces the output from version 4. The default is to display critical values based on the response surface coefficients from MacKinnon-Haug-Michelis (1999). Note that the argument on the right side of the equals sign are letters, not numbers 0-1).
<code>cvsz = arg</code> <i>(default = 0.05)</i>	Specify the size of MacKinnon-Haug-Michelis (1999) critical values to be displayed. The size must be between 0.0001 and 0.9999; values outside this range will be reset to the default value of 0.05. This option is ignored if you set <code>cvtype = ol</code> .
<code>p</code>	Print output of the test.

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.

Cross-references

See [“Cointegration Testing” on page 363](#) of the *User’s Guide II* for details on the Johansen test.

copy	Commands
------	----------

Copy an object, or a set of objects matching a name pattern, within and between workfiles, workfile pages, and databases. Data in series objects may be frequency converted or match merged.

Syntax

```
copy(options) src_spec dest_spec [src_id dest_id]
copy(options) src_spec dest_spec [@src src_ids @dest dest_id]
```

where *src_spec* and *dest_spec* are of the form:

```
[ctype][container::][page\]object_name
```

There are three parts to the `copy` command: (1) a specification of the location and names of the source objects; (2) a specification of the location and names of the destination objects; (3) optional source and destination IDs if the copy operation involves match merging.

The source and destination objects are specified in multiple (optional) parts: (1) the *container* specification is the name of a workfile or database; (2) the *page* specification is the name of a page within a workfile or a subdirectory within a database; and (3) the *object_name* specification is the name of an object or a wildcard pattern corresponding to multiple objects.

The *ctype* specification is rarely required, but permits you to specify precisely your source or destination in cases where a database and workfile share the same name. In this case, *ctype* may be used to indicate the container to which you are referring by prefixing the container name with “:” to indicate the workfile, or “::” to indicate the database with the common name.

When parts of the source or destination specification are not provided, EViews will fill in default values where possible. The default container is the active workfile, unless the “::” prefix is used in which case the default container is the default database. The default page within a workfile is always the active page. The default name for the destination object is the name of the object within the source container.

If ID series are not provided in the command, then EViews will perform frequency conversion when copying data whenever the source and destination containers have different frequencies. If ID series are provided, then EViews will perform a general match merge between the source and destination using the specified ID series. In the case where you wish to copy your data using match merging with special treatment for date matching, you must use the special keyword “@DATE” as an ID series for the source or destination. If “@DATE” is not specified as an identifier in either the source or destination IDs, EViews will perform an exact match merge using the given identifiers.

If ID series are not specified, but a conversion option requiring a general match merge is used (e.g., “c = med”), “@DATE @DATE” will be appended to the list of IDs and a general date match merge will be employed.

See [Link::linkto](#) (p. 226) for additional discussion of the differences embodied in these choices.

The general syntax described above covers all possible uses of the `copy` command. The following paragraphs provide examples of the specific syntax used for some common cases of the command.

Copying Within a Workfile

Copy an object within the default workfile page as a new object with a different name:

- `copy(options) src_name dest_name`

Copy an object from the `src_page` page into the default workfile page using the specified name:

- `copy(options) src_page\src_name dest_name`

Copy an object from the `src_page` page into the `dest_page` page, keeping the same name:

- `copy(options) src_page\src_names dest_page\`

Copy an object from the `src_page` page to the default workfile page, match merging any series data using a single `src_id` and a single `dest_id` identifier series:

- `copy(options) src_page\src_name dest_name src_id dest_id`

Copy an object from the `src_page` page to the `dest_page` match merging any series data using multiple source and destination identifier series:

- `copy(options) src_page\src_name dest_page\dest_name @src src_id1 src_id2 ... src_id_n @dest dest_id1 dest_id2 ... dest_id_n`

Copying Between Containers (Workfiles and Databases)

Copy one or more objects from the `src_page` of the workfile `src_workfile` to the `dest_page` of the workfile `dest_workfile`, using the name or name pattern given in `src_names`:

- `copy(options) src_workfile::src_page\src_names dest_workfile::dest_page\`

Copy an object from database `src_database` to the default page in the container `dest_container`:

- `copy(options) src_database::src_name dest_container::dest_name`

Note that if both a workfile and database exist matching the name provided in *dest_container*, EViews will favor the workfile unless the “::” prefix is used to specify explicitly that the database should be used.

Options

Basic Options

overwrite / o	Overwrite any existing object with the destination name in the destination container. Error only if a non-editable series is encountered in the destination location.
merge / m	If the source object is a series, merge the data from the source series into any existing destination series, preserving any values in the destination series that are not present in the source. For all other object types, overwrite any existing object with the source object. Error if a non-editable series is encountered in the destination location.
protect / p	Protect objects in the destination location from overwriting or merging. If there is an existing object in the destination container, cancel the copy operation for that object, but do not generate an error.
noerr	Suppress errors that are generated during the copy. For example, if the overwrite option is used, suppress any error caused by attempting to overwrite a non-editable series such as an index series used in the workfile structure.
link	When copying from a database, copy as a database link.

Group Copy Options

When copying a group object from workfile to database:

<i>g = arg</i>	Method for copying group objects from a 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).
----------------	---

When copying a group object from a database to a workfile:

<i>g = arg</i>	Method for copying group objects from a database or workfile to a workfile: “b” (copy both group definition and series), “d” (copy only the series), “l” (copy only the group definition).
----------------	--

Note that copying a group object containing expressions or auto-updating series between workfiles only copies the expressions, and not the underlying series.

Frequency Conversion Options

If the `copy` command does not specify identifier series, EViews will perform frequency conversion of the data contained in series objects whenever the source and destination containers do not have the same frequency.

The following options control the frequency conversion method when copying series and group objects into a workfile page and converting from *low* to *high* frequency:

<code>c = arg</code>	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 into a workfile page and converting from *high* to *low* frequency:

<code>c = arg</code>	<p><i>High to low conversion methods removing NAs:</i> “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).</p> <p><i>High to low conversion methods propagating NAs:</i> “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).</p>
----------------------	--

Note that if no conversion method is given in the command, the conversion method specified within the series object will be used as the default. If the series does not contain an explicit conversion method, the global option settings will be used to determine the method.

Match Merge Options

These options are available when ID series are specified in the `copy` command.

<code>smp1 = smp1_spec</code>	Sample to be used when computing contractions during copying using match merge. Either provide the sample range in double quotes or specify a named sample object. By default, EViews will use the entire workfile sample “@ALL”.
<code>c = arg</code>	Set the match merge contraction method. If you are copying 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 copying an alpha series, only the non-summary methods “max”, “min”, “obs”, “nas”, “first”, “last”, “unique” and “none” are supported. For copying of numeric series, the default contraction method is “c = mean”; for copying of alpha series, the default is “c = unique”.
<code>quant = number</code>	Quantile value to be used when contracting using the “c = quant” option (e.g., “quant = .3”).
<code>nacat</code>	Treat “NA” values as a category when copying using general match merge operations.

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 copy 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, copying with match merging will repeat the value of the source for every observation with matching identifier values in 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

```
copy good_equation best_equation
```


makes an exact copy of GOOD_EQUATION and names it BEST_EQUATION.

```
copy graph_1 wf2::wkly\graph1
```

copies GRAPH_1 from the default page of the current workfile to GRAPH1 in the page WKLY of the workfile WF2.

```
copy gdp usdat::
```

copies GDP from the current workfile to GDP in the USDAT database or workfile.

```
copy ::gdp macro1::gdp_us
```

copies GDP from the default database to either the open workfile MACRO1, or the database named MACRO1 if there is no open workfile with that name. If there is an open workfile MACRO1 you may use

```
copy ::gdp ::macro1::gdp_us
```

to specify explicitly that you wish to write to the MACRO1 database.

```
copy(smpl="1990 2000") page1\pop page2\ @src county @date @dest  
county @date
```

copies POP data for 1990 through 2005 from PAGE1 to PAGE2, match merge using the ids COUNTY and the date structure of the two pages.

```
copy(smpl="1990 2000", c=mean) panelpage\inc countypage\ county  
county
```

copies the INC data from the PANELPAGE to the COUNTYPAGE, match merging using the values of the COUNTY series, and contracting the panel data by computing means for each county using the specified sample.

```
copy countypage\pop panelpage\ county county
```

match merges the POP data from the COUNTYPAGE to the PANELPAGE using the values of the COUNTY series.

```
copy(c=x, merge) quarterly::page1\ser* annual::page6\*
```

copies all objects with names beginning with “SER” on page PAGE1 of workfile QUARTERLY into page PAGE6 of workfile ANNUAL using the existing names. Series objects with data that can be (high-to-low) frequency converted will take the maximum value within a low-frequency period as the conversion method. If destination series already exist with the same name as the source series, the data will be merged. If destination objects (non-series) exist with the same name as source series, they will be overwritten.

Note that since databases are read from disk, you may provide a path for the database in the container specification, as in:

```
copy "c:\my_data\dba.edb::ser01" ser02
```

which copies the object SER01 from the database DBA.EDB located in the path “C:\MY DATA\” to SER02 in the default workfile page.

```
copy gd* "c:\my data\findat::"
```

makes a duplicate of all objects in the default page of the current workfile with names starting with “GD” to the database FINDAT in the root of “C:\MY DATA\”.

Cross-references

See “Copying Objects” on page 265 of the *User’s Guide I* for a discussion of copying and moving objects.

See also [fetch](#) (p. 717), [store](#) (p. 787), and [Link::linkto](#) (p. 226).

cor	Commands
-----	----------

Compute Pearson product-moment (ordinary) correlations for the specified series or groups.

Syntax

```
cor(options) arg1 [arg2 arg3...]
```

where *arg1*, *arg2*, etc. are the names of series or groups.

Note that this command is a limited feature version of the group view [Group::cor](#) (p. 190).

Options

wgt = <i>name</i> (optional)	Name of series containing weights.
wgtmethod = <i>arg</i> (default = “sstdev”)	Weighting method (when weights are specified using “weight =”): frequency (“freq”), inverse of variances (“var”), inverse of standard deviation (“stdev”), scaled inverse of variances (“svar”), scaled inverse of standard deviations (“sstdev”).
pairwise	Compute using pairwise deletion of observations with missing cases (pairwise samples).
out = <i>name</i>	Basename for saving output. All results will be saved in Sym matrices named using the string “CORR”, appended to the basename (e.g., the correlation specified by “out = my” is saved in the Sym matrix “MYCORR”).
p	Print the result.

Examples

```
cor height weight age
```

displays a 3×3 Pearson correlation matrix for the three series HEIGHT, WEIGHT, and AGE.

```
cor(pairwise) height weight age
```

computes the correlation using samples with pairwise missing value exclusion.

```
cor(out=aa) height weight age
```

computes the Pearson correlation for the series and saves the results in the symmetric matrix object AACORR.

Cross-references

See also [cov](#) (p. 704), [@cor](#) (p. 847), and [@cov](#) (p. 847).

count	Commands
-------	--------------------------

Estimates models where the dependent variable is a nonnegative integer count.

Syntax

```
count(options) y x1 [x2 x3...]  
count(options) specification
```

Follow the `count` keyword by the name of the dependent variable and a list of regressors.

Options

<code>d = arg</code> (default = "p")	Likelihood specification: Poisson likelihood ("p"), normal quasi-likelihood ("n"), exponential likelihood ("e"), negative binomial likelihood or quasi-likelihood ("b").
<code>v = positive_num</code> (default = 1)	Specify fixed value for QML parameter in normal and negative binomial quasi-likelihoods.
<code>q</code> (default	Use quadratic hill-climbing as the maximization algorithm.
<code>r</code>	Use Newton-Raphson as the maximization algorithm.
<code>b</code>	Use Berndt-Hall-Hall-Hausman as the maximization algorithm.
<code>h</code>	Quasi-maximum likelihood (QML) standard errors.
<code>g</code>	GLM standard errors.
<code>m = integer</code>	Set maximum number of iterations.

<i>c = scalar</i>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.
<i>s</i>	Use the current coefficient values in C as starting values.
<i>s = number</i>	Specify a number between zero and one to determine starting values as a fraction of the EViews default values (out of range values are set to “s = 1”).

Examples

The command:

```
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.

```
count arrest c job police
makeresid(g) res_g
```

estimates a Poisson count model of ARREST on a constant, JOB, and POLICE, and stores the generalized residuals in the series RES_G.

```
count(d=p) y c x1
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.

```
count(d=p, h) y c x1
```

estimates the same model with QML standard errors and covariances.

Cross-references

See [“Count Models” on page 246](#) of the *User’s Guide II* for additional discussion.

COV	Commands
-----	--------------------------

Compute Pearson product-moment (ordinary) covariances for the specified series or groups.

Syntax

```
cor(options) arg1 [arg2 arg3...]
```

where *arg1*, *arg2*, *etc.* are the names of series or groups.

Note that this command is a limited feature version of the group view `Group::cov` ([p. 194](#)).

Options

<code>wgt = name</code> (optional)	Name of series containing weights.
<code>wgtmethod = arg</code> (default = “sstdev”)	Weighting method (when weights are specified using “weight = ”): frequency (“freq”), inverse of variances (“var”), inverse of standard deviation (“stdev”), scaled inverse of variances (“svar”), scaled inverse of standard deviations (“sstdev”).
<code>pairwise</code>	Compute using pairwise deletion of observations with missing cases (pairwise samples).
<code>df</code>	Compute covariances with a degree-of-freedom correction for the mean.
<code>outfmt = arg</code> (default = “sheet”)	Output format: list (“list”), spreadsheet (“sheet”).
<code>out = name</code>	Baseline for saving output. All results will be saved in Sym matrices named using the string “COV”, appended to the baseline (e.g., the covariance specified by “out = my” is saved in the Sym matrix “MYCOV”).
<code>p</code>	Print the result.

Examples

```
cov height weight age
```

displays a 3×3 Pearson covariance matrix for the three series HEIGHT, WEIGHT, and AGE.

```
cov(pairwise) height weight age
```

computes the covariance using samples with pairwise missing value exclusion.

```
cov(out=aa) height weight age
```

computes the Pearson covariance for the series and saves the results in the symmetric matrix object AACOV.

Cross-references

See also [cor](#) (p. 702), [@cor](#) (p. 847), and [@cov](#) (p. 847).

create	Commands
--------	----------

Create workfile.

This command has been replaced by [wfccreate](#) (p. 801) and [pagecreate](#) (p. 746).

cross	Commands
-------	----------

Displays cross correlations (correlograms) for a pair of series.

Syntax

```
cross(n,options) ser1 ser2 [ser3 ...]
```

You must specify the number of lags *n* to use in computing the cross correlations as the first option. EViews will create an untitled group from the specified series and groups, and will display the cross correlation view for the group.

Options

The following options may be specified inside the parentheses after the number of lags:

p	Print the cross correlogram.
---	------------------------------

Examples

```
cross(36) log(m1) dlog(cpi)
```

displays the cross correlogram between the log of M1 and the first difference of the log of CPI, using up to 36 leads and lags.

```
equation eq1.arch sp500 c
eq1.makesresid(s) res_std
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 squared standardized residual and the standardized residual, up to 24 leads and lags. This correlogram provides a rough check of asymmetry in the ARCH effect.

Cross-references

See [“Cross Correlations and Correlograms”](#) on page 409 of the *User’s Guide I* for discussion.

data	Commands
------	----------

Enter data from keyboard.

Opens an unnamed group window to edit one or more series.

Syntax

```
data arg1 [arg2 arg3 ...]
```

Follow the `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 96](#) of the *User’s Guide I* for a discussion of the process of entering data from the keyboard.

db	Commands
----	--------------------------

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

```
db(options) [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 may 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* or 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 261](#) of the *User’s Guide I* for additional discussion.

Options

See [dbopen \(p. 709\)](#) 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 [Chapter 10. “EViews Databases,” on page 257](#) of the *User’s Guide I* for a discussion of EViews databases.

See also [dbcreate](#) (p. 708) and [dbopen](#) (p. 709).

dbcopy	Commands
--------	--------------------------

Make a copy of an existing database.

Syntax

`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 from or to 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 [Chapter 10. “EViews Databases,” on page 257](#) of the *User’s Guide I* for a discussion of EViews databases.

See also [dbrename](#) (p. 712) and [dbdelete](#) (p. 709).

dbcreate	Commands
----------	--------------------------

Create a new database.

Syntax

`dbcreate [path/]db_name [as shorthand_name]`

Follow the `dbcreate` keyword by a name for the new database. You may 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* or 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 261](#) of the *User’s Guide I* for additional discussion.

Examples

```
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](#) (p. 709) or [db](#) (p. 707).

Cross-references

See [Chapter 10. “EViews Databases,” on page 257](#) of the *User’s Guide I* for a discussion of EViews databases.

See also [db](#) (p. 707) and [dbopen](#) (p. 709).

dbdelete	Commands
----------	--------------------------

Delete an existing database (all files associated with the specified database).

Syntax

```
dbdelete [path\]db_name
```

Follow the `dbdelete` keyword by the name of the database to be deleted. You may include a path name to delete a database not in the default path.

Examples

```
dbdelete c:\temp\testdat
```

deletes all files associated with the TESTDAT database in the specified directory.

Cross-references

See [Chapter 10. “EViews Databases,” on page 257](#) of the *User’s Guide I* for a discussion of EViews databases.

See also [dbcopy](#) (p. 708) and [dbdelete](#) (p. 709).

dbopen	Commands
--------	--------------------------

Open an existing database.

Syntax

```
dbopen(options) [path\]db_name [as shorthand_name]
```

Follow the `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 *shorthand_name* or 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 *shorthand_name* will be assigned automatically. See [“Database Short-hands” on page 261](#) of the *User’s Guide I* 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

<code>type = arg, t = arg</code>	Specify the database type: AREMOS-TSD (“a”, “aremos”, “tsd”), DRIBase (“b”, “dribase”), EViews (“e”, “evdb”), FAME (“f”, “fame”), GiveWin/PcGive (“g”, “give”), Haver Analytics (“h”, “haver”), Rats Portable/Troll (“l”, “trl”), RATS 4.x (“r”, “rats”), TSP portable (“t”, “tsp”), EcoWin (“ecowin”).
----------------------------------	---

The following options may be required when connecting to a remote server:

<code>s = server_id,</code> <code>server = server_id</code>	Server name.
<code>u = user,</code> <code>username = user</code>	Username.
<code>p = pswd,</code> <code>password = pswd</code>	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 \(p. 707\)](#) or [dbcreate \(p. 708\)](#) to create a new database.

Cross-references

See [Chapter 10. “EViews Databases,” on page 257](#) of the *User’s Guide I* for a discussion of EViews databases.

See also [db \(p. 707\)](#) and [dbcreate \(p. 708\)](#).

dbpack	Commands
--------	--------------------------

Pack an existing database.

Syntax

```
dbpack [path\]db_name
```

Follow the `dbpack` keyword by a database name. You may 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 283](#) of the *User’s Guide I* for additional discussion.

See also [dbrebuild \(p. 711\)](#) and [dbrepair \(p. 712\)](#)

dbrebuild	Commands
-----------	--------------------------

Rebuild an existing database.

Rebuild a seriously damaged database into a new database file.

Syntax

```
dbrebuild [path\]source_name [path\]dest_name
```

Follow the `dbrebuild` keyword by the name of the database to be rebuilt, and then a new database name.

Examples

If you issue the command:

```
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 `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 283](#) of the *User’s Guide I* for a discussion.

See also [dbpack \(p. 711\)](#) and [dbrepair \(p. 712\)](#).

dbrename	Commands
----------	--------------------------

Rename an existing database.

`dbrename` renames all files associated with the specified database.

Syntax

`dbrename [path\]old_name [path\]new_name`

Follow the `dbrename` keyword with the current name of an existing database and the new name for the database.

Examples

```
dbrename testdat mydat
```

Renames all files associated with the TESTDAT database in the specified directory to MYDAT in the default directory.

Cross-references

See [Chapter 10. “EViews Databases,” on page 257](#) of the *User’s Guide I* for a discussion of EViews databases.

See also [db \(p. 707\)](#) and [dbcreate \(p. 708\)](#). See also [dbcopy \(p. 708\)](#) and [dbdelete \(p. 709\)](#).

dbrepair	Commands
----------	--------------------------

Repair an existing database.

This command is used to repair a damaged database.

Syntax

`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 so damaged that it cannot be repaired, you may have to rebuild it into a new database using the [dbrebuild](#) (p. 711) command. See “[Maintaining the Database](#)” on [page 283](#) of the *User’s Guide I* for a discussion of EViews database maintenance.

See also [dbpack](#) (p. 711) and [dbrebuild](#) (p. 711).

delete	Commands
--------	--------------------------

Deletes objects from a workfile or a database.

Syntax

```
delete arg1 [arg2 arg3 ...]
```

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.

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”, you may use the command:

```
delete temp*
```

To delete the objects CONS and INVEST from the database MACRO1, use:

```
delete macro1::cons macro1::invest
```

Cross-references

See [Chapter 4. “Object Basics,”](#) on [page 63](#) of the *User’s Guide I* for a discussion of working with objects, and [Chapter 10. “EViews Databases,”](#) on [page 257](#) of the *User’s Guide I* for a discussion of EViews databases.

do	Commands
----	----------

Execute without opening window.

Syntax

do *procedure*

do is most useful in EViews programs where you wish to run a series of commands without opening windows in the workfile area.

Examples

```
output(t) c:\result\junk1
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 without opening a window, and prints the results to the disk file.

Cross-references

See also [show \(p. 777\)](#).

driconvert	Commands
------------	----------

Convert the entire DRI Basic Economics database into an EViews database.

You must create an EViews database to store the converted DRI data *before* you use this command. This command may be very time-consuming.

Syntax

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 may include a path name to specify a database not in the default path.

Examples

```
dbcreate dribasic
driconvert dribasic
driconvert c:\mydata\dridbase
```

The first line creates a new (empty) database named DRIBASIC in the default directory. The second line copies all the data in the DRI Basic Economics database into in the DRIBASIC database. The last example copies the DRI data into the database DRIDBASE that is located in the C:\MYDATA directory.

Cross-references

See [Chapter 10. “EViews Databases,” on page 257](#) of the *User’s Guide I* for a discussion of EViews databases.

See also [dbcreate \(p. 708\)](#) and [db \(p. 707\)](#).

exit	Commands
------	--------------------------

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 workfiles, if desired, since all changes that you do not save to a disk file will be lost.

Syntax

exit

Cross-references

See also [close \(p. 693\)](#) and [save \(p. 773\)](#).

expand	Commands
--------	--------------------------

Expand a workfile.

No longer supported. See the replacement command [pagestruct \(p. 756\)](#).

facbreak	Commands
----------	--------------------------

Factor breakpoint test for stability.

Carries out a factor breakpoint test for parameter constancy.

Syntax

facbreak(options) ser1 [ser2 ser3 ...] @ x1 x2 x3

You must provide one or more series to be used as the factors with which to split the sample into categories. To specify more than one factor, separate the factors by a space. If the equation is specified by list and contains no linear terms, you may specify a subset of the regressors to be tested for a breakpoint after an “@” sign.

Options

p	Print the result of the test.
---	-------------------------------

Examples

The commands

```
ls log(spot) c log(p_us) log(p_uk)
facbreak season
```

perform a regression of the log of SPOT on a constant, the log of P_US, and the log of P_UK, and employ a factor breakpoint test to determine whether the parameters are stable through the different values of SEASON.

To test whether only the constant term and the coefficient on the log of P_US are “stable” enter the commands:

```
facbreak season @ c log(p_us)
```

Cross-references

See [“Factor Breakpoint Test” on page 152](#) of the *User’s Guide II* for further discussion.

See also [Equation::\[facbreak\]\(#\) \(p. 48\)](#) and [Equation::\[rls\]\(#\) \(p. 84\)](#).

factest	Commands
----------------	--------------------------

Specify and estimate a factor analysis model.

Syntax

```
factest(method = arg, options) x1 [x2 x3...] [@partial z1 z2 z3...]
factest(method = arg, options) matrix_name [[obs] [conditioning]] [@ name1 name2
name3...]
```

where:

method = arg (default = “ml”)	Factor estimation method: “ml” (maximum likelihood), “gls” (generalized least squares), “ipf” (iterated principal factors), “pace” (non-iterative partitioned covariance matrix estimation), “pf” (principal factors), “uls” (unweighted least squares)
----------------------------------	---

and the available options are specific to the factor estimation method (see [“Factor Methods” on page 97](#)).

The `factest` command allows you to estimated a factor analysis model without first declaring a factor object and then applying an estimation method. It provides a convenient method of interactively estimating transitory specifications that are not to be named and saved with the workfile.

Estimation of a factor analysis specification using `factest` only differs from estimation using a named factor and a factor estimation procedure (e.g., `Factor::ipf` (p. 109)) in the use of the “method =” option and in the fact that the command results in an unnamed factor object.

Examples

The command:

```
factest(method=gl) g1
```

estimates a factor analysis model for the series in G1 using GLS. The result is an unnamed factor object. (Almost) equivalently, we may declare and estimate the factor analysis object using the `Factor::gl` (p. 105) estimation procedure:

```
factor f1.gl g1
```

only in the fact that the former yields an unnamed factor object and the latter saves the object F1 in the workfile.

The command:

```
factest(method=ml) group01 @partial ser1 ser2
```

estimates the factor model using the partial correlation for the series in GROUP01, conditional on the series SER1 and SER2. The command is equivalent to:

```
factor f2.ml group01 @partial ser1 ser2
```

except the latter names the factor object F2.

Cross-references

See `Factor::gl` (p. 105), `Factor::ipf` (p. 109), `Factor::ml` (p. 117), `Factor::pf` (p. 125), and `Factor::uls` (p. 138).

fetch	Commands
-------	----------

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 using the object in the database or using the databank file name.

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 an attached conversion method, EViews will use the method set by **Options/Date-Frequency** in the main menu. You can override the conversion method by specifying an explicit conversion method option.

Syntax

`fetch(options) object_list`

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 optionally fetch from individual databank files or search among registered databases.

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.

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 including identifying extensions.

Options

<code>d = db_name</code>	Fetch from specified database.
<code>d</code>	Fetch all registered databases in registry order.
<code>i</code>	Fetch from individual databank files.
<code>link</code>	Fetch as a database link.
<code>notifillegal</code>	When in a program, report illegal EViews object names. By default, objects with illegal names are automatically renamed. (Has no effect in the command window.)

The following options are available for fetch of group objects:

<code>g = arg</code>	Group fetch options: “b” (fetch both group definition and series), “d” (fetch only the series in the group), “l” (fetch only the group definition).
----------------------	---

The database specified by the double colon “::” takes precedence over the database specified by the “d = ” option.

In addition, there are a number of 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:

<code>c = arg</code>	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:

<code>c = arg</code>	<p><i>High to low conversion methods removing NAs:</i> “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).</p> <p><i>High to low conversion methods propagating NAs:</i> “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).</p>
----------------------	--

If no conversion method is specified, the series-specific or global default conversion method will be employed.

Examples

To fetch M1, GDP, and UNEMP from the default database, use:

```
fetch m1 gdp unemp
```

To fetch M1 and GDP from the US1 database and UNEMP from the MACRO database, use the command:

```
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 destination workfile frequency:

```
fetch(d,c=f) sp*
```

You can fetch M1 and UNEMP from individual databank files using:

```
fetch(i) m1 c:\data\unemp
```

To fetch all objects with names starting with “CONS” from the two databases USDAT and UKDAT, use the command:

```
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”.

Use the “notifyillegal” option to display a dialog when fetching the series MYIL-LEG@LNAME that will suggest a valid name and give you the opportunity to name the object before it is inserted into a workfile:

```
pool2.fetch(notifyillegal) myilleg@lname
```

Cross-references

See [Chapter 10. “EViews Databases,” on page 257](#) of the *User’s Guide I* for a discussion of databases, databank files, and frequency conversion. [Appendix C. “Wildcards,” on page 775](#) of the *User’s Guide I* describes the use of wildcard characters.

See also [Series::setconvert \(p. 377\)](#), [store \(p. 787\)](#), and [copy \(p. 696\)](#).

fit	Commands
-----	----------

Computes static forecasts or 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 [Equation::makemodel \(p. 68\)](#) to create a model using the ordered equation results.

(Note that we recommend that you instead use the equation proc ([Equation::fit \(p. 49\)](#)) since it explicitly specifies the equation of interest.)

Syntax

```
fit(options) yhat [y_se]
```

Following the `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

d	In models with implicit dependent variables, forecast the entire expression rather than the normalized variable.
u	Substitute expressions for all auto-updating series in the equation.
g	Graph the fitted values together with the ± 2 standard error bands.
e	Produce the forecast evaluation table.
i	Compute the fitted values of the index. Only for binary, censored and count models.
s	Ignore ARMA terms and use only the structural part of the equation to compute the fitted values.
f = <i>arg</i> (<i>default</i> = "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).
p	Print view.

Examples

```
equation eq1.ls cons c cons(-1) inc inc(-1)
fit c_hat c_se
genr c_up=c_hat+2*c_se
genr c_low=c_hat-2*c_se
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 C_HAT and C_SE. The third and fourth lines compute the ± 2 standard error bounds. The fifth line plots the actual series together with the error bounds.

```
equation eq2.binary(d=1) y c wage edu
fit yf
fit(i) xbeta
genr yhat = 1-@clogit(-xbeta)
```

The first line estimates a logit specification for Y with a conditional mean that depends 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.makemodel (oprobl)
solve oprobl
```

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. 722). We recommend that you use [Equation::forecast](#) (p. 52) and [Equation::fit](#) (p. 49), rather than the interactive command forms. See [Equation::makemodel](#) (p. 68) and [solve](#) (p. 782) for forecasting from systems of equations or ordered equations.

See [Chapter 27. “Forecasting from an Equation,” on page 113](#) of the *User’s Guide II* for a discussion of forecasting in EViews and [Chapter 30. “Discrete and Limited Dependent Variable Models,” on page 209](#) of the *User’s Guide II* for forecasting from binary, censored, truncated, and count models. See [“Forecasting” on page 386](#) of the *User’s Guide II* for a discussion of forecasting from sspace models.

forecast	Commands
----------	--------------------------

Computes (*n*-period ahead) dynamic forecasts of the default equation.

`forecast` computes the forecast using the default equation 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.

(Note that we recommend that you instead use the equation proc ([Equation::forecast](#) (p. 52)) since it explicitly specifies the equation of interest.)

Syntax

```
forecast(options) yhat [y_se]
```

You should enter a name for the forecast series and, optionally, a name for the series containing the standard errors. Forecast standard errors are currently not available for binary or censored models. `forecast` is not available for models estimated using ordered methods.

Options

d	In models with implicit dependent variables, forecast the entire expression rather than the normalized variable.
u	Substitute expressions for all auto-updating series in the equation.
g	Graph the forecasts together with the ± 2 standard error bands.
e	Produce the forecast evaluation table.
i	Compute the forecasts of the index. Only for binary, censored and count models.
s	Ignore ARMA terms and use only the structural part of the equation to compute the forecasts.
f = <i>arg</i> (<i>default</i> = "actual")	Out-of-forecast-sample fill behavior: "actual" (fill observations outside the forecast sample with actual values for the fitted variable), "na" (fill observations outside the forecast sample with missing values).
p	Print view.

Examples

The following lines:

```
smpl 1970q1 1990q4
equation eql.ls con c con(-1) inc
smpl 1991q1 1995q4
forecast con_d
plot con_d
```

estimate a linear regression over the period 1970Q1–1990Q4, computes adynamic forecasts for the period 1991Q1–1995Q4, and plots the forecast as a line graph.

```
equation eql.ls m1 gdp ar(1) ma(1)
forecast m1_bj bj_se
forecast(s) m1_s s_se
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.

Cross-references

To perform static forecasting using the interactive command `fit` (p. 720). We recommend that you use `Equation::forecast` (p. 52) and `Equation::fit` (p. 49), rather than the interactive command forms. For multiple equation forecasting, see `Equation::makemodel` (p. 68), and `solve` (p. 782).

For more information on equation forecasting in EViews, see [Chapter 27. “Forecasting from an Equation,” on page 113](#) of the *User’s Guide II*. For additional discussion of wildcards, see [Appendix C. “Wildcards,” on page 775](#) of the *User’s Guide I*.

freeze	Commands
--------	----------

Creates graph, table, or text objects from a view.

Syntax

`freeze(options, 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 may provide a destination name for the object containing the frozen view in parentheses.

Options

<code>mode = overwrite</code>	Overwrites the object <i>name</i> if it already exists.
-------------------------------	---

Examples

```
freeze gdp.uroot(4,t)
```

creates an untitled table that contains the results of the unit root test of the series GDP.

```
group rates tb1 tb3 tb6
freeze(gra1) rates.line(m)
show gra1.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.

```
freeze(mygra) gra1 gra3 gra4
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.

```
freeze(mode=overwrite, mygra) gra1 gra2 gra3
show mygra.align(2,1,1)
```


creates a graph object MYGRA that combines the three graph objects GRA1, GRA2 and GRA3, and displays MYGRA in two columns. If the object MYGRA already exists, it would be replaced by the new object.

Cross-references

See [“Object Commands” on page 582](#) of the *User’s Guide I*. See also [Chapter 4. “Object Basics,” on page 63](#) of the *User’s Guide I* for further discussion of objects and views of objects. Freezing graph views is described in [“Creating Graph Objects” on page 524](#) of the *User’s Guide I*.

frml	Commands
------	--------------------------

Declare a series object with a formula for auto-updating, or specify a formula for an existing series.

Syntax

```
frml series_name = series_expression
```

```
frml series_name = @clear
```

Follow the `frml` keyword with a name for the object, and an assignment statement. 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 an assignment statement. The following example creates a series named LOW that uses a formula to compute its values.:

```
frml low = inc<=5000 or edu<13
```

The auto-updating series takes the value 1 if either INC is less than or equal to 5000 or EDU is less than 13, and 0 otherwise, 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 = 3
```

```
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 = 3
z = (x+y)/2
```

while similar, produce quite different results, since the absence of the `frml` keyword in the second example means that EViews will generate fixed values in the series instead of defining a formula to compute the series values. In this latter case, the values in the series Z are fixed, and 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 obtains 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 145](#) of the *User’s Guide I*.

See also [Link::link](#) (p. 225).

genr	Commands
------	----------

Generate series.

Generate series or alphas.

Syntax

```
genr ser_name = expression
```

Examples

```
genr y = 3 + x
```

generates a numeric series that takes the values from the series X and adds 3.

```
genr full_name = first_name + last_name
```

creates an alpha series formed by concatenating the alpha series FIRST_NAME and LAST_NAME.

Cross-references

See [Series::series \(p. 375\)](#) and [Alpha::alpha \(p. 6\)](#) for a discussion of the expressions allowed in `genr`.

gmm	Commands
------------	--------------------------

Estimation by generalized method of moments (GMM).

The equation object must be specified with a list of instruments.

Syntax

```
gmm(options) y x1 [x2 x3 ...] @ z1 [z2 z3 ...]
```

```
gmm(options) specification @ z1 [z2 z3 ...]
```

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.

In panel settings, you may specify dynamic instruments corresponding to predetermined variables. 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)”.

Note that dynamic instrument specifications may easily generate excessively large numbers of instruments.

Options

General Options

<code>m = integer</code>	Maximum number of iterations.
<code>c = number</code>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.
<code>l = number</code>	Set maximum number of iterations on the first-stage iteration to get the one-step weighting matrix.
<code>showopts / -showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
<code>deriv = keyword</code>	Set derivative methods. The argument <i>keyword</i> 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.
<code>p</code>	Print results.

Additional Options for Non-Panel Equation estimation

<code>w</code>	Use White’s diagonal weighting matrix (for cross section data).
<code>b = arg (default = “nw”)</code>	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).
<code>q</code>	Use the quadratic kernel. Default is to use the Bartlett kernel.
<code>n</code>	Prewhiten by a first order VAR before estimation.
<code>i</code>	Iterate simultaneously over the weighting matrix and the coefficient vector.
<code>s</code>	Iterate sequentially over the weighting matrix and coefficient vector.
<code>o (default)</code>	Iterate only on the coefficient vector with one step of the weighting matrix.
<code>c</code>	One step (iteration) of the coefficient vector following one step of the weighting matrix.
<code>e</code>	TSLS estimates with GMM standard errors.

Additional Options for Panel Equation estimation

<code>cx = arg</code>	Cross-section effects method: (default) none, fixed effects estimation (“cx = f”), first-difference estimation (“cx = fd”), orthogonal deviation estimation (“cx = od”)
<code>per = arg</code>	Period effects method: (default) none, fixed effects estimation (“per = f”).
<code>levelper</code>	Period dummies always specified in levels (even if one of the transformation methods is used, “cx = fd” or “cx = od”).
<code>wgt = arg</code>	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”).
<code>gmm = arg</code>	GMM weighting: 2SLS (“gmm = 2sls”), White period system covariances (Arellano-Bond 2-step/ <i>n</i> -step) (“gmm = perwhite”), White cross-section system (“gmm = cxwhite”), White diagonal (“gmm = stackedwhite”), Period system (“gmm = persur”), Cross-section system (“gmm = cxsur”), Period heteroskedastic (“cov = perdiag”), Cross-section heteroskedastic (“gmm = cxdiag”). By default, uses the identity matrix unless estimated with first difference transformation (“cx = fd”), in which case, uses (Arellano-Bond 1-step) difference weighting matrix. In this latter case, you should specify 2SLS weights (“gmm = 2sls”) for Anderson-Hsiao estimation.
<code>cov = arg</code>	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”).
<code>keepwgt</code>	Keep full set of GLS/GMM weights used in estimation with object, if applicable (by default, only weights which take up little memory are saved).

<code>nodf</code>	Do not perform degree of freedom corrections in computing coefficient covariance matrix. The default is to use degree of freedom corrections.
<code>coef = arg</code>	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the “C” coefficient vector.
<code>iter = arg</code> (<i>default</i> = “onec”)	Iteration control for GLS and GMM weighting 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 some options are only available for a subset of specifications.

Examples

In a non-panel workfile, we may estimate equations using the standard GMM options. The specification:

```
gmm(w) y c f k @ c z1 z2 z3
```

estimates the linear specification using a White diagonal weighting matrix (one-step, with no iteration). The command:

```
gmm(e, b=v) c(1) + c(2)*(m^c(1) + k^(1-c(1))) @ c z1 z2 z3
```

estimates the nonlinear model using two-stage least squares (instrumental variables) with GMM standard errors computed using Newey-West automatic bandwidth selected weights.

When working with a workfile that has a panel structure, you may use the panel equation estimation options. The command

```
gmm(cx=fd, per=f) dj dj(-1) @ @dyn(dj)
```

estimates an Arellano-Bond “1-step” estimator with differencing of the dependent variable DJ, period fixed effects, and dynamic instruments constructed using DJ with observation specific lags from period $t-2$ to 1.

To perform the “2-step” version of this estimator, you may use:

```
gmm(cx=fd, per=f, gmm=perwhite, iter=oneb) dj dj(-1) @ @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, to use the estimates to form period covariance GMM weights, and then re-estimate the model.

You may iterate the GMM weights to convergence using:

```
gmm(cx=fd, per=f, gmm=perwhite, iter=seq) dj dj(-1) @ @dyn(dj)
```

Alternately:

```
gmm(cx=od, gmm=perwhite, iter=oneb) dj dj(-1) x y @ @dyn(dj,-2,-6)
x(-1) y(-1)
```

estimates an Arellano-Bond “2-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

See [Chapter 25. “Additional Regression Methods,” on page 23](#) and [Chapter 39. “Panel Estimation,” on page 541](#) of the *User’s Guide II* for discussion of the various GMM estimation techniques.

We recommend that you use [Equation::gmm \(p. 54\)](#) rather than this interactive command form.

hconvert	Commands
----------	--------------------------

Convert an entire Haver Analytics Database into an EViews database.

Syntax

```
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 Haver database. 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 [Chapter 10. “EViews Databases,” on page 257](#) of the *User’s Guide I* for a discussion of EViews and Haver databases.

See also [dbcreate](#) (p. 708), [db](#) (p. 707), [hfetch](#) (p. 732) and [hlabel](#) (p. 733).

hfetch	Commands
---------------	--------------------------

Fetch a series from a Haver Analytics database into a workfile.

`hfetch` 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

`hfetch(database_name) series_name`

`hfetch`, 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 `hfetch` 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 status line. The database name is optional if a default database has been specified.

`hfetch` 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

See also [Chapter 10. “EViews Databases,” on page 257](#) of the *User’s Guide I* for a discussion of EViews and Haver databases. Additional information on EViews frequency conversion is provided in [“Frequency Conversion” on page 106](#) of the *User’s Guide I*.

See also [dbcreate](#) (p. 708), [db](#) (p. 707), [hconvert](#) (p. 731) and [hlabel](#) (p. 733).

hist	Commands
-------------	--------------------------

Histogram and descriptive statistics of a series.

The `hist` command computes descriptive statistics and displays a histogram for the series.

Syntax

`hist(options) series_name`

Options

p	Print the histogram.
---	----------------------

Examples

```
hist lwage
```

Displays the histogram and descriptive statistics of LWAGE.

Cross-references

See [“Histogram and Stats” on page 306](#) of the *User’s Guide I* for a discussion of the descriptive statistics reported in the histogram view.

hlabel	Commands
--------	--------------------------

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

```
hlabel(database_name) series_name
```

`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 `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 *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 HAVER-DIR.INI.

If several series names are placed on the line, `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

```
hlabel(c:\data\us1) pop
```

displays the description of the series POP in the US1 database.

Cross-references

See [Chapter 10. “EViews Databases,” on page 257](#) of the *User’s Guide I* for a discussion of EViews and Haver databases.

See also [hfetch \(p. 732\)](#) and [hconvert \(p. 731\)](#).

hpf	Commands
-----	--------------------------

Smooth a series using the Hodrick-Prescott filter.

Syntax

`hpf(options) series_name filtered_name`

Smoothing Options

The degree of smoothing may be specified as an option. You may specify the smoothing as a value, or using a power rule:

<code>lambda = arg</code>	Set smoothing parameter value to <i>arg</i> ; a larger number results in greater smoothing.
<code>power = arg</code> <i>(default = 2)</i>	Set smoothing parameter value using the frequency power rule of Ravn and Uhlig (2002) (the number of periods per year divided by 4, raised to the power <i>arg</i> , and multiplied by 1600). Hodrick and Prescott recommend the value 2; Ravn and Uhlig recommend the value 4.

If no smoothing option is specified, EViews will use the power rule with a value of 2.

Other Options

<code>p</code>	Print the graph of the smoothed series and the original series.
----------------	---

Examples

```
hpf(lambda=1000) gdp gdp_hp
```

smooths the GDP series with a smoothing parameter “1000” and saves the smoothed series as GDP_HP.

Cross-references

See [“Hodrick-Prescott Filter” on page 360](#) of the *User’s Guide I* for details.

load	Commands
-------------	----------

Load a workfile.

Provided for backward compatibility. Same as [wfoopen](#) (p. 802).

logit	Commands
--------------	----------

Estimate binary models with logistic errors.

Provide for backward compatibility. Equivalent to issuing the command, `binary` with the option “(d=1)”.

See [binary](#) (p. 686).

ls	Commands
-----------	----------

Estimation by linear or nonlinear least squares regression.

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

`ls(options) y x1 [x2 x3 ...]`

`ls(options) specification`

Options

General options

<code>m = integer</code>	Set maximum number of iterations.
<code>c = scalar</code>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.
<code>s</code>	Use the current coefficient values in “C” as starting values for equations with AR or MA terms (see also param (p. 761)).

<code>s = number</code>	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 to be used. Note that out of range values are set to “s = 1”. Specifying “s = 0” initializes coefficients to zero. By default EViews uses “s = 1”.
<code>showopts / -showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
<code>deriv = keyword</code>	Set derivative methods. The argument <i>keyword</i> 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.
<code>p</code>	Print basic estimation results.

Additional Options for Panel Equation estimation

<code>cx = arg</code>	Cross-section effects: (default) none, fixed effects (“cx = f”), random effects (“cx = r”).
<code>per = arg</code>	Period effects: (default) none, fixed effects (“per = f”), random effects (“per = r”).
<code>wgt = arg</code>	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”).
<code>cov = arg</code>	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/PCSE (“cov = perdiag”).
<code>keepwghts</code>	Keep full set of GLS weights used in estimation with object, if applicable (by default, only small memory weights are saved).
<code>rancalc = arg (default = “sa”)</code>	Random component method: Swamy-Arora (“rancalc = sa”), Wansbeek-Kapteyn (“rancalc = wk”), Wallace-Hussain (“rancalc = wh”).

<code>nodf</code>	Do not perform degree of freedom corrections in computing coefficient covariance matrix. The default is to use degree of freedom corrections.
<code>b</code>	Estimate using a balanced sample (pool estimation only).
<code>coef = arg</code>	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the “C” coefficient vector.
<code>iter = arg</code> (<i>default</i> = “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.

Examples

```
group rhs c dum1 dum2 dum3 dum4
ls cons rhs ar(1)
```

uses the group definition for RHS to regress CONS on C, DUM1, DUM2, DUM3, and DUM4, with an AR(1) correction.

Cross-references

[Chapter 24. “Basic Regression,” on page 5](#) and [Chapter 25. “Additional Regression Methods,” on page 23](#) of the *User’s Guide II* discuss the various regression methods in greater depth.

See [Chapter 39. “Panel Estimation,” on page 541](#) of the *User’s Guide II* for a discussion of panel equation estimation.

See [Chapter 5. “Special Expression Reference,” on page 815](#), for special terms that may be used in `ls` specifications.

open	Commands
------	--------------------------

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 for this purposes is provided for backward compatibility. We recommend instead that

you use the new commands [wfoopen](#) (p. 802) and [pageload](#) (p. 750) to open a workfile, and [dbopen](#) (p. 709) to open databases.

Syntax

`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 a path is not 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 files will be read as text files.

Options

p	Open file as program file.
t	Open file as text file.
type = arg (“prg” or “txt”)	Specify text or program file type using keywords.

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 [wfoopen](#) (p. 802) and [pageload](#) (p. 750) for opening files as workfiles or workfile pages, and [dbopen](#) (p. 709) for opening database files.

ordered	Commands
---------	----------

Estimation of ordered dependent variable models.

Syntax

equation name.**ordered**(*options*) *y* *x1* [*x2* *x3* ...]

equation name.**ordered**(*options*) *specification*

Options

<i>d</i> = <i>arg</i> (<i>default</i> = “n”)	Specify likelihood: normal likelihood function, ordered probit (“n”), logistic likelihood function, ordered logit (“l”), Type I extreme value likelihood function, ordered Gompit (“x”).
<i>q</i> (<i>default</i>)	Use quadratic hill climbing as the maximization algorithm.
<i>r</i>	Use Newton-Raphson as the maximization algorithm.
<i>b</i>	Use Berndt-Hall-Hall-Hausman as maximization algorithm.
<i>h</i>	Quasi-maximum likelihood (QML) standard errors.
<i>g</i>	GLM standard errors.
<i>m</i> = <i>integer</i>	Set maximum number of iterations.
<i>c</i> = <i>scalar</i>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.
<i>s</i>	Use the current coefficient values in C as starting values.
<i>s</i> = <i>number</i>	Specify a number between zero and one to determine starting values as a fraction of preliminary 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.
<i>deriv</i> = <i>keyword</i>	Set derivative methods. The argument <i>keyword</i> 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.
<i>p</i>	Print results.

If you choose to employ user specified starting values, the parameters corresponding to the limit points must be in ascending order.

Examples

```
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.

```
param c(1) .1 c(2) .2 c(3) .3 c(4) .4 c(5) .5
equation eq1.binary(s) y c x z
coef betahat = @coefs
```

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.

Cross-references

See [“Ordered Dependent Variable Models” on page 226](#) of the *User’s Guide II* for additional discussion.

See [binary \(p. 686\)](#) for the estimation of binary dependent variable models.

output	Commands
--------	--------------------------

Redirect 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, in a spool object, or into frozen table or graph objects in the current workfile.

Syntax

```
output[(f)] base_name
output(options) [path/]file_name
output[(s)] spool_name
output off
```

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 OUT01, the second print command will generate OUT02, 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.

You can use the “s” option, described below, to redirect all output to a spool object.

When followed by the optional keyword `off`, the `output` command turns off output redirection. Subsequent print commands will be directed to the printer.

Options

f	Redirect all output to frozen objects in the default workfile, using <i>base_name</i> .
t	Redirect table and text output to a text file. Graphic output will still be sent to the printer.
r	Redirect all output to an Rich Text Format (RTF) file.
s	Redirect all output to a spool object.
o	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”).
c	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”).

Examples

```
output print_
```

causes the first print command to generate a table or graph object named PRINT_01, the second print command to generate an object named PRINT_02, and so on.

```
output(t) c:\data\results
equation eq1.ls(p) log(gdp) c log(k) log(l)
eq1.resids(g,p)
output off
```

The first line redirects printing to the RESULTS.TXT file, while the print option of the second and third lines 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:

```
output(r) c:\data\results
```

all subsequent output would be sent to the RTF file RESULTS.RTF.

```
output(s) mySpool
```

redirects all output to the MYSPool spool. If the spool already exists, printed objects will be appended to the end of the spool.

Cross-references

See “Print Setup,” beginning on page 771 of the *User’s Guide II* for further discussion.

See also [pon](#) (p. 875), [poff](#) (p. 875).

pageappend	Commands
------------	----------

Append contents of the specified workfile page to the active workfile page.

Syntax

`pageappend`(*options*) *wfname*[*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 objects other than series or alpha objects to 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 [pagecopy](#) (p. 744).

Options

<code>smpl = <i>smpl_spec</i></code>	Specifies an optional sample identifying which observations from the source page are to be appended. Either provide the sample range in double quotes or specify a named sample object. The default is “@all”.
<code>allobj</code>	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 <i>object_list</i> specification.
<code>match</code>	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 <i>object_list</i> specification.
<code>sufix = <i>suffix_arg</i></code> (default = “_a”)	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.
<code>obsid = <i>arg</i></code>	Provides the name of a series used to hold the date or observation ID of each observation in the destination workfile.
<code>wfid = <i>arg</i></code>	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).

Examples

```
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 [“Appending to a Workfile” on page 230](#) of the *User’s Guide I* for discussion.

See also [pagecopy \(p. 744\)](#).

pagecontract	Commands
--------------	----------

Contract the active workfile page according to the specified sample.

Syntax

```
pagecontract simpl_spec
```

where *simpl_spec* is a sample specification. Contraction removes observations not in specified sample from the active workfile page. Note that since this operation is performed in place, you may wish to backup your original page (see [pagecopy \(p. 744\)](#)) prior to contracting.

Examples

```
pagecontract if income<50000 and race=2
```

removes all observations with INCOME values less than or equal to 50000 and RACE not equal to 2.

```
pagecontract 1920 1940 1945 2000
```

removes observations for the years 1941 to 1944.

Cross-references

See [“Contracting a Workfile” on page 233](#) of the *User’s Guide I* for discussion.

See also [pagecopy \(p. 744\)](#).

pagecopy	Commands
----------	----------

Copies all or part of the active workfile page to a new workfile, or to a new page within the default workfile.

Syntax

```
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 be 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, data objects must be copied by value.

Options

<code>bylink</code>	Specifies that series and alpha objects be copied as links to the source page. This option is not available if you use the “ <code>wf =</code> ” option, since linking requires that the destination page be in the same workfile as the source page. Automatically sets the “ <code>dataonly</code> ” option so that only series, alphas, links, and valmaps will be copied.
<code>smpl = <i>smpl_spec</i></code>	Specifies an optional sample identifying which observations from the source page are to be appended. Either provide the sample range in double quotes or specify a named sample object. The default is “ <code>@all</code> ”.
<code>rndobs = <i>integer</i></code>	Copy only a random subsample of <i>integer</i> observations from the specified sample. Not available with “ <code>bylink</code> ” or “ <code>rndpct</code> ”.
<code>rndpct = <i>arg</i></code>	Copy only a random subsample of <i>arg</i> (a number between 0 and 1) of the specified sample. Not available with “ <code>bylink</code> ” or “ <code>rndobs</code> ”.
<code>dataonly</code>	Only series, alphas, links, and valmaps should be copied. The default is to copy all objects (unless the “ <code>bylink</code> ” option is specified, in which case only series objects are copied).
<code>nolinks</code>	Do not copy links from the source page.
<code>wf = <i>wf_name</i></code>	Optional name for the destination workfile. If not provided, EViews will create a new untitled workfile. Not available if copying using the “ <code>bylink</code> ” option.
<code>page = <i>page_name</i></code>	Optional name for the newly created page. If not provided, EViews will use the next available name of the form “ <code>Untitled##</code> ”, where <code>##</code> is a number.

Examples

```
pagecopy(page=allvalue, wf=newwf)
```

will first create a new workfile named NEWWF, 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. Series and alpha objects only will be copied by value from the source page.

Cross-references

See [“Copying from a Workfile” on page 233](#) of the *User’s Guide I* for discussion.

See also [pageappend](#) (p. 742).

pagecreate	Commands
------------	--------------------------

Create a new page in the default workfile. The new page becomes the active page.

Syntax

```
pagecreate(options) freq start_date end_date [num_cross_sections]
pagecreate(options) u num_observations
pagecreate(id_method[,options]) id_list [@srcpage page_list]
pagecreate(idcross[,options]) id1 id2 [@srcpage page1 page2]
pagecreate(idcross[,options]) id1 @range(freq, start_date, end_date) [@srcpage page1]
```

These different forms of the `pagecreate` command encompass three distinct approaches to creating a new workfile page: (1) regular frequency description or unstructured data description; (2) using the union or intersection of unique values from one or more ID series in existing workfile pages; (3) using the cross of unique values from two identifier series or from an identifier series and a date range. Each of these approaches is described in greater detail below.

Regular Frequency or Unstructured Description

The first two forms of the command permit you to create a new workfile page using a regular frequency or unstructured description:

- **pagecreate**(*options*) *freq start_date end_date [num_cross_sections]*
- **pagecreate**(*options*) **u** *num_observations*

The first form of the command should be employed to create a regular frequency page with the specified frequency, start, and end date. The *freq* argument may be specified as “a” (annual), “s” (semi-annual), “q” (quarterly), “m” (monthly), “w” (weekly), “d” (5-day daily), “7” (7-day daily). If you include the optional argument *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 defined using [pagestruct](#) (p. 756).

The second form of the command creates an unstructured workfile with the specified number of observations.

Note that these forms of the command are analogous to [wfcreate](#) (p. 801) except that instead of creating a new workfile, we create a new page in the default workfile.

Unique Values from a Set of Identifier Series

The next form of the command allows for creating pages from the unique values of one or more identifier series found in one or more workfile pages:

- **pagecreate**(*id_method*[*options*]) *identifier_list* [**@srcpage** *page_list*]

The *identifier_list* should include one or more ID series. If more than one ID series is provided, EViews will use the values that are unique across all of the series. If you wish to create a page with a date structure, you should specify *one* of your identifiers using the special “@DATE” keyword identifier, enclosing the series (or the date ID component series) inside parentheses. If you wish to use the date ID values from the source workfile page, you may use the “@DATE” keyword without arguments.

The *id_method* describes how to handle unique ID values that differ across multiple pages:

byid	Use the observed values of the series in the <i>identifier_list</i> in specified page.
idunion / byid	Use the union of the observed values of the series in the <i>identifier_list</i> in the specified pages.
idintersect	Use the intersection of the observed values of the series in the <i>identifier_list</i> in the specified pages.

If the optional source page or list of source pages is not provided, EViews will use the default workfile page. Note that if a single workfile page is used, the two ID methods yield identical results.

Cross of Unique Values from Two Identifier Series or from an Identifier Series and a Date Range

The last two forms of the command create a new page by crossing the unique values in two ID series located in one or more pages, or by crossing an ID series from one page with a date

range. First, you may specify a pair of identifiers, and optionally source pages where they are located,

- `pagecreate(idcross[,options]) id1 id2 [@srcpage page1 page2]`

You may instruct EViews to create a date structured page by specifying one of your two identifiers using a “@DATE” specification as described above.

Alternately, you may provide a single identifier and a range specification using the “@RANGE” keyword with a *freq*, *start_date*, and *end_date*, and optionally, the location of the identifier series.

- `pagecreate(idcross[,options]) id1 @range(freq, start_date, end_date) [@srcpage page1]`

Options

<code>smpl = <i>smpl_spec</i></code>	Specifies an optional sample identifying which observations to use when creating a page using the <i>id_method</i> option. Either provide the sample range in double quotes or specify a named sample object. The default is “@all”. When multiple source workfiles are involved, the specified sample must be valid for all workfiles.
<code>page = <i>page_name</i></code>	Optional name for the newly created page. If not provided, EViews will use the next available name of the form “Untitled##”, where ## is a number.

Examples

Regular Frequency or Unstructured Description

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 ANNUAL, containing data from 1950 to 2005; the second is a 1000 observation unstructured page named UNSTRUCT.

```
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.

Unique Values from a Set of Identifier Series

```
pagecreate(id, page=statepage) state
```

creates a new page STATEIND using the distinct values of STATE in the current workfile page.

```
pagecreate(id, page=statepage) state industry
```


creates a new page named STATEIND, using the distinct STATE/INDUSTRY values in the active page.

```
pagecreate(id, page=stateyear) state @date(year)
pagecreate(id, page=statemonth) @date(year, month)
```

use STATE, along with YEAR, and the YEAR and MONTH series respectively, to form identifiers that will be used in creating the new dated workfile pages.

```
pagecreate(id, smpl="if sex=1") crossid @date
```

creates a new page using CROSSID and existing date ID values of the active workfile page. Note that only observations in the subsample defined by “@all if sex = 1” are used to determine the unique values.

```
pagecreate(id, page=AllStates, smpl="if sex=""Female"") stateid
@srcpage north south east west
```

creates a new page ALLSTATES structured using the union of the unique values of STATEID from the NORTH, SOUTH, EAST and WEST workfile pages that are in the sample “if sex = “Female””. Note the use of the double quote escape character for the double quotes in the sample string.

```
pagecreate(idintersect, page=CommonStates, smpl="1950 2005")
stateid @srcpage page1 page2 page3
```

creates a new page name COMMONSTATES structured using the intersection of the unique values of STATEID taken from the pages PAGE1, PAGE2, and PAGE3.

Cross of Unique Values from Two Identifier Series or from an Identifier Series and a Date Range

```
pagecreate(idcross,page=UndatedPanel) id1 id2 @srcpage page1 page2
```

will add the new page UNDATEDPANEL to the current workfile. UNDATEDPANEL will be structured as an undated panel using values from the cross of ID1 from PAGE1 and ID2 from PAGE2.

To create a dated page using the “idcross” option, you must tag one of the identifiers using an “@DATE” specification:

```
pagecreate(idcross,page=AnnualPanel) id1 @date(year) @srcpage
page1 page2
```

You may also specify the cross of an identifier with a date range:

```
pagecreate(idcross,page=QuarterlyPanel) id1
@range(q, 1950, 2006) @srcpage page1
```

creates a quarterly panel page named QUARTERLYPANEL using the cross of ID1 taken from PAGE1, and quarterly observations from 1950q1 to 2006q4.

Cross-references

See [“Creating a Workfile” on page 38](#) of the *User’s Guide I* for discussion.

See also [wfcreate](#) (p. 801) and [pagedelete](#) (p. 750).

pagedelete	Commands
-------------------	--------------------------

Delete the named page from the default workfile.

Syntax

`pagedelete pgname`

where *pgname* is the name of a page in the default workfile.

Examples

```
pagedelete page1
```

Cross-references

See also [pagecreate](#) (p. 746).

pageload	Commands
-----------------	--------------------------

Load one or more new pages in the default workfile.

Syntax

`pageload [path\]workfile_name`
`pageload(options) source_description [@keep keep_list] [@drop drop_list] [@keepmap keepmap_list] [@dropmap dropmap_list] [@selectif condition]`
`pageload(options)source_description table_description [@keep keep_list] [@drop drop_list] [@keepmap keepmap_list] [@dropmap dropmap_list] [@selectif condition]`

The basic syntax for `pageload` follows that of [wfopen](#) (p. 802). The difference between the two commands is that `pageload` creates a new page in the default workfile, rather than opening or creating a new workfile. 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 “INDIVID1”).

Options

<code>type = arg / t = arg</code>	<p>Optional type specification: Access database file (“access”), Aremos-TSD file (“a”, “aremos”, “tsd”), Binary file (“binary”), Excel file (“excel”), Gauss dataset file (“gauss”), GiveWin/PcGive 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 (“l”, “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”).</p>
-----------------------------------	---

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.

See the extensive set of examples in [wfopen](#) (p. 802).

Cross-references

See “[Creating a Page by Loading a Workfile or Data Source](#)” on page 59 of the *User’s Guide I* for discussion.

See also [wfopen](#) (p. 802) and [pagecreate](#) (p. 746).

pagerename	Commands
-------------------	--------------------------

Load the specified workfile as one or more new pages in the default workfile.

Syntax

```
pagerename old_name new_name
```

renames the *old_name* page in the default workfile to *new_name*. Page names are case-insensitive for purposes of comparison, even though they are displayed using the input case.

Examples

```
pagerename Page1 StateByYear
```

Cross-references

See also [pagecreate](#) (p. 746).

pagesave	Commands
----------	----------

Save the active page in the default workfile as an EViews workfile (.wf1 file) or as a foreign data source.

Syntax

```
pagesave(options) [path/]filename
pagesave(options) source_description [@keep keep_list] [@drop drop_list] [@keepmap
keepmap_list] [@dropmap dropmap_list] [@smp1 smp1_spec]
pagesave(options) source_description table_description [@keep keep_list] [@drop
drop_list] [@keepmap keepmap_list] [@dropmap dropmap_list] [@smp1
smp1_spec]
```

The command 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. See [wfoopen](#) (p. 802) for details on the syntax of *source_descriptions* and *table_descriptions*.

Options

type = <i>arg</i> , t = <i>arg</i>	Optional type specification. Access database file (“access”), Aremos-TSD file (“a”, “aremos”, “tsd”), Binary file (“binary”), EViews database file (“e”, “evdb”), Excel file (“excel”), Gauss dataset file (“gauss”), GiveWin/PcGive 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 (“l”, “trl”), 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 = <i>arg</i>	Write selected maps as: numeric (“n”), character (“c”), both numeric and character (“b”).
nomapval	Do not write mapped values for series with attached value labels (the default is to write mapped values)

Examples

```
pagesave new_wf
```

saves the EViews workfile NEW_WF.WF1 in the default directory.

```
pagesave "c:\documents and settings\my data\consump"
```

saves the workfile CONSUMP.WF1 in the specified path.

Cross-references

See also [wfopen](#) (p. 802) and [wfsave](#) (p. 810).

pageselect	Commands
-------------------	--------------------------

Make the specified page in the default workfile the active page.

Syntax

```
pageselect pgname
```

where *pgname* is the name of a page in the default workfile.

Examples

```
pageselect page2
```

changes the active page to PAGE2.

Cross-references

See also [wfselect](#) (p. 811).

pagestack	Commands
------------------	--------------------------

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 new panel workfile may be created by stacking series, alphas, and 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 stack identifiers.

Syntax

```
pagestack(options) stack_id_list [@ series_to_stack]
```

```
pagestack(options) pool_name [@ series_to_stack]
```

```
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 as a stand in for 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 in the new workfile using the base portion of the series name (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 individual 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_stack* list is not specified, the expression “*? *”, is assumed.

Options

<code>? = name_patt,</code> <code>idreplace =</code> <code>name_patt</code>	Specifies the characters to use instead of the identifier, "?", in naming the stacked series. By default, the <i>name_patt</i> is blank, indicating, for example, that the stacked series corresponding to the pattern "GDP?" will be named "GDP" in the stacked workfile page. If pattern is set to "STK", the stacked series will be named GDP-STK.
<code>interleave</code>	Interleave the observations in the destination stacked workfile (stack by using 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 the series one on top of each other).
<code>wf = wf_name</code>	Optional name for the new workfile. If not provided, EViews will create a new page in the default workfile.
<code>page = page_name</code>	Optional name for the newly created page. If not provided, EViews will use the next available name of the form "Untitled##", where ## is a number.

Examples

Consider a workfile that contains the seven series: GDPUS, GDPUK, GDPJPN, CONSUS, CONSUK, CONSJPN, CONSF, and WORLDGDP.

```
pagestack us uk jpn @ *?
```

creates a new, panel structured workfile page with the series GDP and CONS, containing the stacked GDP? series (GDPUS, GDPUK, and GDPJPN) and stacked CONS? series (CONSUS, CONSUK, and CONSJPN). Note that CONSF and WORLDGDP will not be copied or 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", the 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 instruct EViews to use it to specify the IDs:

```
pagestack(wf=newwf, page=stackctry) countrypool @ gdp? cons?
```

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:

```
pagestack us uk jpn @ gdp? cons? worldgdp consfr
pagestack(wf=altwf) us uk jpn @ 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 link objects CONSFR and WORLDGDP, which repeat the values of the series. In the second case, the new panel page in the workfile ALTWF will contain the stacked GDP and CONS, and series named CONSFR and WORLDGDP containing repeated copies of the values of the series.

The following two commands are equivalent:

```
pagestack(wf=newwf) us uk jpn @ *? *
pagestack(wf=newwf) us uk jpn
```

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.

```
pagestack(?="stk") us uk jpn @ gdp? gdp
```

stacks GDPUS, GDPUK and GDPJPN into a series called GDPSTK 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 246 in the *User’s Guide I*. See also [pageunstack](#) (p. 759).

pagestruct	Commands
------------	--------------------------

Assign a structure to the active workfile page.

Syntax

```
pagestruct(options) [id_list]
pagestruct(options) *
```

where *id_list* is an (optional) list of ID series. The “*” may be used as shorthand for the indices currently in place.

If an *id_list* is provided, EViews will attempt to auto determine the workfile structure. Auto-determination may be overridden through the use of options.

If you do not provide an *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.

Options

none	Remove the existing workfile structure.
freq = <i>arg</i>	Specifies a regular frequency; if not provided EViews will auto-determine the frequency. The frequency may be specified as “a” (annual), “s” (semi-annual), “q” (quarterly), “m” (monthly), “w” (weekly), “d” (5-day daily), “7” (7-day daily), or “u” (unstructured/undated).
start = <i>arg</i>	Start date of the regular frequency structure; if not specified, defaults to “@FIRST”. Legal values for <i>arg</i> are described below.
end = <i>arg</i>	End date of the regular frequency structure; if not specified, defaults to “@LAST”. Legal values for <i>arg</i> are described below.
regular, reg	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 regular frequency calendar. The option has no effect unless a date index is specified.
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.

balance = <i>arg</i> , bal = <i>arg</i>	<p>Balance option (for panel data) describing how EViews should handle data that are unbalanced across ID (cross-section) groups.</p> <p>The <i>arg</i> should be formed using a combination of starts (“s”), ends (“e”), and middles (“m”), as in “balance = se” or “balance = m”.</p> <p>If balancing starts (<i>arg</i> 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).</p> <p>If balancing ends (<i>arg</i> contains “e”), EViews will add any observations required so that each cross-section ends at the same point (the last date or observation observed).</p> <p>If balancing middles (<i>arg</i> 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.</p> <p>Note that “balance = m” is implied by the “regular” option.</p>
dropna	<p>Specifies that observations which contain missing values (NAs or blank strings) in any ID series (including the date or observation ID) be removed from the workfile. If “dropna” is not specified and NAs are found, EViews will prompt in interactive mode and fail in programs.</p>
dropbad	<p>Specifies that observations for which any of the date index series contain values that do not represent dates be removed from the workfile. If “dropbad” is not provided and bad dates are present, EViews will prompt in interactive mode and fail in programs.</p>

The values for start and end dates should contain date literals (actual dates or periods), *e.g.*, “1950q1” and “2/10/1951”, “@FIRST”, or “@LAST” (first and last observed dates in the date ID series). 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 ID series (or a series used to form a date ID) should be tagged using the “@DATE” keyword. For example:

```
pagestruct state @date(year)
pagestruct(regular) @date(year, month)
```

A “*” may be used to indicate the indices defined in the current workfile structure.

```
pagestruct(end=@last+5) *
```

adds 5 observations to the end of the current workfile.

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 remove the existing structure:

```
pagestruct(freq=a, start=1950)
pagestruct(none)
```

Cross-references

For extensive discussion, see “[Structuring a Workfile](#),” beginning on page 203 in the *User’s Guide I*.

pageunstack	Commands
-------------	--------------------------

Unstack workfile page (convert repeated observations to repeated series).

Create a new workfile page by taking series objects (series, alphas, or links) in the default workfile page and breaking them into multiple series (or alphas), one for each distinct value found in a user supplied list of series objects. Typically used on a page with a panel structure.

Syntax

```
pageunstack(options) stack_id obs_id [@ series_to_unstack]
```

where *stack_id* is a single series containing the unstacking ID values used to identify the individual unstacked series, *obs_id* is a series containing the observation IDs, and *series_to_unstack* is an optional list of series objects to be copied to the new workfile.

Options

<code>namepat</code> <code>= name_pattern</code>	<p>Specifies the pattern from which unstacked series names are constructed, where “*” indicates the original series name and “?” indicates the stack ID.</p> <p>By default the <i>name_pattern</i> 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.</p>
<code>wf = wf_name</code>	<p>Optional name for the new workfile. If not provided, EViews will create a new page in the default workfile.</p>
<code>page = page_name</code>	<p>Optional name for the newly created page. If not provided, EViews will use the next available name of the form “Untitled##”, where ## is a number.</p>

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 from which country each observation of GDP and CONS comes. Finally, suppose there is a date series DATEID which identifies the date for each observation. The command:

```
pageunstack country dateid @ gdp cons
```

creates a new workfile page using the workfile frequency and dates found in DATEID. The page will contain the 6 series GDPUS, GDPUK, GDPJPN, CONSUS, CONSUk, and CONSJPN corresponding to the unstacked GDP and CONS.

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. Since the panel has built-in date information, we may use the “@DATE” keyword as the DATEID. The command:

```
pageunstack country @date @ gdp cons
```

uses the date portion of the current workfile structure to identify the dates for the unstacked page.

The *stack_id* must be an ordinary, or an alpha series that uniquely identifies the groupings to use in unstacking the series. *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:

```
pageunstack 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 two commands:

```
pageunstack country dateid @ *
pageunstack 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 may be changed using the “name-pat=” option. For example:

```
pageunstack(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 240 of the *User’s Guide I*. See also [pagestack](#) (p. 753).

param	Commands
-------	----------

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

```
param coef_name1 number1 [coef_name2 number2 coef_name3 number3...]
```

List, in pairs, the names of the coefficient vector and its element number followed by the corresponding starting values for any of the parameters in your equation.

Examples

```
param c(1) .2 c(2) .1 c(3) .5
```

resets the first three values of the coefficient vector C.

```
coef(3) beta
param beta(2) .1 beta(3) .5
```

The first line declares a coefficient vector BETA of length 3 that is initialized with zeros. The second line sets the second and third elements of BETA to 0.1 and 0.5, respectively.

Cross-references

See [“Starting Values” on page 50](#) of the *User’s Guide II* for a discussion of setting initial values in nonlinear estimation.

plot	Commands
------	--------------------------

Line graph.

Provided for backward compatibility. See [line](#) (p. 630).

print	Commands
-------	--------------------------

Sends views of objects to the default printer.

Syntax

```
print(options) object1 [object2 object3 ...]
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 an object, `print` will print the default view for the object.

Options

p	Print in portrait orientation.
l	Print in landscape orientation.

The default orientation is set by clicking on **Print Setup**.

Examples

```
print gdp log(gdp) d(gdp) @pch(gdp)
```

sends a table of GDP, log of GDP, first difference of GDP, and the percentage change of GDP to the printer.

```
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 orientation, you may use the commands:

```
graph mygra.merge graph1 graph2 graph3
```

```
mygra.align(3,1,1)
print(1) mygra
```

To estimate the equation EQ1 and send the output view to the printer.

```
print eq1.ls gdp c gdp(-1)
```

Cross-references

See [“Print Setup,” beginning on page 771](#) of the *User’s Guide II* for a discussion of print options and the **Print Setup** dialog.

See [output \(p. 740\)](#) for print redirection.

probit	Commands
--------	--------------------------

Estimation of binary dependent variable models with normal errors.

Equivalent to “`binary(d=n)`”.

See [binary \(p. 686\)](#).

program	Commands
---------	--------------------------

Declare a program.

Syntax

```
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 17. “EViews Programming,” on page 593](#) of the *User’s Guide I* for further details, and examples of writing EViews programs.

See also [open \(p. 737\)](#).

qreg	Commands
------	----------

Estimate a quantile regression specification.

Syntax

```
qreg(options) y x1 [x2 x3 ...]  
qreg(options) specification
```

EViews will create an unnamed equation object.

Options

quant = <i>number</i> (default = 0.5)	Quantile to be fit (where <i>number</i> is a value between 0 and 1).
cov = <i>arg</i> (default = “sandwich”)	Method for computing coefficient covariance matrix: “iid” (ordinary estimates), “sandwich” (Huber sandwich estimates), “boot” (bootstrap estimates). When “cov = iid” or “cov = sandwich”, EViews will use the sparsity nuisance parameter calculation specified in “spmethod = ” when estimating the coefficient covariance matrix.
bwmethod = <i>arg</i> (default = “hs”)	Method for automatically selecting bandwidth value for use in estimation of sparsity and coefficient covariance matrix: “hs” (Hall-Sheather), “bf” (Bofinger), “c” (Chamberlain).
bw = <i>number</i>	Use user-specified bandwidth value in place of automatic method specified in “bwmethod = ”.
bwsize = <i>number</i> (default = 0.05)	Size parameter for use in computation of bandwidth (used when “bw = hs” and “bw = bf”).
spmethod = <i>arg</i> (default = “kernel”)	Sparsity estimation method: “resid” (Siddiqui using residuals), “fitted” (Siddiqui using fitted quantiles at mean values of regressors), “kernel” (Kernel density using residuals) Note: “spmethod = resid” is not available when “cov = sandwich”.
btmethod = <i>arg</i> (default = “pair”)	Bootstrap method: “resid” (residual bootstrap), “pair” (xy-pair bootstrap), “mcbm” (MCMB bootstrap), “mcmbsa” (MCMB-A bootstrap).
btreps = <i>integer</i> (default = 100)	Number of bootstrap repetitions

<code>btseed = positive integer</code>	Seed the bootstrap random number generator. If not specified, EViews will seed the bootstrap random number generator with a single integer draw from the default global random number generator.
<code>btrnd = arg</code> (<i>default</i> = “kn” or method previously set using <code>rndseed</code> (p. 770))	Type of random number generator for the bootstrap: improved Knuth generator (“kn”), improved Mersenne Twister (“mt”), Knuth’s (1997) lagged Fibonacci generator used in EViews 4 (“kn4”) L’Ecuyer’s (1999) combined multiple recursive generator (“le”), Matsumoto and Nishimura’s (1998) Mersenne Twister used in EViews 4 (“mt4”).
<code>btobs = integer</code>	Number of observations for bootstrap subsampling (when “bsmethod = pair”). Should be significantly greater than the number of regressors and less than or equal to the number of observations used in estimation. EViews will automatically restrict values to the range from the number of regressors and the number of estimation observations. If omitted, the bootstrap will use the number of observations used in estimation.
<code>btout = name</code>	(optional) Matrix to hold results of bootstrap simulations.
<code>k = arg</code> (<i>default</i> = “e”)	Kernel function for sparsity and coefficient covariance matrix estimation (when “spmethod = kernel”): “e” (Epanechnikov), “r” (Triangular), “u” (Uniform), “n” (Normal–Gaussian), “b” (Biweight–Quartic), “t” (Triweight), “c” (Cosinus).
<code>m = integer</code>	Maximum number of iterations.
<code>s</code>	Use the current coefficient values in “C” or explicit coefficients as starting values estimation.
<code>s = number</code> (<i>default</i> = 0)	Determine starting values for equations. Specify a number between 0 and 1 representing the fraction of preliminary least squares coefficient estimates. Note that out of range values are set to the default.
<code>showopts / -showopts</code>	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
<code>p</code>	Print estimation results.

Examples

```
qreg y c x
```

estimates the default least absolute deviations (median) regression for the dependent variable Y on a constant and X. The estimates use the Huber Sandwich method for computing the covariance matrix, with individual sparsity estimates obtained using kernel methods. The bandwidth uses the Hall and Sheather formula.

```
qreg(quant=0.6, cov=boot, btmeth=mcmba) y c x
```

estimates the quantile regression for the 0.6 quantile using MCMB-A bootstrapping to obtain estimates of the coefficient covariance matrix.

Cross-references

See [Chapter 31. “Quantile Regression,” on page 259](#) of the *User’s Guide II* for a discussion of the quantile regression.

range	Commands
-------	--------------------------

Reset the workfile range for a regular frequency workfile.

No longer supported. See the replacement command [pagestruct](#) (p. 756).

read	Commands
------	--------------------------

Import data from a foreign disk file into series.

May be used to import data into an existing workfile from a text, Excel, or Lotus file on disk.

Syntax

```
read(options) [path\]file_name name1 [name2 name3 ...]  
read(options) [path\]file_name n
```

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. 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.

The input specification follows the source file name. 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 series in the file instead of a list of names. EViews will name the *n* series using the names 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.

To import data into alpha series, you must specify the names of your series, and should enter the tag “\$” following the series name (e.g., “NAME \$ INCOME CONSUMP”).

Options

File type options

t = dat, txt	ASCII (plain text) files.
t = wk1, wk3	Lotus spreadsheet files.
t = xls	Excel spreadsheet files.

If you do not specify the “t” option, EViews uses the file name extension to determine the file type. If you specify the “t” option, the file name extension will not be used to determine the file type.

Options for ASCII text files

t	Read data organized by series. Default is to read by observation with series in columns.
na = <i>text</i>	Specify text for NAs. Default is “NA”.
d = t	Treat tab as delimiter (note: you may specify multiple delimiter options). The <i>default</i> is “d = c” only.
d = c	Treat comma as delimiter.
d = s	Treat space as delimiter.
d = a	Treat alpha numeric characters as delimiter.
custom = <i>symbol</i>	Specify symbol/character to treat as delimiter.
mult	Treat multiple delimiters as one.
name	Series names provided in file.
label = <i>integer</i>	Number of lines between the header line and the data. Must be used with the “name” option.
rect (<i>default</i>) / norect	[Treat / Do not treat] file layout as rectangular.
skipcol = <i>integer</i>	Number of columns to skip. Must be used with the “rect” option.
skiprow = <i>integer</i>	Number of rows to skip. Must be used with the “rect” option.
comment = <i>symbol</i>	Specify character/symbol to treat as comment sign. Everything to the right of the comment sign is ignored. Must be used with the “rect” option.
singlequote	Strings are in single quotes, not double quotes.
dropstrings	Do not treat strings as NA; simply drop them.

<code>negparen</code>	Treat numbers in parentheses as negative numbers.
<code>allowcomma</code>	Allow commas in numbers (note that using commas as a delimiter takes precedence over this option).
<code>currency = sym- bol</code>	Specify symbol/character for currency data.

Options for spreadsheet (Lotus, Excel) files

<code>t</code>	Read data organized by series. Default is to read by observation with series in columns.
<code>letter_number</code> (default = "b2")	Coordinate of the upper-left cell containing data.
<code>s = sheet_name</code>	Sheet name for Excel 5–8 Workbooks.

Examples

```
read(t=dat,na=.) a:\mydat.raw id lwage hrs
```

reads data from an ASCII file MYDAT.RAW in the A: drive. The data in the file are listed by observation, the missing value NA is coded as a "." (dot or period), and there are three series, which are to be named ID, LWAGE, HRS (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 using names provided in the file.

```
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 [“Importing Data” on page 95](#) of the *User’s Guide I* for a discussion and examples of importing data from external files.

Unless you need to merge data into an *existing* workfile page, we recommend that you use the more powerful, easy-to-use tools for reading data (see [“Creating a Workfile by Reading from a Foreign Data Source” on page 41](#) of the *User’s Guide I*). See [pageload](#) (p. 750), and [wfoopen](#) (p. 802) for command details.

See also [write](#) (p. 812).

rename	Commands
--------	----------

Rename an object in the active workfile or database.

Syntax

```
rename old_name new_name
```

After the `rename` keyword, list the old object name followed by the new name. Note that the name specifications 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 the object TEMP_U to U2 in database AA.

```
rename a* b*
```

renames all objects beginning with the letter “A” to begin with the letter “B”.

Cross-references

See [Chapter 4. “Object Basics,” on page 63](#) of the *User’s Guide I* for a discussion of working with objects in EViews.

reset	Commands
-------	----------

Compute Ramsey’s regression specification error test.

Syntax

```
reset(n, options)
```

You must provide the number of powers of fitted terms n to include in the test regression.

Options

p	Print the test result.
---	------------------------

Examples

```
ls lwage c edu race gender
```

```
reset(2)
```

carries out the RESET test by including the square and the cube of the fitted values in the test equation.

Cross-references

See [“Ramsey’s RESET Test” on page 170](#) of the *User’s Guide II* for a discussion of the RESET test.

rndint	Commands
---------------	--------------------------

Generate uniform random integers.

The `rndint` command fills series, vector, and matrix objects with (pseudo) random integers drawn uniformly from zero to a user specified maximum. The `rndint` command ignores the current sample and fills the entire object with random integers.

Syntax

`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 *n* of the random integers. *n* should a positive integer.

Examples

```
series index
rndint(index,10)
```

fills the entire series INDEX with integers drawn randomly from 0 to 10. Note that unlike standard series assignment using `genr`, `rndint` ignores the current sample and fills the series for the entire workfile range.

```
sym(3) var3
rndint(var3,5)
```

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 [“Statistical Distribution Functions” on page 754](#) of the *User’s Guide I*

See also [nrnd \(p. 819\)](#), [rnd \(p. 821\)](#) and [rndseed \(p. 770\)](#).

rndseed	Commands
----------------	--------------------------

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.

Note that EViews 5 has updated the seeding routines of two of our pseudo-random number generators (backward compatible options are provided). It is strongly recommended that you use new generators.

Syntax

`rndseed(options) integer`

Follow the `rndseed` keyword with the optional generator type and an integer for the seed.

Options

<code>type = arg</code> (<code>default = "kn"</code>)	Type of random number generator: improved Knuth generator ("kn"), improved Mersenne Twister ("mt"), Knuth's (1997) lagged Fibonacci generator used in EViews 4 ("kn4"), L'Ecuyer's (1999) combined multiple recursive generator ("le"), Matsumoto and Nishimura's (1998) Mersenne Twister used in EViews 4 ("mt4").
--	---

When EViews starts up, the default generator type is set to the improved Knuth lagged Fibonacci generator. Unless changed using `rndseed`, Knuth's generator will be used for subsequent pseudo-random number generation.

	Knuth ("kn4")	L'Ecuyer ("le")	Mersenne Twister ("mt4")
Period	2^{129}	2^{319}	$2^{19937} - 1$
Time (for 10^7 draws)	27.3 secs	15.7 secs	1.76 secs
Cases failed Diehard test	0	0	0

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 ["Statistical Distribution Functions"](#) on [page 754](#) of the *User's Guide I*.

At press time, further information on the improved seeds may be found on the web at the following addresses:

Knuth generator: <http://sunburn.stanford.edu/~knuth/news02.html#rng>

Mersenne twister: <http://www.math.keio.ac.jp/~matumoto/MT2002/emt19937ar.html>

See also `nrnd` (p. 819), `rnd` (p. 821) and `rndint` (p. 770).

run	Commands
------------	-----------------

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

`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

<i>integer</i> (default = 1)	Set maximum errors allowed before halting the program.
c	Run program file without opening a window for display of the program file.
verbose / quiet	Verbose mode in which messages will be sent to the status line at the bottom of the EViews window (slower execution), or quiet mode which suppresses workfile display updates (faster execution).
v / q	Same as [verbose / quiet].
ver4 / ver5	Execute program in [version 4 / version 5] compatibility mode.

Examples

`run(q) 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:

```
run simul
print x
```

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 \(p. 874\)](#) statement.

Cross-references

See [“Executing a Program” on page 595](#) of the *User’s Guide I* for further details.

See also [include \(p. 874\)](#).

save	Commands
------	----------

Save current workfile to disk.

This usage is provided only for backward compatibility, as it has been replaced with the equivalent [wfsave \(p. 810\)](#) command.

Syntax

```
save [path\]file_name
```

Follow the keyword with a name for the file. If an explicit path is not specified, the file will be stored in the default directory, as set in the **File Locations** global options.

Examples

```
save MyWorkfile
```

saves the current workfile with the name `MYWORKFILE.WF1` in the default directory.

```
save c:\data\MyWF1
```

saves the current workfile with the name `MYWF1.WF1` in the specified directory.

Cross-references

See [wfsave \(p. 810\)](#).

seas	Commands
------	----------

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 [Series::x11](#) (p. 400) and [Series::x12](#) (p. 402).

Syntax

```
seas(options) series_name name_adjust [name_fac]
```

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.

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`.

Cross-references

See [“Seasonal Adjustment” on page 339](#) of the *User’s Guide I* for a discussion of seasonal adjustment methods.

See also [seasplot](#) (p. 651), [Series::x11](#) (p. 400), and [Series::x12](#) (p. 402).

setcell	Commands
---------	----------

Insert contents into cell of a table.

The `setcell` command puts a string or number into a cell of a table.

Syntax

```
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	Center the text/number in the cell.
r	Right-justify the text/number in cell.
l	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

```
setcell(tab1, 2, 1, "Subtotal")
```

puts the string “Subtotal” in row 2, column 1 of the table object named TAB1.

```
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 20](#) of the *User's Guide I* describes table formatting using commands. See [Chapter 15](#) of the *User's Guide I* for a discussion and examples of table formatting in EViews.

See also [Table::setWidth](#) (p. 529).

setcolwidth	Commands
-------------	----------

Set width of a column of a table.

Provided for backward compatibility. See [Table::setWidth \(p. 529\)](#) for the new method of setting the width of table and spreadsheet columns.

Syntax

```
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 20. “Working with Tables,” on page 675](#) of the *User’s Guide I* describes table formatting using commands. See also [Chapter 15. “Graphs, Tables, Text, and Spools,” on page 523](#) of the *User’s Guide I* for a discussion and examples of table formatting in EViews.

See also [Table::setWidth \(p. 529\)](#) and [Table::setheight \(p. 523\)](#).

setline	Commands
---------	----------

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 [Table::setlines \(p. 525\)](#).

Syntax

```
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 20. “Working with Tables,”](#) on page 675 of the *User’s Guide I* describes table formatting using commands. See also [Chapter 15. “Graphs, Tables, Text, and Spools,”](#) on page 523 of the *User’s Guide I* for a discussion and examples of table formatting in EViews.

See [Table::setlines](#) (p. 525) for more flexible line drawing tools.

shell	Commands
-------	--------------------------

Start the Windows command shell, optionally executing a command.

Syntax

```
shell(options) [arg1 arg2 arg3...]
```

See [spawn](#) (p. 783) for available options. By default, the Windows command shell will be started in hidden mode with the exit code for success set to zero.

Examples

```
shell mkdir c:\newdir
```

makes a new directory “c:\newdir”.

```
shell(out=flist) dir /b *.wfl
```

lists all workfiles in the current directory, saving output in a table named FLIST.

Cross-references

See [spawn](#) (p. 783) for details on spawning a new process.

show	Commands
------	--------------------------

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

```
show object_name.view_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 [“Object Commands” on page 582](#) of the *User’s Guide I* for discussion, and [Chapter 1. “Object View and Procedure Reference,” on page 3](#) for a complete listing of the views of the various objects.

See also [close \(p. 693\)](#).

smooth	Commands
--------	--------------------------

Exponential smoothing.

Forecasts a series using one of a number of exponential smoothing techniques. By default, `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.

`smooth` automatically calculates in-sample forecast errors and puts them into the series RESID.

Syntax

```
smooth(method) series_name smooth_name [freq]
```

You should follow the `smooth` keyword with the name of the series and a name for the smoothed series. You must also specify the smoothing method in parentheses. 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

<code>s[,x]</code>	Single exponential smoothing for series with no trend. You may optionally specify a number <i>x</i> between zero and one for the mean parameter.
<code>d[,x]</code>	Double exponential smoothing for series with a trend. You may optionally specify a number <i>x</i> between zero and one for the mean parameter.
<code>n[,x,y]</code>	Holt-Winters without seasonal component. You may optionally specify numbers <i>x</i> and <i>y</i> between zero and one for the mean and trend parameters, respectively.
<code>a[,x,y,z]</code>	Holt-Winters with additive seasonal component. You may optionally specify numbers <i>x</i> , <i>y</i> , and <i>z</i> , between zero and one for the mean, trend, and seasonal parameters, respectively.
<code>m[,x,y,z]</code>	Holt-Winters with multiplicative seasonal component. You may optionally specify numbers <i>x</i> , <i>y</i> , and <i>z</i> , between zero and one for the mean, trend, and seasonal parameters, respectively.

Other Options:

<code>p</code>	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

```
smooth(s) 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.

```
smooth(n,e,.3) tb3 tb3_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.

```
smpl @first @last-10
smooth(m,.1,.1,.1) order order_hw
smpl @all
graph gral.line order order_hw
show gral
```

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

See [“Exponential Smoothing” on page 354](#) of the *User’s Guide I* for a discussion of exponential smoothing methods.

smpl	Commands
------	--------------------------

Set sample range.

The `smpl` command sets the workfile sample to use for statistical operations and series assignment expressions.

Syntax

```
smpl smpl_spec
smpl 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 704](#) of the *User’s Guide I*. 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:

<code>@all</code>	The entire workfile range.
<code>@first</code>	The first observation in the workfile.
<code>@last</code>	The last observation in the workfile.

In panel settings, you may use the additional keywords:

<code>@firstmin</code>	The earliest of the first observations (computed across cross-sections).
<code>@firstmax</code>	The latest of the first observations.
<code>@lastmin</code>	The earliest of the last observations.
<code>@lastmax</code>	The latest of the last observations.

Examples

```
smpl 1955m1 1972m12
```

sets the workfile sample from 1955M1 to 1972M12.

```
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 86](#) of the *User’s Guide I* for a discussion of samples in EViews.

See also [Sample::set \(p. 352\)](#) and [Sample::sample \(p. 351\)](#).

solve	Commands
--------------	--------------------------

Solve the model.

`solve` finds the solution to a simultaneous equation model for the set of observations specified in the current workfile sample.

Syntax

`solve(options) model_name`

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. 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 [Model::solveopt \(p. 288\)](#).

Examples

```
solve mod1
```

solves the model MOD1 using the default solution method.

```
solve(m=500,e) nonlin2
```

solves the model NONLIN2 with an extended search of up to 500 iterations.

Cross-references

See [Chapter 36. “Models,” on page 407](#) of the *User’s Guide II* for a discussion of models.

See also [Model::model \(p. 284\)](#), [Model::msg \(p. 284\)](#) and [Model::solveopt \(p. 288\)](#).

sort	Commands
-------------	--------------------------

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

```
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 254](#) of the *User’s Guide I*.

spawn	Commands
-------	----------

Spawn a new process.

Syntax

```
spawn(options) filename [arg1 arg2 arg3...]
```

Follow the keyword with a filename indicating the process to spawn, and optional arguments to be passed to the process.

Options

"n" or "normal"	Create process in normal mode. The process will typically create a maximized window and may wait for user input.
"m" or "minimized"	Create process in a minimized window. Note that some applications may not accept requests to run in minimized mode.
"h" or "hidden"	Create process in hidden mode (without a visible window). Note that some applications may not accept requests to run in hidden mode.
t = <i>isecs</i>	Specifies the maximum time in seconds that EViews should wait for the process to complete. If the timeout interval is reached and the process has not completed, EViews will generate an error. A timeout setting of zero may be used to indicate that EViews should not wait for the spawned process to complete. If no timeout option is provided, EViews will wait indefinitely for the process to complete.
exit = <i>icode</i>	Specifies the exit code that the process will return if it is not completed successfully. If the process returns an exit code other than the specified value, EViews will generate an error. If the exit code option is not specified, EViews will never generate an error no matter what exit code is returned by the process.
out = <i>tablename</i>	If an output table name is specified, EViews will capture any data written to standard output by the spawned process and store it into a table object with the specified name in the workfile. Note that this option will have no effect unless either the minimized or hidden option is used and the timeout value is not zero.

Examples

```
spawn "c:\program files\microsoft office\office11\excel.exe"  
test.xls
```

starts a new Excel process, passing it the command line argument "test.xls".

Cross-references

See [shell](#) (p. 777) for information on starting a Windows command shell.

stats	Commands
-------	----------

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.

`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.

Syntax

```
stats(options) ser1 [ser2 ser3 ...]
```

Options

`p` Print the stats table.

Examples

```
stats height weight age
```

opens an untitled group window displaying the histogram and descriptive statistics for the common sample of the three series.

Cross-references

See [“Descriptive Statistics & Tests” on page 306](#) and [page 379](#) of the *User’s Guide I* for a discussion of the descriptive statistics views of series and groups.

See also [boxplot \(p. 613\)](#) and [hist \(p. 732\)](#).

statusline	Commands
------------	----------

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

```
statusline message_string
```

Examples

```
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 17. “EViews Programming,” on page 593](#) of the *User’s Guide I* for a discussion and examples of programs, control variables and string variables.

steps	Commands
-------	--------------------------

Estimation by stepwise least squares.

Syntax

`steps(options) y x1 [x2 x3 ...] @ z1 z2 z3`

Specify the dependent variable followed by a list of variables to be included in the regression, but not part of the search routine, followed by an “@” symbol and a list of variables to be part of the search routine. If no included variables are required, simply follow the dependent variable with an “@” symbol and the list of search variables.

Options

- method =
arg

Stepwise regression method: “stepwise” (default), “uni” (uni-directional), “swap” (swapwise), “comb” (combinatorial).
- nvars = int

Set the number of search regressors. Required for swapwise and combinatorial methods, optional for uni-directional and stepwise methods.

Stepwise and uni-directional method options

- back

Set stepwise or uni-directional method to run backward. If omitted, the method runs forward.
- tstat

Use *t*-statistic values as a stopping criterion. (Default uses *p*-values).
- ftol = number
(default = 0.5)

Set forward stopping criterion value.
- btol = number
(default = 0.5)

Set backward stopping criterion value.

<code>fmaxstep = int</code> (<i>default</i> = 1000)	Set the maximum number of steps forward.
<code>bmaxstep = int</code> (<i>default</i> = 1000)	Set the maximum number of steps backward.
<code>tmaxstep = int</code> (<i>default</i> = 2000)	Set the maximum total number of steps.

Swapwise method options

<code>minr2</code>	Use minimum R-squared increments. (Default uses maximum R-squared increments.)
--------------------	--

Combinatorial method options

<code>force</code>	Suppress the warning message issued when a large number of regressions will be performed.
--------------------	---

Examples

```
stepsls(method=comb,nvars=3) y c @ x1 x2 x3 x4 x5 x6 x7 x8
```

performs a combinatorial search routine to search for the three variables from the set of X1, X2, ..., X8, yielding the largest R-squared in a regression of Y on a constant and those three variables.

Cross-references

See [“Stepwise Least Squares Regression,” beginning on page 55.](#)

store	Commands
--------------	--------------------------

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.

Syntax

```
store(options) object_list
```

Follow the `store` command keyword with a list of 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 2 and earlier where the default was to store objects in individual databank files).*

You may 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 parenthe-

ses, in which case all objects without an explicit database name will be stored in the specified database.

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 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

<code>d = db_name</code>	Store to the specified database.
<code>i</code>	Store to individual databank files.
<code>1 / 2</code>	Store series in [single / double] precision to save space.
<code>o</code>	Overwrite object in database (default is to merge data, where possible).
<code>g = arg</code>	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 766](#) of the *User’s Guide II*.

Examples

```
store m1 gdp unemp
```

stores the three objects M1, GDP, UNEMP in the default database.

```
store(d=us1) m1 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) m1 c:\data\unemp
```

stores M1 and UNEMP in individual databank files.

Cross-references

“[Basic Data Handling](#)” on page 77 of the *User’s Guide I* discusses exporting data in other file formats. See [Chapter 10. “EViews Databases,”](#) on page 257 of the *User’s Guide I* for a discussion of EViews databases and databank files.

For additional discussion of wildcards, see [Appendix C. “Wildcards,”](#) on page 775 of the *User’s Guide I*.

See also [fetch](#) (p. 717) and [copy](#) (p. 696).

testadd	Commands
---------	----------

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

```
testadd(options) arg1 [arg2 arg3 ...]
```

List the names of the series or groups of series to test for omission after the keyword. The test is applied to the default equation, if defined.

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:

Cross-references

See “[Coefficient Tests](#)” on page 142 of the *User’s Guide II* for further discussion.

See also [testdrop](#) (p. 790).

testdrop	Commands
-----------------	--------------------------

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

```
testdrop(options) arg1 [arg2 arg3 ...]
```

List the names of the series or groups of series to test for omission after the keyword. The test is applied to the default equation, if defined.

Options

p	Print output from the test.
---	-----------------------------

Examples

```
ls sales c adver lsales ar(1)
testdrop adver
```

tests whether ADVER should be excluded from the specification for SALES. The commands:

Cross-references

See [“Coefficient Tests” on page 142](#) of the *User’s Guide II* for further discussion of testing coefficients.

See also [testadd \(p. 789\)](#) and [Equation::wald \(p. 93\)](#).

tic	Commands
------------	--------------------------

Reset the timer.

Syntax

```
Command:    tic
```

Examples

The sequence of commands:

```
tic
[some commands]
toc
```

resets the timer, executes commands, and then displays the elapsed time in the status line.
Alternatively:

```
tic
[some commands]
!elapsed = @toc
```

resets the time, executes commands, and saves the elapsed time in the control variable
!ELAPSED.

Cross-references

See also [toc](#) (p. 791) and [@toc](#) (p. 880).

toc	Commands
-----	--------------------------

Display elapsed time (since timer reset) in seconds.

Syntax

Command: **toc**

Examples

The sequence of commands:

```
tic
[some commands]
toc
```

resets the timer, executes commands, and then displays the elapsed time in the status line.
The set of commands:

```
tic
[some commands]
!elapsed = @toc
[more commands]
toc
```

resets the time, executes commands, saves the elapsed time in the control variable
!ELAPSED, executes additional commands, and displays the total elapsed time in the status
line.

Cross-references

See also [tic](#) (p. 790) and [@toc](#) (p. 880).

tsls	Commands
------	----------

Two-stage least squares.

Syntax

```
tsls(options) y x1 [x2 x3 ...] @ z1 [z2 z3 ...]  
tsls(options) specification @ z1 [z2 z3 ...]
```

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. You may estimate nonlinear equations or equations specified with formulas by first providing a specification, then listing the instrumental variables after an “@”-sign.

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.

Options

General options

m = <i>integer</i>	Set maximum number of iterations.
c = <i>number</i>	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.
deriv = <i>keyword</i>	Set derivative methods. The argument <i>keyword</i> 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.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
p	Print estimation results.

Additional Options for Non-Panel Equation estimation

w = <i>series_name</i>	Weighted TSLS. 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.

<code>s</code>	Use the current coefficient values in “C” as starting values for equations with AR or MA terms (see also param (p. 761)).
<code>s = number</code>	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”.
<code>z</code>	Turn off backcasting in ARMA models.

Additional Options for Panel Equation estimation

<code>cx = arg</code>	Cross-section effects. For fixed effects estimation, use “cx = f”; for random effects estimation, use “cx = r”.
<code>per = arg</code>	Period effects. For fixed effects estimation, use “cx = f”; for random effects estimation, use “cx = r”.
<code>wgt = arg</code>	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”).
<code>cov = arg</code>	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”).
<code>keepwgts</code>	Keep full set of GLS weights used in estimation with object, if applicable (by default, only small memory weights are saved).
<code>rancalc = arg</code> (<i>default = “sa”</i>)	Random component method: Swamy-Arora (“rancalc = sa”), Wansbeek-Kapteyn (“rancalc = wk”), Wallace-Hussain (“rancalc = wh”).
<code>nodf</code>	Do not perform degree of freedom corrections in computing coefficient covariance matrix. The default is to use degree of freedom corrections.
<code>coef = arg</code>	Specify the name of the coefficient vector (if specified by list); the default is to use the “C” coefficient vector.

<code>iter = arg</code> (<i>default</i> = “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.
<code>s</code>	Use the current coefficient values in “C” as starting values for equations with AR or MA terms (see also param (p. 761)).
<code>s = number</code>	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”.

Examples

```
tsls y_d c cpi inc ar(1) @ lw(-1 to -3)
```

estimates AN using TSLS regression of Y_D on a constant, CPI, INC with AR(1) using a constant, LW(-1), LW(-2), and LW(-3) as instruments.

```
param c(1) .1 c(2) .1
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.

Cross-references

See “Two-stage Least Squares” on page 37 and “Two-Stage Least Squares” on page 309 of the *User’s Guide II* for details on two-stage least squares estimation in single equations and systems, respectively. “Instrumental Variables” on page 502 of the *User’s Guide II* discusses estimation using pool objects, while “Instrumental Variables Estimation” on page 544 of the *User’s Guide II* discusses estimation in panel structured workfiles.

See also [ls](#) (p. 735).

ubreak	Commands
--------	----------

Andrews-Quandt test for unknown breakpoint.

Carries out the Andrews-Quandt test for parameter stability at some unknown breakpoint.

Syntax

```
ubreak(options) trimlevel @ x1 x2 x3
```

You must provide the level of trimming of the data. The level must be one of the following: 49, 48, 47, 45, 40, 35, 30, 25, 20, 15, 10, or 5. If the equation is specified by list and contains no linear terms, you may specify a subset of the regressors to be tested for a breakpoint after an “@” sign.

Options

wfname = <i>series_name</i>	Store the individual Wald F -statistics into the series <i>series_name</i> .
lname = <i>series_name</i>	Store the individual likelihood ratio F -statistics into the series <i>series_name</i> .
p	Print the result of the test.

Examples

```
ls log(spot) c log(p_us) log(p_uk)
ubreak 15
```

regresses the log of SPOT on a constant, the log of P_US, and the log of P_UK, and then carries out the Andrews-Quandt test, trimming 15% of the data from each end.

To test whether only the constant term and the coefficient on the log of P_US are subject to a structural break, use:

```
ubreak @ c log(p_us)
```

Cross-references

See [“Quandt-Andrews Breakpoint Test” on page 166](#) of the *User’s Guide II* for further discussion.

See also [Equation::chow](#) (p. 41) and [Equation::rls](#) (p. 84).

unlink	Commands
--------	----------

Break links in series objects.

Syntax

```
unlink link_names
```

`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.

Examples

```
unlink gdp income
```

converts the link series GDP and INCOME to ordinary series.

```
unlink *
```

breaks all links in the current workfile page.

Cross-references

See [Chapter 8. “Series Links,” on page 173](#) of the *User’s Guide I* for a detailed description of link objects. See also [Link::link \(p. 225\)](#) and [Link::linkto \(p. 226\)](#).

uroot	Commands
-------	----------

Carries out unit root tests on a single series, pool series, group of series, or panel structured series.

The ordinary, single series 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).

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 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

```
uroot(options) object_name
```

where *object_name* can be the name of a series, group or pool object.

Options

Basic Specification Options

You should specify the exogenous variables and order of dependent variable differencing in the test equation using the following options:

<code>const</code> (<i>default</i>)	Include a constant in the test equation.
<code>trend</code>	Include a constant and a linear time trend in the test equation.
<code>none</code>	Do not include a constant or time trend (only available for the ADF and PP tests).
<code>dif = integer</code> (<i>default</i> = 0)	Order of differencing of the series prior to running the test. Valid values are {0, 1, 2}.

For backward compatibility, the shortened forms of these options, “c”, “t”, and “n”, are presently supported. For future compatibility we recommend that you use the longer forms.

For ordinary (non-panel) unit root tests, you should specify the test type using one of the following keywords:

<code>adf</code> (<i>default</i>)	Augmented Dickey-Fuller.
<code>dfgls</code>	GLS detrended Dickey-Fuller (Elliot, Rothenberg, and Stock).
<code>pp</code>	Phillips-Perron.
<code>kpss</code>	Kwiatkowski, Phillips, Schmidt, and Shin.
<code>ers</code>	Elliot, Rothenberg, and Stock (Point Optimal).
<code>np</code>	Ng and Perron.

For panel testing, you may use one of the following keywords to specify the test:

<code>sum</code> (<i>default</i>)	Summary of all of the panel unit root tests.
<code>llc</code>	Levin, Lin, and Chu.
<code>breit</code>	Breitung.
<code>ips</code>	Im, Pesaran, and Shin.
<code>adf</code>	Fisher - ADF.
<code>pp</code>	Fisher - PP.
<code>hadri</code>	Hadri.

Options for ordinary (non-panel) unit root tests

<code>hac = arg</code>	<p>Method of estimating the frequency zero spectrum: “bt” (Bartlett kernel), “pr” (Parzen kernel), “qs” (Quadratic Spectral kernel), “ar” (AR spectral), “ardt” (AR spectral - OLS detrended data), “argls” (AR spectral - GLS detrended data).</p> <p>Applicable to PP, KPSS, ERS, and NP tests. <i>The default settings are test specific</i> (“bt” for PP and KPSS, “ar” for ERS, “argls” for NP).</p>
<code>band = arg,</code> <code>b = arg</code> <code>(default = “nw”)</code>	<p>Method of selecting the bandwidth: “nw” (Newey-West automatic variable bandwidth selection), “a” (Andrews automatic selection), “number” (user specified bandwidth).</p> <p>Applicable to PP, KPSS, ERS, and NP tests when using kernel sums-of-covariances estimators (where “hac = ” is one of {bt, pz, qs}).</p>
<code>lag = arg</code> <code>(default = “a”)</code>	<p>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).</p> <p>Applicable to ADF and DFGLS tests, and for the other tests when using AR spectral density estimators (where “hac = ” is one of {ar, ardt, argls}).</p>
<code>info = arg</code> <code>(default = “sic”)</code>	<p>Information criterion to use when computing automatic lag length selection: “aic” (Akaike), “sic” (Schwarz), “hqc” (Hannan-Quinn), “msaic” (Modified Akaike), “msic” (Modified Schwarz), “mhqc” (Modified Hannan-Quinn).</p> <p>Applicable to ADF and DFGLS tests, and for other tests when using AR spectral density estimators (where “hac = ” is one of {ar, ardt, argls}).</p>
<code>maxlag = integer</code>	<p>Maximum lag length to consider when performing automatic lag length selection:</p> $\text{default} = \text{int}((12 T / 100)^{0.25})$ <p>Applicable to ADF and DFGLS tests, and for other tests when using AR spectral density estimators (where “hac = ” is one of {ar, ardt, argls}).</p>

Options for panel unit root tests

The following panel specific options are available:

balance	Use balanced (across cross-sections or series) data when performing test.
hac = <i>arg</i> (<i>default</i> = “bt”)	Method of estimating the frequency zero spectrum: “bt” (Bartlett kernel), “pr” (Parzen kernel), “qs” (Quadratic Spectral kernel). Applicable to “Summary”, LLC, Fisher-PP, and Hadri tests.
band = <i>arg</i> , b = <i>arg</i> (<i>default</i> = “nw”)	Method of selecting the bandwidth: “nw” (Newey-West automatic variable bandwidth selection), “a” (Andrews automatic selection), <i>number</i> (user-specified common bandwidth), <i>vector_name</i> (user-specified individual bandwidth). Applicable to “Summary”, LLC, Fisher-PP, and Hadri tests.
lag = <i>arg</i>	Method of selecting lag length (number of first difference terms) to be included in the regression: “a” (automatic information criterion based selection), <i>integer</i> (user-specified common lag length), <i>vector_name</i> (user-specific individual lag length). If the “balance” option is used, $default = \begin{cases} 1 & \text{if } (T_{\min} \leq 60) \\ 2 & \text{if } (60 < T_{\min} \leq 100) \\ 4 & \text{if } (T_{\min} > 100) \end{cases}$ where T_{\min} is the length of the shortest cross-section or series, otherwise <i>default</i> = “a”. Applicable to “Summary”, LLC, Breitung, IPS, and Fisher-ADF tests.
info = <i>arg</i> (<i>default</i> = “sic”)	Information criterion to use when computing automatic lag length selection: “aic” (Akaike), “sic” (Schwarz), “hqc” (Hannan-Quinn), “msaic” (Modified Akaike), “msic” (Modified Schwarz), “mhqc” (Modified Hannan-Quinn). Applicable to “Summary”, LLC, Breitung, IPS, and Fisher-ADF tests.
maxlag = <i>arg</i>	Maximum lag length to consider when performing automatic lag length selection, where <i>arg</i> is an <i>integer</i> (common maximum lag length) or a <i>vector_name</i> (individual maximum lag length) $default = \text{int}(\min_i(12, T_i/3) \cdot (T_i/100)^{0.25})$ where T_i is the length of the cross-section or series.

Other options

p	Print output from the test.
---	-----------------------------

Examples

The command:

```
uroot(adf,const,lag=3,save=mout) gdp
```

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.

```
uroot(dfqls,trend,info=sic) ip
```

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.

```
uroot(kpss,const,hac=pr,b=2.3) unemp
```

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.

```
uroot(np,hac=ardt,info=maic) sp500
```

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 88](#) of the *User’s Guide II* for discussion of standard unit root tests performed on a single series, and [“Panel Unit Root Tests” on page 100](#) of the *User’s Guide II* for discussion of unit roots tests performed on panel structured workfiles, groups of series, or pooled data.

varest	Commands
--------	--------------------------

Specify and estimate a VAR or VEC.

Syntax

```
varest(options) lag_pairs endog_list [@ exog_list]
varest(method = ec, trend, n) lag_pairs endog_list [@ exog_list]
```

The first form of the command estimates a VAR. It is the interactive command equivalent of using `ls` to estimate a named VAR object (see [Var::ls \(p. 561\)](#) for syntax and details).

The second form of the command estimates a VEC. It is the interactive command equivalent of using `ec` to estimate a named VAR object (see [Var::ec](#) (p. 553) for syntax and details).

Examples

```
varest 1 3 m1 gdp
```

estimates an unnamed unrestricted VAR with two endogenous variables (M1 and GDP), a constant and 3 lags (lags 1 through 3).

```
varest(noconst) 1 3 m1 gdp
```

estimates the same VAR, but with no constant term included in the specification.

```
varest(method=ec) 1 4 m1 gdp tb3
```

estimates a VEC with four lagged first differences, three endogenous variables and one cointegrating equation using the default trend option “c”.

```
varest(method=ec,b,2) 1 2 4 4 tb1 tb3 tb6 @ d2 d3 d4
```

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::var](#) (p. 572), [Var::ls](#) (p. 561) and [Var::ec](#) (p. 553).

wfcreate	Commands
----------	----------

Create a new workfile. The workfile becomes the active workfile.

Syntax

```
wfcreate(options) frequency start_date end_date [num_cross_sections]
wfcreate(options) u num_observations
```

The first form of the command may be used to create a new regular frequency workfile with the specified frequency, start, and end date. The frequency may be specified as “a” (annual), “s” (semi-annual), “q” (quarterly), “m” (monthly), “w” (weekly), “d” (5-day daily), “7” (7-day daily). 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. 756).

The second form of the command is used to create an unstructured workfile with the specified number of observations.

Options

<code>wf = wf_name</code>	Optional name for the new workfile.
<code>page = page_name</code>	Optional name for the page in the new workfile.

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 the GRILICHES_GRUNFELD workfile containing a paged named “ANNUAL” with 10 cross-sections of annual data for the years 1935 to 1954.

Cross-references

See also [pagecreate \(p. 746\)](#) and [pagedelete \(p. 750\)](#).

wfopen	Commands
---------------	--------------------------

Open a workfile. Reads in a previously saved workfile from disk, or reads the contents of a foreign data source into a new workfile.

The opened workfile becomes the default workfile; existing workfiles in memory remain on the desktop but become inactive.

Syntax

```
wfopen [path\]workfile_name
wfopen(options) source_description [@keep keep_list] [@drop drop_list] [@keepmap
keepmap_list] [@dropmap dropmap_list] [@selectif condition]
wfopen(options)source_description table_description [@keep keep_list] [@drop
drop_list] [@keepmap keepmap_list] [@dropmap dropmap_list] [@selectif con-
dition]
```

The workfile or external data source is specified as the first argument following the command keyword and options. In most cases, the external data source is a file, so the *source_description* will be the name of the file. Alternatively, the external data source may be the output from a web server, in which case the URL should be provided as the

source_description. Likewise, reading from an ODBC query, the ODBC DSN (data source name) should be used to identify the *source_description*.

If the *source_description* contains spaces, it must be enclosed in (double) quotes, to separate it from the table description.

In cases where there is more than one table that could be formed from the specified external data source, a *table_description* may be provided to select the desired table. For example, when reading from an Excel file, an optional cell range may be provided to specify which data are to be read from the spreadsheet. When reading from an ODBC data source, a SQL query or table name must be used to specify the table of data to be read. When working with a text file, a *table_description* of how to break up the file into columns and rows is required.

Note that ODBC support is provided only in the EViews 5 Enterprise Edition.

Excel, HTML, Text, Binary File Table Descriptors

The following statements may be used in the table descriptions for Excel, HTML, text or binary data sources:

Excel Files

- “range = *arg*”, where *arg* is a range of cells to read from the excel workbook, following the standard excel format [*worksheet!*][*topleft_cell*:*bottomright_cell*].

If the worksheet name contains spaces, it should be placed in single quotes. If the worksheet name is omitted, the cell range is assumed to refer to the currently active sheet. If only a top left cell is provided, a bottom right cell will be chosen automatically to cover the range of non-empty cells adjacent to the specified top left cell. If only a sheet name is provided, the first set of non-empty cells in the top left corner of the chosen worksheet will be selected automatically. As an alternative to specifying an explicit range, a name which has been defined inside the excel workbook to refer to a range or cell may be used to specify the cells to read.

HTML Pages

- “table = *arg*”, where *arg* specifies which table to read in an HTML file/page containing multiple tables.

When specifying *arg*, you should remember that tables are named automatically following the pattern “Table01”, “Table02”, “Table03”, *etc.* If no table name is specified, the largest table found in the file will be chosen by default. Note that the table numbering may include trivial tables that are part of the HTML content of the file, but would not normally be considered as data tables by a person viewing the page.

- “skip = *int*”, where *int* is the number of rows to discard from the top of the HTML table.

Text/Binary Files

- “ftype = [ascii|binary]” specifies whether numbers and dates in the file are stored in a human readable text (ASCII), or machine readable (Binary) form.
- “rectype = [crlf|fixed|streamed]” describes the record structure of the file:
 - “crlf”, each row in the output table is formed using a fixed number of lines from the file (where lines are separated by carriage return/line feed sequences). This is the default setting.
 - “fixed”, each row in the output table is formed using a fixed number of characters from the file (specified in “reclen = *arg*”). This setting is typically used for files that contain no line breaks.
 - “streamed”, each row in the output table is formed by reading a fixed number of fields, skipping across lines if necessary. This option is typically used for files that contain line breaks, but where the line breaks are not relevant to how rows from the data should be formed.
- “reclines = *int*”, number of lines to use in forming each row when “rectype = crlf” (default is 1).
- “reclen = *int*”, number of bytes to use in forming each row when “rectype = fixed”.
- “recfields = *int*”, number of fields to use in forming each row when “rectype = streamed”.
- “skip = *int*”, number of lines (if rectype is “crlf”) or bytes (if rectype is not “crlf”) to discard from the top of the file.
- “comment = *string*”, where *string* is a double-quoted string, specifies one or more characters to treat as a comment indicator. When a comment indicator is found, everything on the line to the right of where the comment indicator starts is ignored.
- “emptylines = [keep|drop]”, specifies whether empty lines should be ignored (“drop”), or treated as valid lines (“keep”) containing missing values. The default is to ignore empty lines.
- “tabwidth = *int*”, specifies the number of characters between tab stops when tabs are being replaced by spaces (default = 8). Note that tabs are automatically replaced by spaces whenever they are not being treated as a field delimiter.
- “fieldtype = [delim|fixed|streamed|undivided]”, specifies the structure of fields within a record:
 - “Delim”, fields are separated by one or more delimiter characters
 - “Fixed”, each field is a fixed number of characters

“Streamed”, fields are read from left to right, with each field starting immediately after the previous field ends.

“Undivided”, read entire record as a single series.

- “quotes = [single|double|both|none]”, specifies the character used for quoting fields, where “single” is the apostrophe, “double” is the double quote character, and “both” means that either single or double quotes are allowed (default is “both”). Characters contained within quotes are never treated as delimiters.
- “singlequote“, same as “quotes = single”.
- “delim = [comma|tab|space|dblspace|white|dblwhite]”, specifies the character(s) to treat as a delimiter. “White” means that either a tab or a space is a valid delimiter. You may also use the abbreviation “d = ” in place of “delim = ”.
- “custom = “arg1””, specifies custom delimiter characters in the double quoted string. Use the character “t” for tab, “s” for space and “a” for any character.
- “mult = [on|off]”, to treat multiple delimiters as one. Default value is “on” if “delim” is “space”, “dblspace”, “white”, or “dblwhite”, and “off” otherwise.
- “endian = [big|little]”, selects the endianness of numeric fields contained in binary files.
- “string = [nullterm|nullpad|spacepad]”, specifies how strings are stored in binary files. If “nullterm”, strings shorter than the field width are terminated with a single zero character. If “nullpad”, strings shorter than the field width are followed by extra zero characters up to the field width. If “spacepad”, strings shorter than the field width are followed by extra space characters up to the field width.

The most important part of the Text/Binary table description is the format statement. You may specify the data format using the following descriptors:

- “fformat = (arg1, arg2, ...)”, Fortran format specification (see syntax below).
- “rformat = (arg1, arg2, ...)” column range format specification (see syntax below).
- “cformat = “fmt “, C printf/scanf format specification.

Fortran Syntax

The Fortran syntax follows the rules of the standard Fortran format statement. The syntax follows the general form:

- “fformat = ([n1]Type[Width][.Precision], [n2]Type[Width][.Precision], ...)”.

where *Type* specifies the underlying data type, and may be one of the following,

I - integer

F - fixed precision

E - scientific

A - alphanumeric

X - skip

and *n1*, *n2*, ... are the number of times to read using the descriptor (default = 1). More complicated Fortran compatible variations on this format are possible.

Column Range Syntax

- “rformat = ([*ser_name1*][*type*] *n1*[-*n2*], [*ser_name1*] [*type*] *n3*[-*n4*], ...)”

where optional type is “\$” for string or “#” for number, and *n1*, *n2*, *n3*, *n4*, *etc.* are the range of columns containing the data.

All Types

- “colhead = *int*”, number of table rows to be treated as column headers.
- “namepos = [first|last|all|none]”, which row(s) of the column headers should be used to form the column name. The setting “first” refers to first line, “last” is last line, “all” is all lines and “none” is no lines. The remaining column header rows will be used to form the column description. The default setting is “all”.
- “Nonames”, the file does not contain a column header (same as “colhead = 0”).
- “names = (“*arg1*”, “*arg2*”, ...)”, user specified column names, where *arg1*, *arg2*, ... are names of the first series, the second series, *etc.* when names are provided, these override any names that would otherwise be formed from the column headers.
- “descriptions = (“*arg1*”, “*arg2*”, ...)”, user specified descriptions of the series. If descriptions are provided, these override any names that would otherwise be formed from the column headers.
- “na = “*arg1*””, text used to represent observations that are missing from the file. The text should be enclosed on double quotes.
- “scan = [*int*| all]”, number of rows of the table to scan during automatic format detection (“scan = all” scans the entire file).
- “firstobs = *int*”, first observation to be imported from the table of data (default is 1). This option may be used to start reading rows from partway through the table.
- “lastobs = *int*”, last observation to be read from the table of data (default is last observation of the file). This option may be used to read only part of the file, which may be useful for testing.

ODBC or Microsoft Access Table Descriptors

When reading from an ODBC or Microsoft Access data source, you must provide a table description to indicate the table of data to be read. You may provide this information on one

of two ways: by entering the name of a table in the data source, or by including an SQL query statement enclosed in double quotes.

Options

`type = arg / t = arg` Optional type specification: Access database file (“access”), Aremos-TSD file (“a”, “aremos”, “tsd”), Binary file (“binary”), Excel file (“excel”), Gauss dataset file (“gauss”), GiveWin/PcGive 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 (“l”, “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”).

`wf = wf_name` Optional name for the new workfile.

`page = page_name` Optional name for the page in the new workfile.

Examples

The following examples illustrate the use of `wfoopen` to open an existing workfiles or to read foreign data into a new workfile.

EViews and MicroTSP

```
wfoopen c:\data\macro
```

loads a previously saved EViews workfile MACRO.WF1 from the DATA directory in the C drive.

```
wfoopen 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 specifies the type explicitly:

```
wfoopen(type=dos) nipa
```

Excel

```
wfoopen "c:\data files\data.xls"
```

loads the active sheet of DATA.XLS into a new workfile.

```
wfoopen(page=GDP) "c:\data files\data.xls" range="GDP data" @drop X
```

reads the data contained in the “GDP data” sheet of DATA.XLS into a new workfile. The data for the series X is dropped, and the name of the new workfile page is “GDP”.

```
wfopen(type=excel, wf=PRCS) c:\data.xls range="prices" @keep s*  
@drop *ic
```

loads into the new PRCS workfile a subset of the data contained in the PRICES data sheet of DATA.XLS. The load keeps only the series that begin with “S”, but does not include the ones that end in “IC”.

```
wfopen c:\data.xls range='sheet1 data'!B2:D30
```

loads into a new workfile the data contained from cells B2 through D30 on worksheet “sheet 1 data” in the workbook DATA.XLS.

Stata

```
wfopen(type=stata) c:\data.dta @dropmap map1 map2
```

load a Stata file DATA.DTA into a new workfile, dropping map MAP1 and MAP2.

SAS

```
wfopen(type=sasxport) c:\data.xpt
```

loads a sas transport file data.xpt into a new workfile.

```
wfopen c:\inst.sas
```

creates a workfile by reading from external data using the SAS program statements in INST.SAS. The program may contain a limited set of SAS statements which are commonly used in reading in a data file.

ASCII text files (.txt, .csv, etc.)

```
wfopen c:\data.csv skip=5, names=(gdp, inv, cons)
```

reads DATA.CSV into a new workfile page, skipping the first 5 rows and naming the series GDP, INV, and CONS.

```
wfopen(type=text) c:\date.txt delim=comma
```

loads the comma delimited data DATE.TXT into a new workfile.

```
wfopen(type=raw, rectype=fixed) c:\data.txt skip=8, for-  
mat=(F10, X23, A4)
```

loads a text file with fixed length data into a new workfile, skipping the first 8 rows. The reading is done as follows: read the first 10 characters as a fixed precision number, after that, skip the next 23 characters (X23), and then read the next 4 characters as strings (A4).

```
wfopen(type=raw, rectype=fixed) c:\data.txt format=2(4F8, 2I2)
```

loads the text file as a workfile using the specified explicit format. The data will be a repeat of four fixed precision numbers of length 8 and two integers of length 2. This is the same description as “format = (F8,F8,F8,F8,I2,I2,F8,F8,F8,F8,I2,I2)”.

```
wfopen(type=raw, rectype=fixed) c:\data.txt rformat=(GDP 1-2 INV 3  
CONS 6-9)
```

loads the text file as a workfile using column range syntax. The reading is done as follows: the first series is located at the first and second character of each row, the second series occupies the 3rd character, the third series is located at character 6 through 9. The series will named GDP, INV, and CONS.

Html

```
wfopen "c:\data.html"
```

loads into a new workfile the data located on the HTML file DATA.HTML located on the C:\ drive

```
wfopen(type=html) "http://www.tradingroom.com.au/apps/mkt/forex.ac" colhead=3, namepos=first
```

loads into a new workfile the data with the given URL located on the website site “http://www.tradingroom.com.au”. The column header is set to three rows, with the first row used as names for columns, and the remaining two lines used to form the descriptions.

ODBC (Enterprise Edition only)

```
wfopen c:\data.dsn CustomerTable
```

opens in a new workfile the table named CUSTOMERTABLE from the ODBC database described in the DATA.DSN file.

```
wfopen(type=odbc) "my server" "select * from customers where id>30"
@keep p*
```

opens in a new workfile with SQL query from database using the server “MY SERVER”, keeping only variables that begin with P. The query selects all variables from the table CUSTOMERS where the ID variable takes a value greater than 30.

Cross-references

See [Chapter 3. “Workfile Basics,”](#) on [page 37](#) of the *User’s Guide I* for a basic discussion of workfiles.

See also [pageload](#) (p. 750), [read](#) (p. 766), [fetch](#) (p. 717), [wfsave](#) (p. 810), and [pagesave](#) (p. 752).

wfsave	Commands
--------	----------

Save the default workfile as an EViews workfile (.wf1 file) or as a foreign file or data source.

Syntax

```
wfsave(options) [path\]filename
wfsave(options) source_description [@keep keep_list] [@drop drop_list] [@keepmap
keepmap_list] [@dropmap dropmap_list] [@smp1 smp1_spec]
wfsave(options) source_description table_description [@keep keep_list] [@drop
drop_list] [@keepmap keepmap_list] [@dropmap dropmap_list] [@smp1
smp1_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. See [wfoopen \(p. 802\)](#) for details on the syntax for *source_descriptions* and *table_descriptions*.

Options

Workfile Save Options

1	Save using single precision.
2	Save using double precision.
c	Save compressed workfile (not compatible with EViews versions prior to 5.0).

The default workfile save settings use the Global Options.

Foreign Source Save Options

<code>type = arg, t = arg</code>	Optional type specification. Access database file (“access”), Aremos-TSD file (“a”, “aremos”, “tsd”), Binary file (“binary”), EViews database file (“e”, “evdb”), Excel file (“excel”), Gauss dataset file (“gauss”), GiveWin/PcGive 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 (“l”, “trl”), SAS transport file (“sasxport”), SPSS file (“spss”), SPSS portable file (“spssport”), Stata file (“stata”), Text file (“text”), TSP portable file (“t”, “tsp”).
<code>mode = create</code>	Create new file only; error on attempt to overwrite.
<code>maptype = arg</code>	Write selected maps as: numeric (“n”), character (“c”), both numeric and character (“b”).
<code>nomapval</code>	Do not write mapped values for series with attached value labels (the default is to write the mapped values if available).

Cross-references

See also [pagesave \(p. 752\)](#), [wfopen \(p. 802\)](#), and [pageload \(p. 750\)](#).

wfselect	Commands
-----------------	--------------------------

Make the selected workfile page the active workfile page.

Syntax

```
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
```

both change the default workfile to MYPROJECT. The first command uses the default active page, while the second changes the page to PAGE2.

Cross-references

See also [pageselect](#) (p. 753).

workfile	Commands
-----------------	--------------------------

Create or change workfiles.

No longer supported; provided for backward compatibility. This command has been replaced by [wfcreate](#) (p. 801) and [pageselect](#) (p. 753).

write	Commands
--------------	--------------------------

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

write(options) [path\]filename arg1 [arg2 arg3 ...]

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.

Note that EViews cannot, at present, write into an existing file. The file that you select will, if it exists, 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

<code>na = string</code>	Specify text string for NAs. Default is “NA”.
<code>names (default) / nonames</code>	[Write / Do not write] series names.
<code>dates (default) / nodates</code>	[Write / Do not write] dates/obs.
<code>d = arg</code>	Specify delimiter (<i>default</i> is tab): “s” (space), “c” (comma).
<code>t</code>	Write by series. Default is to write by obs with series in columns.

Spreadsheet (Lotus, Excel) files

<code>letter_number</code>	Coordinate of the upper-left cell containing data.
<code>names (default) / nonames</code>	[Write / Do not write] series names.
<code>dates (default) / nodates</code>	[Write / Do not write] dates/obs.
<code>dates = arg</code>	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).
<code>t</code>	Write by series. Default is to write by obs with series in columns.

Examples

```
write(t=txt,na=.,d=c,dates) a:\dat1.csv hat1 hat_sel
```

Writes the two series HAT1 and HAT_SE1 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.

```
write(t=txt,na=.,d=c,dates) dat1.csv hat1 hat_sel
```

writes the same file in the default directory.

Cross-references

See [“Exporting to a Spreadsheet or Text File” on page 105](#) of the *User’s Guide I* for a discussion.

See also [pagesave \(p. 752\)](#) and [read \(p. 766\)](#).

References

- Knuth, D. E. (1997). *The Art of Computer Programming, Volume 2, Semi-numerical Algorithms*, 3rd edition, Reading, MA: Addison-Wesley Publishing Company. *Note: the C implementation of the lagged Fibonacci generator is described in the errata to the 2nd edition, downloadable from Knuth's web site.*
- L'Ecuyer, P. (1999). "Good Parameters and Implementations for Combined Multiple Recursive Random Number Generators," *Operations Research*, 47(1), 159-164
- MacKinnon, James G., Alfred A. Haug, and Leo Michelis (1999), "Numerical Distribution Functions of Likelihood Ratio Tests For Cointegration," *Journal of Applied Econometrics*, 14, 563-577.
- Matsumoto, M. and T. Nishimura (1998). "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator," *ACM Transactions on Modeling and Computer Simulation*, 8(1), 3-30.
- Osterwald-Lenum, Michael (1992). "A Note with Quantiles of the Asymptotic Distribution of the Maximum Likelihood Cointegration Rank Test Statistics," *Oxford Bulletin of Economics and Statistics*, 54, 461-472.
- Ravn, Morten O. and Harald Uhlig (2002). "On Adjusting the Hodrick-Prescott Filter for the Frequency of Observations," *Review of Economics and Statistics*, 84, 371-375

Chapter 5. Special Expression Reference

The following reference is an alphabetical listing of special expressions that may be used in series assignment and generation, or as terms in estimation specifications

Special Expression Summary

- [ar](#).....autoregressive error specification (p. 815).
- [@expand](#)automatic dummy variables (p. 816).
- [ma](#)moving average error specification (p. 818).
- [na](#)not available (missing value) code (p. 818).
- [nrnd](#)normal random number generation (p. 819).
- [pdl](#).....polynomial distributed lag specification (p. 820).
- [rnd](#)uniform random number generation (p. 821).
- [sar](#)seasonal autoregressive error specification (p. 821).
- [sma](#).....seasonal moving average error specification (p. 822).

Special Expression Entries

The following section provides an alphabetical listing of the commands and functions associated with the EViews programming language. Each entry outlines the basic syntax and and provides examples and cross references.

ar	Special Expression
----	------------------------------------

Autoregressive error specification.

The AR specification can appear in an [ls](#) (p. 735) or [tsls](#) (p. 792) 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 [Chapter 26. “Time Series Regression,”](#) on [page 63](#) of the *User’s Guide II* for details on ARMA and seasonal ARMA modeling.

See also [sar](#) ([p. 821](#)), [ma](#) ([p. 818](#)), and [sma](#) ([p. 822](#)).

@expand	Special Expression
----------------	------------------------------------

Automatic dummy variables.

The `@expand` expression may be added in estimation to indicate the use of one or more automatically created dummy variables.

Syntax

`@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 dummy variables defined above, but no dummy is created for “SEX = 0, REGION = “North””. In the expression:

```
@expand(sex, region, @droplast)
```

no dummy is created for “SEX = 1, REGION = “WEST””.

The expression:

```
@expand(sex, region, @drop(0, "West"), @drop(1, "North"))
```

creates 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.

```
eq.ls income @expand(sex) *age
```

regresses INCOME on regressor AGE with category specific slopes, one for “SEX = 0” and one for “SEX = 1”.

Cross-references

See [“Automatic Categorical Dummy Variables” on page 28](#) of the *User’s Guide II* for further discussion.

ma	Special Expression
----	------------------------------------

Moving average error specification.

The `ma` specification may be added in an `ls` ([p. 735](#)) or `tsls` ([p. 792](#)) 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 “[Time Series Regression](#)” on [page 63](#) of the *User’s Guide II* for details on ARMA and seasonal ARMA modeling.

See also `sma` ([p. 822](#)), `ar` ([p. 815](#)), and `sar` ([p. 821](#)).

na	Special Expression
----	------------------------------------

Not available code. “NA” is used to represent missing observations.

Examples

```
smpl if y >= 0
series z = y
smpl if y < 0
z = na
```

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:

```
series y = nrnd
y = log(y)
```

will replace all positive value of Y with $\log(Y)$ and all negative values with “NA”.

```
series test = (yt <> na)
```

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 2. Previously, NA values were coded as 1e-37. This implied that in EViews 2, you could use the expression:

```
series z = (y>=0)*x + (y<0)*na
```

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 `smpl` statement as in the first example above, or the `@recode` or `@nan` function.

Cross-references

See [“Missing Values” on page 129](#) of the *User’s Guide I* for a discussion of working with missing values in EViews.

nrnd	Special Expression
------	------------------------------------

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 754 for a list of other random number generating functions from various distributions.

See also [rnd](#) (p. 821), [rndint](#) (p. 770) and [rndseed](#) (p. 770).

pdl	Special Expression
-----	------------------------------------

Polynomial distributed lag specification.

This expression allows you to estimate polynomial distributed lag specifications in `ls` or `tsls` estimation. `pdl` forces the coefficients of a distributed lag to lie on a polynomial. The expression can only be used in estimation by list.

Syntax

```
pdl(series_name, lags, order[,options])
```

Options

The PDL specification must be provided in parentheses after the keyword `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

```
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.

```
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.

```
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 “pdl(*)” in the instrument list instead of repeating the full PDL specification.

Cross-references

See [“Polynomial Distributed Lags \(PDLs\)” on page 23](#) of the *User’s Guide II* for further discussion.

rnd	Special 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

```
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 [“Statistical Distribution Functions” on page 754](#) of the *User’s Guide I*.

See also [nrnd \(p. 819\)](#), [rndint \(p. 770\)](#) and [rndseed \(p. 770\)](#).

sar	Special 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

```
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.

```
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 71](#) of the *User’s Guide II* for details on ARMA and seasonal ARMA modeling.

See also [sma \(p. 822\)](#), [ar \(p. 815\)](#), and [ma \(p. 818\)](#).

sma	Special Expression
-----	------------------------------------

Seasonal moving average error specification.

`sma` can be included in a `ls` or `tsls` specification to specify a multiplicative seasonal moving average term. A `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 `ma` terms and that specified by the `sma` terms. The purpose of the `sma` expression is to allow you to form the product of lag polynomials.

Examples

```
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.

```
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 “z” option turns off backcasting.

Cross-references

See [“ARIMA Theory,” beginning on page 71](#) of the *User’s Guide II* for details on ARMA and seasonal ARMA modeling.

See also [sar \(p. 821\)](#), [ar \(p. 815\)](#), and [ma \(p. 818\)](#).

Chapter 6. String and Date Function Reference

EViews provides a full library of string and date functions for use with alphanumeric and date values. [Chapter 22. “Strings and Dates,” on page 695](#) of the *User’s Guide I* contains a discussion of the use of strings and dates in EViews, and provides a description of the string and date functions.

String Function Summary

String Functions

- [@datestr](#)converts a date number into a string ([p. 827](#)).
- [@dateval](#)converts a string into a date number ([p. 827](#)).
- [@dtoo](#)returns observation number corresponding to the date specified by a string ([p. 828](#)).
- [@eqna](#)tests for equality of string values treating empty strings as ordinary blank values ([p. 828](#)).
- [@insert](#)inserts string into another string ([p. 829](#)).
- [@instr](#)finds the starting position of the target string in a string ([p. 829](#)).
- [@isempty](#)tests string against empty string ([p. 830](#)).
- [@left](#)returns the leftmost characters string ([p. 830](#)).
- [@len, @length](#)finds the length of a string ([p. 830](#)).
- [@lower](#)returns the lowercase representation of a string ([p. 831](#)).
- [@ltrim](#)returns the string with spaces trimmed from the left end ([p. 831](#)).
- [@mid](#)returns a substring from a string ([p. 832](#)).
- [@neqna](#)tests for inequality of string values treating empty strings as ordinary blank values ([p. 833](#)).
- [@otod](#)returns a string associated with a the date or observation value ([p. 833](#)).
- [@replace](#)replaces substring in a string ([p. 834](#)).
- [@right](#)returns the rightmost characters of a string ([p. 834](#)).
- [@rtrim](#)returns the string with spaces trimmed from the right end ([p. 835](#)).
- [@str](#)returns a string representing the given number ([p. 835](#)).
- [@strdate](#)returns string corresponding to each element in workfile ([p. 835](#)).
- [@strlen](#)finds the length of a string ([p. 836](#)).
- [@strnow](#)returns a string representation of the current date ([p. 836](#)).
- [@trim](#)returns the string with spaces trimmed from the both ends ([p. 836](#)).
- [@upper](#)returns the uppercase representation of a string ([p. 837](#)).
- [@val](#)returns the number that a string represents ([p. 837](#)).

Date Function Summary

Date Functions

- [@dateadd](#) add to a date (p. 824).
- [@datediff](#) computes difference between dates (p. 825).
- [@datefloor](#) rounds date down to start of period(p. 825).
- [@datepart](#) extracts part of date (p. 826).
- [@datestr](#) converts a date number into a string (p. 827).
- [@dateval](#) converts a string into a date number (p. 827).
- [@dtoo](#) returns observation number corresponding to the date specified by a string (p. 828).
- [@makedate](#) converts numeric values into a date number (p. 831).
- [@now](#) returns the date number associated with the current time (p. 833).
- [@otod](#) returns a string associated with a the date or observation value (p. 833).
- [@strdate](#) returns string corresponding to each element in workfile (p. 835).
- [@strnow](#) returns a string representation of the current date (p. 836).

Date and String Entries

The following is an alphabetical listing of the functions used when working with strings and dates in EViews.

@dateadd	Date Function
----------	---------------

- Syntax: `@dateadd(d, offset[, u])`
- Argument 1: date number, *d*
- Argument 2: number of time units, *offset*
- Argument 3: time unit, *u*
- Return: date number

Returns the date number given by *d* offset by *offset* time units as specified by the time unit string *u*. If no time unit is specified, EViews will use the workfile regular frequency, if available.

Example:

Suppose that the value of *d* is 730088.0 (midnight, December 1, 1999). Then we can add and subtract 10 days from the date by using the functions

```
@dateadd(730088.0, 10, "dd")
@dateadd(730088.0, -10, "dd")
```

which return 730098.0 (December 11, 1999) and (730078.0) (November 21, 1999). Note that these results could have been obtained by taking the original numeric value plus or minus 10.

To add 5 weeks to the existing date, simply specify “W” or “WW” as the time unit string:

```
@dateadd(730088.0, 5, "ww")
```

returns 730123.0 (January 5, 2000).

@datediff	Date Function
------------------	---------------

Syntax: `@datediff(d1, d2[, u])`

Argument 1: date number, *d1*

Argument 2: date number, *d2*

Argument 3: time unit, *u*

Return: date number

Returns the difference between two date numbers *d1* and *d2*, measured by time units specified by the time unit string *u*. If no time unit is specified, EViews will use the workfile regular frequency, if available.

Example:

Suppose that *date1* is 730088.0 (December 1, 1999) and *date2* is 729754.0 (January 1, 1999), then,

```
@datediff(730088.0, 729754.0, "dd")
```

returns 334 for the number of days between the two dates. Note that this result is simply the difference between the two numbers.

The following expressions calculate differences in months and weeks:

```
@datediff(730088.0, 729754.0, "mm")
```

```
@datediff(730088.0, 729754.0, "ww")
```

return 11 and 47 for the number of months and weeks between the dates.

@datefloor	Date Function
-------------------	---------------

Syntax: `@datefloor(d1, u[, step])`

Argument 1: date number, *d1*

Argument 2: time unit, *u*

Argument 3: step, *step*

Return: date number

Finds the first possible date number defined by *d1* in the regular frequency defined by the given time unit *u* and optional step *step*. The frequency is defined relative to the basedate of Midnight, January 2001.

If *step* is omitted, the frequency will use a step of 1 so that by default, @DATEFLOOR will find the beginning of the period defined by the time unit.

Example:

Suppose that *date1* is 730110.8 (7 PM, December 23, 1999). Then the @DATEFLOOR values

```
@datefloor(730110.8, "dd")
@datefloor(730110.8, "mm")
```

yield 730110.0 (midnight, December 23, 1999) and 730088.0 (midnight, December 1, 1999), the date numbers for the begining of the day and month associated with 730110.8, respectively.

```
@datefloor(730110.8, "hh", 6)
@datefloor(730110.8, "hh", 12)
```

return 730110.75 (6 PM, December 23, 1999), and 730110.5 (Noon, December 23, 1999). which are the earliest date numbers for 6 and 12 hour regular frequencies anchored at Midnight, January 1, 2001.

@datepart	Date Function
-----------	---------------

Syntax:	@datepart(<i>d1</i> , <i>u</i>)
Argument 1:	date number, <i>d1</i>
Argument 2:	time unit, <i>u</i>
Return:	number

Returns a numeric part of a date number given by *u*, where *u* is a time unit string.

Example:

Consider the *d1* date value 730110.5 (noon, December 23, 1999). The @DATEPART values for

```
@datepart(730110.5, "dd")
@datepart(730110.5, "w")
@datepart(730110.5, "ww")
@datepart(730110.5, "mm")
@datepart(730110.5, "yy")
```

are 23 (day of the month), 4 (week in the month), 52 (week in the year), 12 (month in the year), and 1999 (year), respectively).

@datestr	Date Function 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 704](#) of the *User's Guide I* for additional details on date numbers and date format strings. See also [@strdate](#) (p. 835).

@dateval	Date Function String Function
----------	---

Syntax: `@dateval(str1[, fmt])`
Argument 1: string, *str1*
Argument 2: date format string, *fmt*
Return: date number

Convert the string representation of a date, *str*, into a date number using the optional format string *fmt*.

Example:

```
@dateval("12/1/1999", "mm/dd/yyyy")
```

will return the date number for December 1, 1999 (730088) while

```
@dateval("12/1/1999", "dd/mm/yyyy")
```

will return the date number for January 12, 1999 (729765).

See “[Dates](#)” on page 704 of the *User’s Guide I* for additional details on date numbers and date format strings.

@dtoo	Date Function String Function
--------------	---

Syntax: @dtoo(*str*)
Argument: string, *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 [stomna](#) (p. 864)).

Examples:

```
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. 833).

@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. 830) and [@neqna](#) (p. 833).

@insert	String Function
----------------	---------------------------------

Syntax: `@insert(str1, str2, n)`
 Argument 1: string, *str1*
 Argument 2: string, *str2*
 Argument 3: integer, *n*
 Return: string

Inserts the string *str2* into the base string *str1* at the position given by the integer *n*.

Example:

```
@insert("I believe it can be done", "not ", 16)
```

returns “I believe it cannot be done”.

See also [@replace](#) (p. 834), [@instr](#) (p. 829) and [@mid](#) (p. 832).

@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. 832).

@isempty	String Function
-----------------	---------------------------------

Syntax: [@isempty\(str\)](#)

Argument: string, *str*

Return: integer

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. 830).

@left	String Function
--------------	---------------------------------

Syntax: [@left\(str, n\)](#)

Argument 1: string, *str*

Argument 2: integer, *n*

Return: string

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:

```
scalar scl = @left("I did not do it",5)
```

returns "I did".

See also [@right](#) (p. 834), and [@mid](#) (p. 832).

@len, @length	String Function
----------------------	---------------------------------

Syntax: [@len\(str\)](#), [@length\(str\)](#)

Argument: string, *str*

Return: integer

Returns an integer value for the length of the string *str*.

Example:

```
@length("I did not do it")
```

returns the value 15.

See also [@mid](#) (p. 832).

@lower	String Function
---------------	---------------------------------

Syntax: [@lower](#)(*str*)

Argument: string, *str*

Return: string

Returns the lowercase representation of the string *str*.

Example:

```
@length("I did NOT do it")
```

returns the string “i did not do it”.

See also [@upper](#) (p. 837).

@ltrim	String Function
---------------	---------------------------------

Syntax: [@ltrim](#)(*str*)

Argument: string, *str*

Return: string

Returns the string *str* with spaces trimmed from the left.

Example:

```
@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. 835) and [@trim](#) (p. 836).

@makedate	Date Function
------------------	-------------------------------

Syntax: [@makedate](#)(*arg1*[, *arg2*[,*arg3*]], *fmt*)

Argument 1: number, *arg1*

Argument 2: number(s) *arg2*, *arg3*

Argument 3: date format, *fmt*

Return: date number

Takes the numeric values given by the arguments *arg1*, and optionally, *arg2*, etc. and returns a date number using the required format string, *fmt*.

Example:

The expressions,

```
@makedate(1999, "YYYY")
@makedate(99, "yy")
```

both return the date number 729754.0 corresponding to 12 midnight on January 1, 1999.

```
@makedate(199003, "YYYYmm")
@makedate(1990.3, "yyyymm")
@makedate(1031990, "ddmmyyyy")
@makedate(30190, "mddy")
```

all return the value 726526.0, representing March 1, 1990.

See [“Translating Ordinary Numbers into Date Numbers” on page 716](#) of the *User’s Guide I* for additional details.

@mid	String Function
------	---------------------------------

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 \(p. 830\)](#) and [@right \(p. 834\)](#).

@neqna	String Function
---------------	---------------------------------

Syntax: `@neqna(str1, str2)`

Argument 1: string, *str1*

Argument 2: string, *str2*

Return: integer

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:

```
@neqna("abc", "abc")
```

returns a 0, while

```
@eqna("", "def")
```

returns a 1.

See also [@isempty](#) (p. 830) and [@eqna](#) (p. 828).

@now	Date Function
-------------	-------------------------------

Syntax: `@now`

Return: date number

Returns the date number associated with the current time.

@otod	Date Function String Function
--------------	---

Syntax: `@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 `@otod(1)` returns the date or observation label for the start of the workfile.

See also [@dtoo](#) (p. 828).

@replace	String Function
-----------------	-----------------

Syntax: @replace(*str1*, *str2*, *str3*[,*n*])
Argument 1: string, *str1*
Argument 2: string, *str2*
Argument 3: string, *str3*
Argument 4: integer, *n*
Return: string

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:

```
@replace("Do you think that you can do it?", "you", "I")
```

returns the string “Do I think that I can do it?”, while

```
@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. 829) and [@mid](#) (p. 832).

@right	String Function
---------------	-----------------

Syntax: @right(*str*, *n*)
Argument 1: string, *str*
Argument 2: integer, *n*
Return: same as source

Returns a string containing *n* characters from the right end of *str*. If the source is shorter than *n*, the entire string is returned. Example:

```
%1 = @right("I doubt it", 8)
```

returns the string “doubt it”.

See also [@left](#) (p. 830), [@mid](#) (p. 832).

@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. 831) and [@trim](#) (p. 836).

@str	String Function
------	-----------------

Syntax: `@str(d,[fmt])`

Argument 1: scalar, *d*

Argument 2: numeric format string, *fmt*

Return: string

Returns a string representing the given number. You may provide an optional format string.

Example:

```
%1 = @str(15.23456)
```

```
alpha newa = @str(32.56)
```

See also [@val](#) (p. 837).

@strdate	Date Function String Function
----------	---------------------------------

Syntax: `@strdate(fmt)`

Argument: date format string, *fmt*

Return: string corresponding to each element in workfile

Return the set of workfile row dates as strings, using the date format string *fmt*.

See also [@datestr](#) (p. 827) and [@strnow](#) (p. 836).

@strlen	String Function
----------------	-----------------

Syntax: @strlen(*s*)

Argument: string, *s*

Return: number *n*

Same as [@len](#), [@length](#) (p. 830).

@strnow	Date Function String Function
----------------	---------------------------------

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:

```
@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. 835) and [@datestr](#) (p. 827).

@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:

```
@trim(" I doubt that I did it. ")
```

returns the string "I doubt that I did it."

See also [@ltrim](#) (p. 831) and [@rtrim](#) (p. 835).

@upper	String Function
--------	-----------------

Syntax: @upper(*str*)

Argument: string, *str*

Return: string

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. 831).

@val	String Function
------	-----------------

Syntax: @val(*str*[, *fmt*])

Argument 1: string, *str*

Argument 2: numeric format string, *fmt*

Return: scalar

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(tab1(3,4))
```

```
scalar sc3 = @val("-234.35")
```

See also [@str](#) (p. 835).

Chapter 7. Matrix Language Reference

The following entries constitute a listing of the functions and commands used in the EViews matrix language. For a description of the EViews matrix language, see [Chapter 18. “Matrix Language,”](#) on page 627 of the *User’s Guide I*.

Matrix Command and Function Summary

Utility Commands and Functions

colplacePlaces column vector into matrix (p. 844).
@columnextractExtracts column from matrix (p. 845).
@columnsNumber of columns in matrix object (p. 845).
@convertConverts series or group to a vector or matrix after removing NAs (p. 846).
@explodeCreates square matrix from a sym matrix object (p. 851).
@filledmatrixCreates matrix filled with scalar value (p. 851).
@filledrowvectorCreates rowvector filled with scalar value (p. 851).
@filledsymCreates sym filled with scalar value (p. 852).
@filledvectorCreates vector filled with scalar value (p. 852).
@getmaindiagonalExtracts main diagonal from matrix (p. 852).
@identityCreates identity matrix (p. 853).
@implodeCreates sym from lower triangle of square matrix (p. 853).
@kroneckerComputes the Kronecker product of two matrix objects (p. 855).
@makediagonalCreates a square matrix with ones down a specified diagonal and zeros elsewhere (p. 855).
matplacePlaces matrix object in another matrix object (p. 856).
mtosConverts a matrix object to series or group (p. 857).
@onesCreates a matrix, sym or vector of ones (p. 858).
@permutePermutes the rows of the matrix (p. 859).
@resampleRandomly draws from the rows of the matrix (p. 860).
@rowextractExtracts rowvector from matrix object (p. 861).
rowplacePlaces a rowvector in matrix object (p. 861).
@rowsReturns the number of rows in matrix object (p. 861).
@scaleScale the rows or columns of a matrix, or the rows and columns of a sym matrix (p. 862).
@sortSorts a matrix or vector (p. 863).

- stom** Converts series or group to vector or matrix after removing observations with NAs (p. 864).
- stomna** Converts series or group to vector or matrix without removing observations with NAs (p. 864).
- @subextract** Extracts submatrix from matrix object (p. 865).
- @transpose** Transposes matrix object (p. 866).
- @unitvector** Extracts column from an identity matrix (p. 867).
- @vec** Stacks columns of a matrix object (p. 867).
- @vech** Stacks the lower triangular portion of matrix by column (p. 867).

Matrix Algebra Functions

- @cholesky** Computes Cholesky factorization (p. 842).
- @cond** Calculates the condition number of a square matrix or sym (p. 845).
- @det** Calculate the determinant of a square matrix or sym (p. 848).
- @eigenvalues** Returns a vector containing the eigenvalues of a sym (p. 849).
- @eigenvectors** Returns a square matrix whose columns contain the eigenvectors of a matrix (p. 849).
- @inner** Computes the inner product of two vectors or series, or the inner product of a matrix object (p. 853).
- @inverse** Returns the inverse of a square matrix object or sym (p. 854).
- @issingular** Returns 1 if the square matrix or sym is singular, and 0 otherwise (p. 855).
- @norm** Computes the norm of a matrix object or series (p. 858).
- @outer** Computes the outer product of two vectors or series, or the outer product of a matrix object (p. 858).
- @pinverse** Returns the Moore-Penrose pseudo-inverse of a square matrix object or sym (p. 859).
- @rank** Returns the rank of a matrix object (p. 859).
- @solvesystem** Solves system of linear equations, $Mx = v$, for x (p. 863).
- @svd** Performs singular value decomposition (p. 865).
- @trace** Computes the trace of a square matrix or sym (p. 866).

Matrix Element Functions

- @ediv** Computes element by element division of two matrices (p. 849).
- @einv** Computes element by element inversion of a matrix (p. 850).
- @emult** Computes element by element multiplication of two matrices (p. 850).
- @epow** Raises each element in a matrix to a power (p. 850).

Matrix Descriptive Statistics Functions

You may use any of the descriptive statistics functions that are described in “[Descriptive Statistics](#)” on page 738 of the *Command Reference*. These statistics will be applied to the matrix object as a whole so that, for example, using `@mean` computes the mean taken over all non NA values in the matrix.

In addition, the following functions have special forms that apply to matrix objects:

- `@cor`..... Computes correlation between two vectors, or between the columns of a matrix (p. 847).
- `@cov` Computes covariance between two vectors, or between the columns of a matrix using n as the divisor (p. 847).
- `@covp` Computes covariance between two vectors, or between the columns of a matrix using n as the divisor (p. 847).
- `@covs`..... Computes covariance between two vectors, or between the columns of a matrix using $n - 1$ as the divisor (p. 847).
- `@inner` Computes the inner product of two vectors or series, or the inner product of a matrix object (p. 853).

There are also a handful of column functions that return results computed for each column of the matrix:

- `@cimax` Returns the index of the maximal value in each column of a matrix (p. 842).
- `@cimin`..... Returns the index of the minimal value in each column of a matrix (p. 843).
- `@cmax` Returns the maximal value in each column of a matrix (p. 843).
- `@cmean` Returns the mean value in each column of a matrix (p. 843).
- `@cmin`..... Returns the minimal value for each column of the matrix (p. 844).
- `@cnas`..... Returns the number of NA values in each column of a matrix (p. 844).
- `@cobs`..... Returns the number of non-NA values in each column of a matrix (p. 844).
- `@csum` Returns the sum of each column of a matrix (p. 848).

Additional Functions

In addition, EViews also supports matrix element versions of the following categories of functions:

Category	Matrix Element Support
Operators (p. 734)	addition, subtraction, multiplication, and comparison operators (note that the comparison operators perform the comparison for every element of the matrix object and return a 1 if true for all elements, and a 0 if the comparison fails for any element).
Basic Mathematical Functions (p. 735)	All, except for “@inv” and “@recode”.
Special Functions (p. 751)	All
Trigonometric Functions (p. 754)	All
Statistical Distribution Functions (p. 754)	All

Matrix Language Entries

The following section provides an alphabetical listing of the commands and functions associated with the EViews matrix language. Each entry outlines the basic syntax and provides examples and cross references.

@cholesky	Matrix Algebra Function
------------------	---

Syntax: @cholesky(*s*)
Argument: sym, *s*
Return: matrix

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:

```
matrix fact = @cholesky(s1)
matrix orig = fact*@transpose(fact)
```

@cimax	Matrix Algebra Function
---------------	---

Syntax: @cimax(*m*)
Argument: matrix, *m*
Return: vector

Returns a vector containing the index (*i.e.*, row number) of the maximal values of each column of the matrix m . Example:

```
vector v1 = @cimax(m1)
```

See also [@cimin](#) (p. 843), [@cmax](#) (p. 843), [@cmin](#) (p. 844).

@cimin	Matrix Column Function
---------------	--

Syntax: [@cimin](#)(m)

Argument: matrix, m

Return: vector

Returns a vector containing the index (*i.e.*, row number) of the minimal values of each column of the matrix m . Example:

```
vector v1 = @cimin(m1)
```

See also [@cimax](#) (p. 842), [@cmax](#) (p. 843), [@cmin](#) (p. 844).

@cmax	Matrix Column Function
--------------	--

Syntax: [@cmax](#)(m)

Argument: matrix, m

Return: vector

Returns a vector containing the maximal values of each column of the matrix m . Example:

```
vector v1 = @cmax(m1)
```

See also [@cimax](#) (p. 842), [@cimin](#) (p. 843), [@cmin](#) (p. 844).

@cmean	Matrix Column Function
---------------	--

Syntax: [@cmean](#)(m)

Argument: matrix, m

Return: vector

Returns a vector containing the mean values of each column of the matrix m . Example:

```
vector v1 = @cmean(m1)
```

See also [@csum](#) (p. 848).

@cmin	Matrix Column Function
--------------	--

Syntax: `@cmin(m)`

Argument: matrix, *m*

Return: vector

Returns a vector containing the minimal values of each column of the matrix *m*. Example:

```
vector v1 = @cmin(m1)
```

See also [@cimax](#) (p. 842), [@cimin](#) (p. 843), [@cmax](#) (p. 843).

@cnas	Matrix Column Function
--------------	--

Syntax: `@cnas(m)`

Argument: matrix, *m*

Return: vector

Returns a vector containing the number of missing values in each column of the matrix *m*.

Example:

```
vector v1= @cnas(m1)
```

@cobs	Matrix Column Function
--------------	--

Syntax: `@cobs(m)`

Argument: matrix, *m*

Return: vector

Returns a vector containing the number of non-missing values in each column of the matrix *m*. Example:

```
vector v1= @cobs(m1)
```

colplace	Matrix Utility Command
-----------------	--

Syntax: `colplace(m, v, n)`

Argument 1: matrix, *m*

Argument 2: vector, *v*

Argument 3: integer, *n*

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:


```
colplace(m1,v1,3)
```

The third column of M1 will be set equal to the vector V1.

See also [rowplace](#) (p. 861).

@columnextract	Matrix Utility Function
-----------------------	---

Syntax: `@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:

```
vector v1 = @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. 861).

@columns	Matrix Utility Function
-----------------	---

Syntax: `@columns(o)`

Argument: matrix, vector, rowvector, sym, scalar, or series, *o*

Return: integer

Returns the number of columns in the matrix object *o*. Example:

```
scalar sc2 = @columns(m1)
```

```
vec1(2) = @columns(s1)
```

See also [@rows](#) (p. 861).

@cond	Matrix Algebra Function
--------------	---

Syntax: `@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:

```
scalar s1 = @cond(m1)
mat1(1,4) = @cond(s1,2)
```

@convert	Matrix Utility Function
----------	-------------------------

- Syntax: @convert(*o*, *smp*)
- Argument 1: series or group, *o*
- Argument 2: (optional) sample, *smp*
- Return: vector or matrix

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:

```
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:

```
matrix m1 = @convert(grp1)
matrix m2 = @convert(grp1, smp1)
```

For a conversion method that preserves NAs, see [stomna](#) (p. 864).

@cor	Matrix Descriptive Statistic
-------------	------------------------------

Syntax: `@cor(v1, v2)`
Argument 1: vector, rowvector, or series, *v1*
Argument 2: vector, rowvector, or series, *v2*
Return: scalar

Syntax: `@cor(o)`
Argument: matrix object or group, *o*
Return: sym

If used with two vector or series objects, *v1* and *v2*, `@cor` returns the correlation between the two vectors or series. Examples:

```
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.

```
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. 847).

@cov	Matrix Descriptive Statistic
-------------	------------------------------

Syntax: `@cov(v1, v2)`
Argument 1: vector, rowvector, or series, *v1*
Argument 2: vector, rowvector, or series, *v2*
Return: scalar

Syntax: `@cov(o)`
Argument: matrix object or group, *o*
Return: sym

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. 847).

@covp	Matrix Descriptive Statistic
--------------	--

Compute covariances using n as the divisor in the moment calculation. Same as [@cov](#) (p. 847). See also [@covs](#) (p. 848)

@covs	Matrix Descriptive Statistic
--------------	--

Compute covariances using $n - 1$ as the divisor in the moment calculation. See also [@cov](#) (p. 847) or [@covp](#) (p. 848).

@csum	Matrix Descriptive Statistic
--------------	--

Syntax: `@csum(m)`
Argument: `matrix, m`
Return: `vector`

Returns a vector containing the summation of the rows in each column of the matrix *m*.
Example:

```
vector v1= @csum(m1)
```

See also [@cmean](#) (p. 843).

@det	Matrix Algebra Function
-------------	---

Syntax: `@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:

```
scalar s1 = @det(m1)
vec4(2) = @det(s2)
```

See also [@rank](#) (p. 859).

@ediv	Matrix Element Function
--------------	---

Syntax: `@ediv(m1,m2)`

Argument 1: `matrix,m1`

Argument 2: `matrix,m2`

Return: `matrix`

Returns the element by element division of two matrix objects or syms. Each element of the returned matrix is equal to the corresponding element in *m1* divided by the corresponding element in *m2*. Note *m1* and *m2* must be of identical dimensions. Example:

```
matrix m3 = @ediv(m1,m2)
```

See also [@einv](#) (p. 850), [@emult](#) (p. 850), and [@epow](#) (p. 850).

@eigenvalues	Matrix Algebra Function
---------------------	---

Syntax: `@eigenvalues(s)`

Argument: `sym, s`

Return: `vector`

Returns a vector containing the eigenvalues of the sym. The eigenvalues are those scalars λ 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. 849)). 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 $Sv = nv$, 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. 849).

@einv	Matrix Element Function
--------------	---

Syntax: `@einv(m)`

Argument: `matrix, m`

Return: `matrix`

Returns the element by element inverse of a matrix object or sym. Each element of the returned matrix is equal to 1 divided by the corresponding element of the input matrix.

Example:

```
matrix m2 = @einv(m1)
```

See also [@ediv](#) (p. 849), [@emult](#) (p. 850), and [@epow](#) (p. 850).

@emult	Matrix Element Function
---------------	---

Syntax: `@emult(m1, m2)`

Argument 1: `matrix, m1`

Argument 2: `matrix, m2`

Return: `matrix`

Returns the element by element multiplication of two matrix objects or syms. Each element of the returned matrix is equal to the corresponding element in *m1* multiplied by the corresponding element in *m2*. Note *m1* and *m2* must be of identical dimensions. Example:

```
matrix m3 = @emult(m1, m2)
```

See also [@ediv](#) (p. 849), [@einv](#) (p. 850), and [@epow](#) (p. 850).

@epow	Matrix Element Function
--------------	---

Syntax: `@emult(m1, n)`

Argument 1: `matrix, m1`

Argument 2: `matrix, m2, scalar or number, n`

Return: `matrix`

Returns a matrix where every element is equal to the corresponding element in *m1* raised to the power *n*. If the second argument is a matrix, the each element of *m1* will be raised to the power of the corresponding element of *m2*. Example:

```
matrix m2 = @epow(m1, 3)
```

See also [@ediv](#) (p. 849), [@einv](#) (p. 850), and [@emult](#) (p. 850).

@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:

```
matrix m2 = @explode(s1)
```

See also [@implode](#) (p. 853).

@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:

```
matrix m2 = @filledmatrix(3,2,7)
```

creates a 3×2 matrix where each element is set to 7. See also [Matrix::fill](#) (p. 256).

See [Coef::fill](#) (p. 18), [Rowvector::fill](#) (p. 339), [Series::fill](#) (p. 366), [Sym::fill](#) (p. 464), and [Vector::fill](#) (p. 580) for routines to perform general filling of matrix objects.

@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:

```
rowvector r1 = @filledrowvector(3,1)
```

creates a 3 element rowvector where each element is set to 1. See also

See [Coef::fill](#) (p. 18), [Matrix::fill](#) (p. 256), [Sym::fill](#) (p. 464), and [Vector::fill](#) (p. 580) for routines to perform general filling of matrix objects.

@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:

```
sym s2= @filledsym(3, 9)
```

creates a 3×3 sym where each element is set to 9. See also [Sym::fill](#) (p. 464).

See [Coef::fill](#) (p. 18), [Matrix::fill](#) (p. 256), [Rowvector::fill](#) (p. 339), and [Vector::fill](#) (p. 580) for routines to perform general filling of matrix objects.

@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:

```
vector r1 = @filledvector(5, 6)
```

creates a 5 element column vector where each element is set to 6. See also [Vector::fill](#) (p. 580).

See [Coef::fill](#) (p. 18), [Matrix::fill](#) (p. 256), [Rowvector::fill](#) (p. 339), and [Sym::fill](#) (p. 464) for routines to perform general filling of matrix objects.

@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:


```
vector v1 = @getmaindiagonal(m1)
vector v2 = @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. 851).

@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, *v1* and *v2*, `@inner` returns the dot or inner product of the two vectors. Examples:

```
scalar s1 = @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:

```
scalar s1 = @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:

```
sym s2 = @inner(gr1)
sym s3 = @inner(gr1, smp1)
```

See also [@outer](#) (p. 858).

@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. 863).

@issingular	Matrix Algebra Function
--------------------	---

Syntax: @issingular(*o*)
Argument: matrix or sym, *o*
Return: integer

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 s1 = @issingular(m1)
```

@kronecker	Matrix Algebra Function
-------------------	---

Syntax: @kronecker(*o1*, *o2*)
Argument 1: matrix object, *o1*
Argument 2: matrix object, *o2*
Return: matrix

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:

```
matrix m3 = @kronecker(m1,m2)
```

@makediagonal	Matrix Utility Function
----------------------	---

Syntax: @makediagonal(*v*, *k*)
Argument 1: vector or rowvector, *v*
Argument 2: (optional) integer, *k*
Return: sym or 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 sym 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*.

Examples:

```
sym s1 = @makediagonal(v1)
```

```
matrix m2 = @makediagonal(v1,1)
matrix m4 = @makediagonal(r1,-3)
```

S1 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:

```
matrix(10,10) m1
vector(9) rho = .3
m1 = @makediagonal(rho,-1) + @makediagonal(rho,+1)
m1 = m1 + @identity(10)
```

Note that to make a diagonal matrix with the same elements on the diagonal, you may use `@identity`, multiplied by the scalar value.

See also [@identity](#) (p. 853).

matplace	Matrix Utility Command
-----------------	--

Syntax:	<code>matplace(<i>m1</i>, <i>m2</i>, <i>n1</i>, <i>n2</i>)</code>
Argument 1:	matrix, <i>m1</i>
Argument 2:	matrix, <i>m2</i>
Argument 3:	integer, <i>n1</i>
Argument 4:	integer, <i>n2</i>

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:

```
matrix(100,5) m1
matrix(100,2) m2
matplace(m1,m2,1,1)
```

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. Example:

```
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.

```
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 18. “Matrix Language,” on page 627](#) of the *User’s Guide I* for further discussions and examples of matrices.

See also [stom](#) (p. 864) and [stomna](#) (p. 864).

@norm	Matrix Algebra Function
--------------	---

Syntax: @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 “-1” for the infinity norm, “0” for the Frobenius norm, and an integer “*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)
```

@ones	Matrix Algebra Function
--------------	---

Syntax: @ones(*n1*,*n2*)
Argument 1: integer, *n1*
Argument 2: integer,*n2*
Return: vector, matrix or sym

Creates a vector, matrix or sym filled with the value 1. The size of the created matrix is given by the integers *n1* (number of rows) and *n2* (number of columns). Example:

```
matrix m1=@ones(3,2)
vector v1=@ones(18)
```

@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:

```
matrix m1=@outer(v1,v2)
```

```
matrix m4=@outer(r1,r2)
```

See also [@inner](#) (p. 853).

@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.

```
matrix xp = @permute(x)
```

To draw with replacement from rows of a matrix, use [@resample](#) (p. 860).

@pinverse	Matrix Algebra Function
------------------	---

Syntax: `@inverse(m)`

Argument: matrix or sym, *m*

Return: matrix or sym

Returns the Moore-Penrose pseudo-inverse of a matrix object or sym. The pseudoinverse has the property that both pre-multiplying and post-multiplying the pseudo-inverse by the source matrix returns the source matrix, and that both pre-multiplying and post-multiplying the source matrix by the pseudo-inverse will return the inverse. The pseudo-inverse of a matrix returns a matrix, while the pseudo-inverse of a sym returns a sym. Note that pseudo-inverting a sym is much faster than inverting a matrix.

Examples:

```
matrix m2 = @pinverse(m1)
sym s2 = @pinverse(s1)
sym s3 = @pinverse(@implode(m2))
```

@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).

```
scalar rank1 = @rank(m1)
scalar rank2 = @rank(s1)
```

See also [@svd](#) (p. 865).

@resample	Matrix Utility Function
-----------	-------------------------

Syntax:	@resample(<i>m1</i> , <i>n2</i> , <i>n3</i> , <i>v4</i>)
Input 1:	matrix <i>m1</i>
Input 2:	(optional) integer <i>n2</i>
Input 3:	(optional) positive integer <i>n3</i>
Input 4:	(optional) vector <i>v4</i>
Output:	matrix

This function returns a matrix whose rows are randomly drawn with replacement from rows of the input matrix.

n2 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* + *n2* rows and *c* columns. By default, *n2*=0 .

n3 represents the block size for the resample procedure. If you specify *n3* > 1 , then blocks of consecutive rows of length *n3* will be drawn with replacement from the first *r* − *n3* + 1 rows of the input matrix.

You may provide a name for the vector *v4* 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).

```
matrix xb = @bootstrap(x)
```

To draw without replacement from rows of a matrix, use [@permute](#) (p. 859).

@rowextract	Matrix Utility Function
--------------------	---

Syntax: `@rowextract(m, n)`
Argument 1: matrix or sym, *m*
Argument 2: integer, *n*
Return: rowvector

Extracts a rowvector from row *n* of the matrix object *m*. Example:

```
rowvector r1 = @rowextract(m1,3)
```

See also [@columnextract](#) (p. 845).

rowplace	Matrix Utility Command
-----------------	--

Syntax: `rowplace(m, r, n)`
Argument 1: matrix, *m*
Argument 2: rowvector, *r*
Argument 3: integer

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:

```
rowplace(m1,r1,4)
```

See also [colplace](#) (p. 844).

@rows	Matrix Utility Function
--------------	---

Syntax: `@rows(o)`
Argument: matrix, vector, rowvector, sym, series, or group, *o*
Return: scalar

Returns the number of rows in the matrix object, *o*.

Example:

```
scalar scl=@rows(m1)
scalar size=@rows(m1)*@columns(m1)
```

For series and groups [@rows](#) (p. 861) returns the number of observations in the workfile range. See also [@columns](#) (p. 845).

@scale	Matrix Utility Function
---------------	-------------------------

Syntax: `@scale(m, v[,p])`
Argument 1: matrix or sym, *m*
Argument 2: vector or rowvector, *v*
Argument 3: (optional) number, *p*
Return: matrix or sym

Scale the rows *or* columns of a matrix, or the rows *and* columns of a sym matrix.

- If *m* is a matrix and *v* is a vector, the *i*-th row of *m* will be scaled by the *i*-th element of *v*, optionally raised to the power *p* (row scaling). The length *v* must equal the number of rows of *m*.
- If *m* is a matrix and *v* is a rowvector, the *i*-th column of *m* will be scaled by the *i*-th element of *v*, optionally raised to the power *p* (column scaling). The length *v* must equal the number of columns of *m*.
- If *m* is a sym object, then *v* may either be a vector or a rowvector. The (*i*,*j*)-th element of *m* will be scaled by both the *i*-th and *j*-th elements of *v* (row and column scaling). The length *v* must equal the number of rows (and columns) of *m*.

Let *M* be the matrix object, *V* be the vector or rowvector, and $\Lambda = \text{diag}(V_1, V_2, \dots, V_p)$ be the diagonal matrix formed from the elements of *V*. Then `@scale(m, v, p)` returns:

- $\Lambda^p M$, if *M* is a matrix and *V* is a vector.
- $M \Lambda^p$, if *M* is a matrix and *V* is a rowvector.
- $\Lambda^p M \Lambda^p$, if *M* is a sym matrix.

Example:

```
sym covmat = @cov(grp1)
vector vars = @getmaindiagonal(covmat)
sym corrmatrix = @scale(covmat, vars, -0.5)
```

computes the covariance matrix for the series in the group object GRP1, extracts the variances to a vector VARS, then uses VARS to obtain a correlation matrix from the covariance matrix.

```
matrix covmat1 = covmat
matrix rowscaled = @scale(covmat, vars, -0.5)
matrix colscaled = @scale(covmat, @transpose(vars), -0.5)
sym corrmatrix1 = colscaled
```

performs the same scaling in multiple steps. Note that the COLSCALED matrix is square and symmetric, but not a sym object.

@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. 854).

@sort	Matrix Algebra Function
--------------	---

Syntax: `@sort(v, o, t)`
 Argument 1: vector
 Argument 2: (optional) option, *o*
 Argument 3: (optional) option, *t*
 Return: vector

Returns the sorted elements of the matrix or vector object o . The order of sorting is set using o : “a” (ascending) or “d” (descending). Ties are broken according to the setting of t : “i” (ignore), “f” (first), “l” (last), “a” (average), “r” randomize. The defaults are “a”, and “i”, respectively.

Note that sorting a matrix sorts every element of the matrix and arranges them by column, with the first element in the first row of the first column, and the last element in the last row of the last column.

Example:

```
vector rank1 = @sort(v1,"a","f")
```

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:

```
stom(ser1, v1)
stom(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:

```
stom(grp1, m1)
stom(grp1, m2, smp1)
```

For a conversion method that preserves NAs, see [stomna](#) (p. 864).

stomna	Matrix Utility Command
--------	--

Syntax: `stomna(o1, o2, smp)`
Argument 1: series or group, *o1*
Argument 2: vector or matrix, *o2*
Argument 3: (optional) sample *smp*

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:

```
stom(ser1, v1)
stom(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:

```
stomna(grp1,m1)
stomna(grp1,m2,smp1)
```

For conversion methods that automatically remove observations with NAs, see [@convert](#) (p. 846) and [stom](#) (p. 864).

@subextract	Matrix Utility Function
--------------------	-------------------------

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:

```
matrix m2 = @subextract(m1,5,9,6,11)
matrix m2 = @subextract(m1,5,9)
```

@svd	Matrix Algebra Function
-------------	-------------------------

Syntax: `@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:

$$\begin{aligned} m1 &= UWV' \\ U'U &= V'V = I \end{aligned} \tag{7.1}$$

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 [@rank](#) (p. 859) function for a related discussion.

Examples:

```
matrix m2
vector v1
matrix m3 = @svd(m1,v1,m2)
```

@trace	Matrix Algebra Function
--------	---

Syntax: @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:

```
matrix m2 = @transpose(m1)
rowvector r2 = @transpose(v1)
```

@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 instead you wish to create a 10-element vector of ones, you should use a declaration statement of the form:

```
vector v1=@ones(10)
```

See also [@ones](#) (p. 858).

@vec	Matrix Utility Function
-------------	---

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	Matrix Utility Function
--------------	---

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 other. The vector has the same number of elements as the source matrix has in its lower triangle. Example:

```
vector v1 = @vech(m1)
```


Chapter 8. Programming Language Reference

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 17. “EViews Programming,”](#) on [page 593](#) of the *User’s Guide I*.

Programming Summary

Support Commands

[open](#) opens a program file from disk ([p. 874](#)).
[output](#) redirects print output to objects or files ([p. 740](#)).
[poff](#) turns off automatic printing in programs ([p. 875](#)).
[pon](#) turns on automatic printing in programs ([p. 875](#)).
[program](#) declares a program ([p. 763](#)).
[run](#) runs a program ([p. 876](#)).
[statusline](#) sends message to the status line ([p. 876](#)).
[tic](#) reset the timer ([p. 790](#)).
[toc](#) display elapsed time (since timer reset) in seconds ([p. 791](#)).

Program Statements

[call](#) calls a subroutine within a program ([p. 870](#)).
[else](#) denotes start of alternative clause for IF ([p. 871](#)).
[endif](#) marks end of conditional commands ([p. 871](#)).
[endsub](#) marks end of subroutine definition ([p. 871](#)).
[exitloop](#) exits from current loop ([p. 872](#)).
[for](#) start of FOR execution loop ([p. 873](#)).
[if](#) conditional execution statement ([p. 873](#)).
[include](#) include subroutine in programs ([p. 874](#)).
[next](#) end of FOR loop ([p. 874](#)).
[return](#) exit subroutine ([p. 876](#)).
[step](#) (optional) step size of a FOR loop ([p. 876](#)).
[stop](#) halts execution of program ([p. 877](#)).
[subroutine](#) declares subroutine ([p. 878](#)).
[then](#) part of IF statement ([p. 878](#)).
[to](#) upper limit of FOR loop ([p. 879](#)).
[wend](#) end of WHILE loop ([p. 880](#)).
[while](#) start of WHILE loop ([p. 881](#)).

Support Functions

- [@date](#) string containing the current date (p. 870).
- [@errorcount](#) number of errors encountered (p. 872).
- [@evpath](#) string containing the directory path for the EViews executable (p. 872).
- [@isobject](#) checks for existence of object (p. 874).
- [@tempopath](#) string containing the directory path for EViews temporary files (p. 878).
- [@time](#) string containing the current time (p. 879).
- [@toc](#) calculates elapsed time (since timer reset) in seconds (p. 880).

Programming Language Entries

The following section provides an alphabetical listing of the commands and functions associated with the EViews programming language. Each entry outlines the basic syntax and provides examples and cross references.

call	Program Statement
------	-----------------------------------

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 621 of the *User’s Guide I*. See also [subroutine](#) (p. 878), [endsub](#) (p. 871).

@date	Support Function
-------	----------------------------------

Syntax: [@date](#)
Return: string

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. 879).

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

```

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 610 of the *User’s Guide I*. See also, [if](#) (p. 873), [endif](#) (p. 871), [then](#) (p. 878).

endif	Program Statement
-------	-------------------

End of IF statement. Marks the end of an IF, or an IF-ELSE statement.

Syntax

```

if [condition] then
    [commands if condition true]
endif

```

Cross-references

See “IF Statements” on page 610 of the *User’s Guide I*. See also, [if](#) (p. 873), [else](#) (p. 871), [then](#) (p. 878).

endsub	Program Statement
--------	-------------------

Mark the end of a subroutine.

Syntax

```

subroutine name(arguments)
    commands
endsub

```

Cross-references

See “Defining Subroutines,” beginning on page 619 of the *User’s Guide I*. See also, [subroutine](#) (p. 878), [return](#) (p. 876).

@errorcount	Support Function
--------------------	----------------------------------

Syntax: **@errorcount**

Argument: none

Return: integer

Number of errors encountered. Returns a scalar containing the number of errors encountered during program execution.

@evpath	Support Function
----------------	----------------------------------

Syntax: **@evpath**

Return: string

Returns a string containing the directory path for the EViews executable.

Examples

If your currently executing copy of EViews is installed in “D:\EVIEWWS”, then

```
%y = @evpath
```

assigns a string of the form “D:\EVIEWWS”.

Cross-references

See also [cd](#) (p. 689) and [@temppath](#) (p. 878).

exitloop	Program Statement
-----------------	-----------------------------------

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 612 of the *User’s Guide I*. See also, [stop](#) (p. 877), [return](#) (p. 876), [for](#) (p. 873), [next](#) (p. 874), [step](#) (p. 876).

for	Program Statement
-----	-------------------

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 612 of the *User’s Guide I*. See also, [exitloop](#) (p. 872), [next](#) (p. 874), [step](#) (p. 876).

if	Program Statement
----	-------------------

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 610 of the *User’s Guide I*. See also [else](#) (p. 871), [endif](#) (p. 871), [then](#) (p. 878).

include	Program Statement
----------------	-----------------------------------

Include another file in a program.

The `include` statement is used to include the contents of another file in a program file.

Syntax

`include filename`

Cross-references

See “Multiple Program Files” on page 618 of the *User’s Guide I*. See also [call](#) (p. 870).

@isobject	Support Function
------------------	----------------------------------

Syntax: `@isobject(str)`

Argument: string, *str*

Return: integer

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.

next	Program Statement
-------------	-----------------------------------

End of FOR loop. `next` marks the end of a FOR loop in a program.

Syntax

`for [conditions of the FOR loop]`

`[commands]`

`next`

Cross-references

See “The FOR Loop,” beginning on page 612 of the *User’s Guide I*. See also, [exitloop](#) (p. 872), [for](#) (p. 873), [step](#) (p. 876).

open	Command
-------------	-------------------------

Open a file. Opens a workfile, database, program file, or ASCII text file.

See [open](#) (p. 737).

output	Command
---------------	-------------------------

Redirects printer output or display estimation output.

See [output](#) (p. 740).

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 771 of the *User’s Guide II* for a discussion of printer control.

See also [pon](#) (p. 875).

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. 740)). 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 771 of the *User’s Guide II* for a discussion of printer control.

See also [poff](#) (p. 875).

program	Command
----------------	-------------------------

Create a program.

See [program](#) (p. 763).

return	Program Statement
---------------	-----------------------------------

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

```
if [condition] then
    return
endif
```

Cross-references

See “Subroutines,” [beginning on page 619](#) of the *User’s Guide I*. See also [exitloop](#) (p. 872), [stop](#) (p. 877).

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.

See [run](#) (p. 772).

statusline	Command
-------------------	-------------------------

Send a text message to the EViews status line.

Syntax

```
statusline string
```

Example

```
for !i = 1 to 10
    statusline Iteration !i
next
```

step	Program Statement
-------------	-----------------------------------

Step size of a FOR loop.

Syntax

```
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

```

for !j=5 to 1 step -1
    series x = nrnd*!j
next

```

repeatedly executes the commands in the loop with the control variable !J set to “5”, “4”, “3”, “2”, “1”.

```

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:

```

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 612](#) of the *User’s Guide I*. See also [exitloop](#) (p. 872), [for](#) (p. 873), [next](#) (p. 874).

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. 872), [return](#) (p. 876).

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 619](#) of the *User’s Guide I*. See also [endsub](#) (p. 871).

@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:\EVIEWWS”, then:

```
%y = @temppath
```

assigns a string of the form “D:\EVIEWWS”.

Cross-references

See also [cd](#) (p. 689) and [@evpath](#) (p. 872).

then	Program Statement
-------------	-----------------------------------

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 610 of the *User’s Guide I*. See also, [else](#) (p. 871), [endif](#) (p. 871), [if](#) (p. 873).

@time	Support Function
--------------	----------------------------------

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. 870).

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 612 of the *User’s Guide I*.

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 612 of the *User’s Guide I*. See also, [exitloop](#) (p. 872), [for](#) (p. 873), [next](#) (p. 874).

@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. 790) and [toc](#) (p. 791).

wend	Program Statement
------	-----------------------------------

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 616 of the *User’s Guide I*. See also [while](#) (p. 881).

while	Program Statement
-------	-----------------------------------

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 616](#) of the *User’s Guide I*. See also [wend \(p. 880\)](#).

Appendix A. Operator and Function Listing

The following reference is an alphabetical listing of operators and functions which may be used in series assignment and generation, and in many cases, in matrix operations or element evaluation. Additional details on these operators and functions is provided in [Appendix A. “Operator and Function Reference,” beginning on page 733](#) of the *User’s Guide I*.

Operator / Function	Description
+	add
–	subtract
*	multiply
/	divide
^	raise to the power
<	less than
<=	less than or equal to
<>	not equal to
=	equal to
>	greater than
>=	greater than or equal to
@abs(x), abs(x)	absolute value
@acos(x)	arc cosine (real results in radians)
and	logical and
@asin(x)	arc sine (real results in radians)
@atan(x)	arc tangent (results in radians)
@beta(a,b)	beta integral
@betainc(x,a,b)	incomplete beta integral
@betaincder(x,a,b,s)	derivative of the incomplete beta integral
@betaincinv(p,a,b)	inverse of the incomplete beta integral
@betalog(a,b)	natural logarithm of the beta integral
@binom(n,x)	binomial coefficient
@binomlog(n,x)	natural logarithm of the binomial coefficient
@cbeta(x,a,b)	beta cumulative distribution (CDF)
@cbinom(x,n,p)	binomial cumulative distribution (CDF)
@cchisq(x,v)	chi-square cumulative distribution (CDF)

Operator / Function	Description
@ceiling(x)	smallest integer not less than
@cellid	element
@cexp(x,m)	exponential cumulative distribution (CDF)
@cextreme(x)	extreme value cumulative distribution (CDF)
@cfdist(x,v1,v2)	F -distribution cumulative distribution (CDF)
@cgamma(x,b,r)	gamma cumulative distribution (CDF)
@cgcd(x,r)	generalised error cumulative distribution (CDF)
@chisq(x,v)	chi-square p-value
@claplace(x)	Laplace cumulative distribution (CDF)
@clogistic(x)	logistic cumulative distribution (CDF)
@cloglog(x)	complementary log-log function
@clognorm(x,m,s)	log normal cumulative distribution (CDF)
@cnegbin(x,n,p)	negative binomial cumulative distribution (CDF)
@cnorm(x)	normal cumulative distribution (CDF)
@columns(g)	number of columns.
@cor(x,y[,s])	correlation
@cos(x)	cosine (argument in radians)
@cov(x,y[,s])	population covariance
@covp(x,y[,s])	population covariance
@covs(x,y[,s])	sample covariance
@cpareto(x,a,k)	Pareto cumulative distribution (CDF)
@cpoisson(x,m)	Poisson cumulative distribution (CDF)
@crossid	observations in range
@ctdist(x,v)	Student's t -distribution cumulative distribution (CDF)
@cumbmax(x[,s])	backwards cumulative maximum
@cumbmean(x[,s])	backwards cumulative mean
@cumbmin(x[,s])	backwards cumulative minimum
@cumbnas(x[,s])	backwards cumulative number of NA observations
@cumbobs(x[,s])	backwards cumulative number of non-NA observations
@cumbprod(x[,s])	backwards cumulative product
@cumbstdev(x[,s])	backwards cumulative sample standard deviation
@cumbstdevp(x[,s])	backwards cumulative population standard deviation
@cumbstdevs(x[,s])	backwards cumulative sample standard deviation
@cumbsum(x[,s])	backwards cumulative sum

Operator / Function	Description
@cumbsumsq(<i>x[,s]</i>)	backwards cumulative sum-of-squares
@cumbvar(<i>x[,s]</i>)	backwards cumulative population variance
@cumbvarp(<i>x[,s]</i>)	backwards cumulative population variance
@cumbvars(<i>x[,s]</i>)	backwards cumulative sample variance
@cummax(<i>x[,s]</i>)	cumulative maximum
@cummean(<i>x[,s]</i>)	cumulative mean
@cummin(<i>x[,s]</i>)	cumulative minimum
@cumnas(<i>x[,s]</i>)	cumulative number of NA observations
@cumobs(<i>x[,s]</i>)	cumulative number of non-NA observations
@cumprod(<i>x[,s]</i>)	cumulative product
@cumstdev(<i>x[,s]</i>)	cumulative sample standard deviation
@cumstdevp(<i>x[,s]</i>)	cumulative population standard deviation
@cumstdevs(<i>x[,s]</i>)	cumulative sample standard deviation
@cumsum(<i>x[,s]</i>)	cumulative sum
@cumsumsq(<i>x[,s]</i>)	cumulative sum-of-squares
@cumvar(<i>x[,s]</i>)	cumulative population variance
@cumvarp(<i>x[,s]</i>)	cumulative population variance
@cumvars(<i>x[,s]</i>)	cumulative sample variance
@cunif(<i>x,a,b</i>)	uniform cumulative distribution (CDF)
@cweib(<i>x,m,a</i>)	Weibull cumulative distribution (CDF)
d(<i>x</i>)	first difference
d(<i>x,n</i>)	<i>n</i> -th order difference
d(<i>x,n,s</i>)	<i>n</i> -th order difference with a seasonal difference at <i>s</i>
@date	element
@dateadd(<i>d, offset[,u]</i>)	calculates date number offsets
@datediff(<i>d1,d2[,u]</i>)	difference between two dates
@datefloor(<i>d1,u[,step]</i>)	calculates a date floor
@datepart(<i>d1,u</i>)	calculates the date part of a number
@datestr(<i>d[,fmt]</i>)	converts date to string
@dateval(<i>str1[,fmt]</i>)	converts string to date
@day	day of month
@dbeta(<i>x,a,b</i>)	beta density function
@dbinom(<i>x,n,p</i>)	binomial density function
@dchisq(<i>x,v</i>)	chi-square density function

Operator / Function	Description
@dexp(<i>x</i> , <i>m</i>)	exponential density function
@dextreme(<i>x</i>)	extreme value density function
@dfdistrib(<i>x</i> , <i>v1</i> , <i>v2</i>)	F -distribution density function
@dgamma(<i>x</i> , <i>b</i> , <i>r</i>)	gamma density function
@dged(<i>x</i> , <i>r</i>)	generalised error density function
@digamma(<i>x</i>), @psi(<i>x</i>)	first derivative of the log gamma function
@dlaplace(<i>xb</i>)	Laplace density function
dlog(<i>x</i>)	first difference of the logarithm
dlog(<i>x</i> , <i>n</i>)	n -th order difference of the logarithm
dlog(<i>x</i> , <i>n</i> , <i>s</i>)	n -th order difference of the logarithm with a seasonal difference at s
@dlogistic(<i>x</i>)	logistic density function
@dlognorm(<i>x</i> , <i>m</i> , <i>s</i>)	log normal density function
@dnegbin(<i>x</i> , <i>n</i> , <i>p</i>)	negative binomial density function
@dnorm(<i>x</i>)	normal density function
@dpareto(<i>x</i> , <i>a</i> , <i>k</i>)	Pareto density function
@dpoisson(<i>x</i> , <i>m</i>)	Poisson density function
@dtdistrib(<i>x</i> , <i>v</i>)	Student's t -distribution density function
@dtoo(<i>str</i>)	converts string to observation number
@dunif(<i>x</i> , <i>a</i> , <i>b</i>)	uniform density function
@dweib(<i>x</i> , <i>m</i> , <i>a</i>)	Weibull density function
@elem(<i>x</i> , "v")	element
@enddate	observations in range
@eqna(<i>str1</i> , <i>str2</i>)	test for equality of strings
@eqna(<i>x</i> , <i>y</i>)	equal to
@erf(<i>x</i>)	error function
@erfc(<i>x</i>)	complementary error function
@exp(<i>x</i>), exp(<i>x</i>)	exponential, e^x
@fact(<i>x</i>)	factorial, $x!$
@factlog(<i>x</i>)	natural logarithm of the factorial, $\log_e(x!)$
@fdistrib(<i>x</i> , <i>v1</i> , <i>v2</i>)	F -distribution p-value
@floor(<i>x</i>)	largest integer not greater than
@fv(<i>r</i> , <i>n</i> , <i>x</i> [, <i>fv</i> , <i>t</i>])	future value
@gamma(<i>x</i>)	(complete) gamma function

Operator / Function	Description
@gammader(<i>x</i>)	first derivataive of the gamma function
@gammainc(<i>x</i> , <i>a</i>)	incomplete gamma function
@gammaincder(<i>x</i> , <i>a</i> , <i>n</i>)	derivative of the incomplete gamma function
@gammaincinv(<i>p</i> , <i>a</i>)	inverse of the incomplete gamma function
@gammalog(<i>x</i>)	logarithm of the gamma function
@gmean(<i>x</i> [, <i>s</i>])	geometric mean
@iff(<i>s</i> , <i>x</i> , <i>y</i>)	recode by condition
@inner(<i>x</i> , <i>y</i> [, <i>s</i>])	inner product
@insert(<i>str1</i> , <i>str2</i> [, <i>n</i>])	string insertion
@instr(<i>str1</i> , <i>str2</i> [, <i>n</i>])	string location
@inv(<i>x</i>)	reciprocal, $1/x$
@isempty(<i>str</i>)	tests for blank strings
@isna(<i>x</i>)	equal to NA
@isperiod(<i>x</i>)	period dummy
@kurt(<i>x</i> [, <i>s</i>])	kurtosis
@kurtsby(<i>arg1</i> , <i>arg2</i> [, <i>s</i>])	kurtosis of <i>arg1</i> observations for each <i>arg2</i> group
@left(<i>str</i> , <i>n</i>)	returns the left part of a string
@len(<i>str</i>)	length of a string
@length(<i>str</i>)	length of a string
@log(<i>x</i>), log(<i>x</i>)	natural logarithm, $\log_e(x)$
@log10(<i>x</i>)	base-10 logarithm, $\log_{10}(x)$
@logit(<i>x</i>)	logistic transform
@logx(<i>x</i> , <i>b</i>)	base- <i>b</i> logarithm, $\log_b(x)$
@lower(<i>str</i>)	lowercases a string
@ltrim(<i>str</i>)	removes spaces from a string
@makedate(<i>arg1</i> [, <i>arg2</i> [, <i>arg3</i>]], <i>fmt</i>)	converts number to date
@map(<i>x</i> [, <i>mapname</i>])	mapped value
@mav(<i>x</i> , <i>n</i>)	<i>n</i> -period backward moving average. NAs are not propagated.
@mavc(<i>x</i> , <i>n</i>)	<i>n</i> -period centered moving average. NAs are not propagated.
@max(<i>x</i> [, <i>s</i>])	maximum
@maxsby(<i>arg1</i> , <i>arg2</i> [, <i>s</i>])	maximum value of <i>arg1</i> observations for each <i>arg2</i> group
@mcor(<i>x</i> , <i>y</i> , <i>n</i>)	<i>n</i> -period backwards moving correlation. NAs are not propagated.

Operator / Function	Description
@mcov(<i>x</i> , <i>y</i> , <i>n</i>)	<i>n</i> -period backwards population moving covariance. NAs are not propagated.
@mcovp(<i>x</i> , <i>y</i> , <i>n</i>)	<i>n</i> -period backwards moving population covariance. NAs are not propagated.
@mcovs(<i>x</i> , <i>y</i> , <i>n</i>)	<i>n</i> -period backwards moving sample covariance. NAs are not propagated.
@mean(<i>x</i> [, <i>s</i>])	mean
@meansby(<i>arg1</i> , <i>arg2</i> [, <i>s</i>])	mean of <i>arg1</i> observations for each <i>arg2</i> group
@median(<i>x</i> [, <i>s</i>])	median
@mediansby(<i>arg1</i> , <i>arg2</i> [, <i>s</i>])	median of <i>arg1</i> observations for each <i>arg2</i> group
@mid(<i>str</i> , <i>n1</i> [, <i>n2</i>])	returns the middle part of a string
@min(<i>x</i> [, <i>s</i>])	minimum
@minner(<i>x</i> , <i>y</i> , <i>n</i>)	<i>n</i> -period backwards inner product of X and Y. NAs are not propagated.
@minsby(<i>arg1</i> , <i>arg2</i> [, <i>s</i>])	minimum value of <i>arg1</i> observations for each <i>arg2</i> group
@mkurt(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards kurtosis. NAs are not propagated.
@mmax(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards moving maximum. NAs are not propagated.
@mmin(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards moving minimum. NAs are not propagated.
@mnas(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards number of NA observations. NAs are not propagated.
@mobs(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards number of non-NA observations. NAs are not propagated.
@mod(<i>x</i> , <i>y</i>)	floating point remainder
@month	month
@movav(<i>x</i> , <i>n</i>)	<i>n</i> -period backward moving average. NAs are propagated.
@movavc(<i>x</i> , <i>n</i>)	<i>n</i> -period centered moving average. NAs are propagated.
@movcor(<i>x</i> , <i>y</i> , <i>n</i>)	<i>n</i> -period backwards moving correlation. NAs are propagated.
@movcov(<i>x</i> , <i>y</i> , <i>n</i>)	<i>n</i> -period backwards moving population covariance. NAs are propagated.
@movcovp(<i>x</i> , <i>y</i> , <i>n</i>)	<i>n</i> -period backwards moving population covariance. NAs are propagated.
@movcovs(<i>x</i> , <i>y</i> , <i>n</i>)	<i>n</i> -period backwards moving sample covariance. NAs are propagated.
@movinner(<i>x</i> , <i>y</i> , <i>n</i>)	<i>n</i> -period backwards inner product of X and Y. NAs are propagated.

Operator / Function	Description
@movkurt(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards kurtosis. NAs are propagated.
@movmax(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards moving maximum. NAs are propagated.
@movmin(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards moving minimum. NAs are propagated.
@movnas(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards number of NA observations
@movobs(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards number of non-NA observations
@movskew(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards skewness. NAs are propagated.
@movstdev(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards moving sample standard deviation. NAs are propagated.
@movstdevp(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards moving population standard deviation. NAs are propagated.
@movstdevs(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards moving sample standard deviation. NAs are propagated.
@movsum(<i>x</i> , <i>n</i>)	<i>n</i> -period backward moving sum. NAs are propagated.
@movsumsq(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards sum-of-squares. NAs are propagated.
@movvar(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards moving population variance. NAs are propagated.
@movvarp(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards moving population variance. NAs are propagated.
@movvars(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards moving sample variance. NAs are propagated.
@mskew(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards skewness. NAs are not propagated.
@mstdev(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards moving sample standard deviation. NAs are not propagated.
@mstdevp(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards moving population standard deviation. NAs are not propagated.
@mstdevs(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards moving sample standard deviation. NAs are not propagated.
@msum(<i>x</i> , <i>n</i>)	<i>n</i> -period backward moving sum. NAs are not propagated.
@msumsq(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards sum-of-squares. NAs are not propagated.
@mvar(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards moving population variance. NAs are not propagated.
@mvarp(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards moving population variance. NAs are not propagated.
@mvars(<i>x</i> , <i>n</i>)	<i>n</i> -period backwards moving sample variance. NAs are not propagated.
@nan(<i>x</i> , <i>y</i>)	recode NAs in X to Y
@nas(<i>x</i> [, <i>s</i>])	number of NAs

Operator / Function	Description
@nasby(<i>arg1</i> , <i>arg2</i> [, <i>s</i>])	number of <i>arg1</i> NA values for each <i>arg2</i> group.
@neqna(<i>str1</i> , <i>str2</i>)	tests for inequality of strings
@neqna(<i>x</i> , <i>y</i>)	not equal to
@now	returns current time
@nper(<i>r</i> , <i>x</i> , <i>pv</i> [, <i>fv</i> , <i>t</i>])	annuity number of periods
@obs(<i>x</i> [, <i>s</i>])	number of observations
@obsby(<i>arg1</i> , <i>arg2</i> [, <i>s</i>])	number of non-NA <i>arg1</i> observations for each <i>arg2</i> group.
@obsid	cross-section observation number
@obsrange	observations in range
@obsmpl	observations in sample
or	logical or
@otod(<i>n</i>)	converts observation number to string
@pc(<i>x</i>)	one-period percentage change (in percent)
@pca(<i>x</i>)	one-period percentage change—annualized (in percent)
@pch(<i>x</i>)	one-period percentage change (in decimal)
@pcha(<i>x</i>)	one-period percentage change—annualized (in decimal)
@pchy(<i>x</i>)	one-year percentage change (in decimal)
@pcy(<i>x</i>)	one-year percentage change (in percent)
@pmt(<i>r</i> , <i>n</i> , <i>pv</i> [, <i>fv</i> , <i>t</i>])	payment amount
@prod(<i>x</i> [, <i>s</i>])	product
@psi(<i>x</i>)	see @digamma
@pv(<i>r</i> , <i>n</i> , <i>x</i> [, <i>fv</i> , <i>t</i>])	present value
@qbeta(<i>s</i> , <i>a</i> , <i>b</i>)	beta quantile function (inverse CDF)
@qbinom(<i>s</i> , <i>n</i> , <i>p</i>)	binomial quantile function (inverse CDF)
@qchisq(<i>p</i> , <i>v</i>)	chi-square quantile function (inverse CDF)
@qexp(<i>p</i> , <i>m</i>)	exponential quantile function (inverse CDF)
@qextreme(<i>p</i>)	extreme value quantile function (inverse CDF)
@qfdist(<i>p</i> , <i>v1</i> , <i>v2</i>)	<i>F</i> -distribution quantile function (inverse CDF)
@qgamma(<i>p</i> , <i>b</i> , <i>r</i>)	gamma quantile function (inverse CDF)
@qged(<i>p</i> , <i>r</i>)	generalised error quantile function (inverse CDF)
@qlaplace(<i>x</i>)	Laplace quantile function (inverse CDF)
@qlogistic(<i>p</i>)	logistic quantile function (inverse CDF)
@qlognorm(<i>p</i> , <i>m</i> , <i>s</i>)	log normal quantile function (inverse CDF)
@qnegbin(<i>s</i> , <i>n</i> , <i>p</i>)	negative binomial quantile function (inverse CDF)

Operator / Function	Description
@qnorm(p)	normal quantile function (inverse CDF)
@qpareto(p,a,k)	Pareto quantile function (inverse CDF)
@qpoisson(p,m)	Poisson quantile function (inverse CDF)
@qtdist(p,v)	Student's <i>t</i> -distribution quantile function (inverse CDF)
@quantile(x,q[,m,s])	quantile
@quantilesby(arg1, arg2, [,s])	quantiles of <i>arg1</i> observations for each <i>arg2</i> group.
@quarter	quarter of the year in a dated workfile
@qunif(p,a,b)	uniform quantile function (inverse CDF)
@qweib(p,m,a)	Weibull quantile function (inverse CDF)
@ranks(x[,o,t,s])	rank
@rate(n,x,pv[,fv,t])	interest
@rbeta(a,b)	beta random number generator
@rbinom(a,b)	binomial random number generator
@rchisq(v)	chi-square random number generator
@recode(s,x,y)	recode by condition
@replace(str1, str2, str3[,n])	replaces a string with another
@rexp(m)	exponential random number generator
@rextreme(p)	extreme value random number generator
@rfdist(v1,v2)	<i>F</i> -distribution random number generator
@rfirst(g)	row-wise first non-NA value.
@rfirsti(g)	row-wise first non-NA index.
@rgamma(b,r)	gamma random number generator
@rged(r)	generalised error random number generator
@right(str, n)	returns the right parts of a string
@rlaplace	Laplace random number generator
@rlast(g)	row-wise last non-NA value.
@rlasti(g)	row-wise last non-NA index.
@rlogistic	logistic random number generator
@rlognorm(m,s)	log normal random number generator
@rmax(g)	row-wise maximum value.
@rmaxi(g)	row-wise maximum index.
@rmean(g)	row-wise mean.
@rmin(g)	row-wise minimum value.

Operator / Function	Description
@rmini(<i>g</i>)	row-wise minimum index.
@rnas(<i>g</i>)	row-wise number of NAs.
@rnegbin(<i>n</i> , <i>p</i>)	negative binomial random number generator
@rnorm	normal random number generator
@robs(<i>g</i>)	row-wise number of non-NAs.
@round(<i>x</i>)	round to the nearest integer
@rpareto(<i>a</i> , <i>k</i>)	Pareto random number generator
@rpoisson(<i>m</i>)	Poisson random number generator
@rstdev(<i>g</i>)	row-wise sample standard deviation.
@rstdevp(<i>g</i>)	row-wise population standard deviation.
@rstdevs(<i>g</i>)	row-wise sample standard deviation.
@rsum(<i>g</i>)	row-wise sum.
@rsumsq(<i>g</i>)	row-wise sum-of-squares.
@rtdist(<i>v</i>)	Student's <i>t</i> -distribution random number generator
@rtrim(<i>str</i>)	removes spaces from a string
@runif(<i>a</i> , <i>b</i>)	uniform random number generator
@rvalcount(<i>g</i> , <i>v</i>)	row-wise count of observations matching <i>v</i> .
@rvar(<i>g</i>)	row-wise population variance.
@rvarp(<i>g</i>)	row-wise population variance
@rvars(<i>g</i>)	row-wise sample variance.
@rweib(<i>m</i> , <i>a</i>)	Weibull random number generator
@seas(<i>x</i>)	seasonal dummy
@sin(<i>x</i>)	sine (argument in radians)
@skew(<i>x</i> [, <i>s</i>])	skewness
@skewsby(<i>arg1</i> , <i>arg2</i> [, <i>s</i>])	skewness of <i>arg1</i> observations for each <i>arg2</i> group.
@sqrt(<i>x</i>), <i>sqr</i> (<i>x</i>)	square root
@stdev(<i>x</i> [, <i>s</i>])	sample standard deviation
@stdevp(<i>x</i> [, <i>s</i>])	population standard deviation
@stdevpsby(<i>arg1</i> , <i>arg2</i> [, <i>s</i>])	standard deviation of <i>arg1</i> observations for each <i>arg2</i> group (division by <i>n</i>).
@stdevsby(<i>arg1</i> , <i>arg2</i> [, <i>s</i>])	sample standard deviation of <i>arg1</i> observations for each <i>arg2</i> group (division by <i>n</i> - 1).
@stdevssby(<i>arg1</i> , <i>arg2</i> [, <i>s</i>])	sample standard deviation of <i>arg1</i> observations for each <i>arg2</i> group (division by <i>n</i> - 1).

Operator / Function	Description
@str(<i>d</i> , <i>[fmt]</i>)	converts a number to a string
@strdate(<i>fmt</i>)	string corresponding to each element in workfile
@strdate(<i>x</i>)	string dates
@strlen(<i>str</i>)	length of a string
@strnow(<i>fmt</i>)	returns current time as a string
@sum(<i>x</i> [, <i>s</i>])	sum
@sumsby(<i>arg1</i> , <i>arg2</i> [, <i>s</i>])	sum of <i>arg1</i> observations for each <i>arg2</i> group.
@sumsq(<i>x</i> [, <i>s</i>])	sum-of-squares
@sumsqsbys(<i>arg1</i> , <i>arg2</i> [, <i>s</i>])	sum of squares of <i>arg1</i> observations for each <i>arg2</i> group.
@tan(<i>x</i>)	tangent (argument in radians)
@tdist(<i>x</i> , <i>v</i>)	<i>t</i> -distribution <i>p</i> -value
@trend	time trend
@trend(<i>[x]</i>)	time trend
@trendc	calendar time trend
@trendc(<i>[x]</i>)	calendar time trend
@trigamma(<i>x</i>)	second derivative of the log gamma function
@trim(<i>str</i>)	removes spaces from a string
@unmap(<i>x</i> , <i>mapname</i>)	unmapped numeric value
@unmaptxt(<i>x</i> , <i>mapname</i>)	unmapped text value
@upper(<i>str</i>)	uppercases a string
@val(<i>str</i> [, <i>fmt</i>])	converts a string to a number
@var(<i>x</i> [, <i>s</i>])	population variance
@varp(<i>x</i> [, <i>s</i>])	population variance
@varpsby(<i>arg1</i> , <i>arg2</i> [, <i>s</i>])	variance of <i>arg1</i> observations for each <i>arg2</i> group (division by <i>n</i>).
@vars(<i>x</i> [, <i>s</i>])	sample variance
@varsby(<i>arg1</i> , <i>arg2</i> [, <i>s</i>])	variance of <i>arg1</i> observations for each <i>arg2</i> group (division by <i>n</i>).
@varssby(<i>arg1</i> , <i>arg2</i> [, <i>s</i>])	sample variance of <i>arg1</i> observations for each <i>arg2</i> group (division by <i>n</i> - 1).
@weekday	day of the week
@year	year

Index

Numerics

2sls (Two-Stage Least Squares) [88](#), [327](#), [503](#), [792](#)
3sls (Three-Stage least squares) [477](#)

A

@abs [883](#)
abs [883](#)
Absolute value [883](#)
@acos [883](#)
add [185](#), [298](#)
Add factor
 assign [272](#)
 initialize [273](#)
Add text to graph [145](#)
addassign [272](#)
addinit [273](#)
addtext [145](#)
align [148](#)
Align multiple graphs [148](#)
@all [781](#)
Alpha [5](#)
 auto-updating [8](#)
 command entries [6](#)
 data members [5](#)
 declare [6](#)
 element functions [5](#)
 genr [10](#)
 make valmap [11](#)
 procs [5](#)
 spreadsheet view [13](#)
 views [5](#)
alpha [6](#)
Alpha series *See* Alpha.
and [883](#)
Anderson-Darling test [365](#)
Andrews test [88](#)
Andrews-Quandt breakpoint test [92](#), [795](#)
anticov [100](#)
Anti-image covariance [100](#)
append [235](#), [275](#), [410](#), [430](#), [479](#), [538](#), [546](#)
Append specification line. *See* append.

AR
 autoregressive error [815](#)
 inverse roots of polynomial in VAR [547](#)
 seasonal [821](#)
ar [815](#)
Arc cosine [883](#)
Arc sine [883](#)
Arc tangent [883](#)
ARCH
 residual LM test for [59](#)
 See also GARCH.
arch [479](#)
ARCH test [59](#)
ARCH-M. *See* ARCH.
archtest [34](#), [684](#)
area [603](#)
Area graph [603](#)
arlm [546](#)
ARMA
 structure view from equation [35](#)
arroots [547](#)
ASCII
 open file as workfile [802](#)
 save table to file [515](#)
@asin [883](#)
@atan [883](#)
Augmented Dickey-Fuller test [219](#), [331](#), [396](#), [796](#)
auto [36](#), [685](#)
Autocorrelation
 compute and display [43](#), [193](#), [363](#), [550](#)
 multivariate VAR residual test [498](#), [565](#)
Autogressive error. *See* AR.
Autoregressive conditional heteroskedasticity
 See ARCH and GARCH.
Auxiliary commands
 summary [675](#)
Average shifted histogram [615](#)
Axis
 rename label [163](#)
 set scaling in graph [170](#)
axis [149](#)

B

Background color [516](#)
band [606](#)
Band graph [606](#)
Band-pass filter [358](#)
bar [609](#)
Bar graph [609](#)
Baxter-King band-pass filter [358](#)
BDS test [357](#)
bdstest [357](#)
Beta function
 CDF [883](#), [884](#)
 density [885](#), [886](#)
 inverse CDF [890](#)
 random number generator [891](#)
Binary
 dependent variables [37](#), [686](#)
 file [802](#)
 model prediction table [73](#)
binary [37](#), [686](#)
Binomial function
 CDF [883](#)
 density [885](#)
 inverse CDF [890](#)
 random number generator [891](#)
block [275](#)
Block structure of model [275](#)
Bollerslev-Wooldridge
 See ARCH.
Boolean
 and [883](#)
 or [890](#)
Bootstrap
 rows of matrix [860](#)
 standard errors for quantile regression [764](#)
Boxplot [613](#)
 customize individual elements [170](#)
boxplot [613](#)
bpf [358](#)
Breakpoint test [41](#), [48](#), [92](#), [715](#), [795](#)
Breusch-Godfrey test
 See Serial correlation.
Breusch-Pagan-Godfrey heteroskedasticity test [59](#)

C

call [870](#)
CARCH
 See ARCH.
Categorize [361](#)
Causality test [186](#), [687](#)
cause [186](#), [687](#)
@cbeta [883](#), [884](#)
@cbinom [883](#)
@cchisq [883](#)
ccopy [688](#)
cd [689](#)
@ceiling [884](#)
Cell
 background color [516](#)
 borders [525](#)
 display format [21](#), [209](#), [264](#), [343](#), [379](#), [468](#),
 [519](#), [583](#)
 font selection [518](#)
 height [523](#)
 indentation [12](#), [22](#), [212](#), [265](#), [344](#), [382](#), [469](#),
 [524](#), [584](#)
 justification [13](#), [23](#), [213](#), [266](#), [345](#), [382](#), [469](#),
 [524](#), [585](#)
 merging multiple [526](#)
 set text color [528](#)
 width [24](#), [214](#), [266](#), [346](#), [383](#), [470](#), [529](#), [586](#)
@cellid [884](#)
censored [40](#), [689](#)
Censored dependent variable models [40](#), [689](#)
Centered moving average [887](#), [888](#)
@cexp [884](#)
@cfdist [884](#)
cfetch [691](#)
@cgamma [884](#)
@cged [884](#)
Change default directory [689](#)
checkderivs [236](#)
Chi-square function
 CDF [883](#)
 density [885](#)
 inverse CDF [890](#)
 random number generator [891](#)
Cholesky [842](#)
@cholesky [842](#)

-
- chow [41](#), [691](#)
 - Chow test [41](#), [48](#), [691](#), [715](#)
 - Christiano-Fitzgerald band-pass filter [358](#)
 - @cimax [842](#)
 - @cimin [843](#)
 - clabel [692](#)
 - @claplace [884](#)
 - classify [361](#)
 - Classify series into categories [361](#)
 - cleartext [548](#)
 - @clogistic [884](#)
 - @clognorm [884](#)
 - Close
 - EViews application [715](#)
 - window [693](#)
 - close [693](#)
 - @cmax [843](#)
 - @cmean [843](#)
 - @cmin [844](#)
 - @cnas [844](#)
 - @cneqbin [884](#)
 - @cnorm [884](#)
 - @cobs [844](#)
 - Coef [15](#)
 - command entries [16](#)
 - data members [16](#)
 - declare [16](#)
 - display name [17](#)
 - fill values [18](#), [366](#)
 - graph views [15](#)
 - procs [15](#)
 - spreadsheet view [24](#)
 - views [15](#)
 - coef [16](#)
 - Coefficient
 - See `coef`.
 - coefcov [42](#), [237](#), [299](#), [432](#), [484](#)
 - Coefficient
 - covariance matrix [42](#), [237](#), [299](#), [432](#), [484](#)
 - update default coef vector [93](#), [244](#), [330](#), [450](#), [505](#)
 - Coefficient restrictions
 - confidence ellipse [39](#), [235](#), [298](#), [430](#), [483](#)
 - count [187](#), [300](#), [548](#), [694](#)
 - Cointegration
 - make cointegrating relations from VEC [562](#)
 - test [187](#), [300](#), [694](#)
 - test from a VAR [548](#)
 - Color
 - keywords for specifying [517](#)
 - @RGB specification [517](#)
 - colplace [844](#)
 - Column
 - extract from matrix [845](#)
 - number of columns in matrix [845](#), [884](#)
 - place in matrix [844](#)
 - stack matrix [867](#)
 - stack matrix (lower triangle) [867](#)
 - width [24](#), [214](#), [266](#), [346](#), [383](#), [470](#), [529](#), [586](#), [776](#)
 - Column statistics
 - maximum [843](#)
 - maximum index [842](#)
 - means [843](#)
 - minimum [844](#)
 - minimum index [843](#)
 - NAs, number of [844](#)
 - observations, number of [844](#)
 - sums [848](#)
 - @columnextract [845](#)
 - @columns [845](#), [884](#)
 - Commands
 - auxiliary [675](#)
 - basic command summary [675](#)
 - execute without opening window [714](#)
 - comment [411](#), [510](#)
 - Comments
 - in spools [411](#)
 - in table cells [510](#)
 - @cond [845](#)
 - Condition number of matrix [845](#)
 - Conditional standard deviation
 - display graph of [53](#), [488](#)
 - Conditional variance
 - make series from ARCH [66](#), [495](#)
 - Confidence ellipse [39](#), [235](#), [298](#), [430](#), [483](#)
 - control [276](#)
 - Convert
 - date to observation number [828](#)
 - matrix object to series or group [857](#)
 - matrix to sym [853](#)
 - observation number to date [833](#)
 - pool to panel [753](#)

- scalar to string [835](#)
- series or group to matrix (drop NAs) [846](#), [864](#)
- series or group to matrix (keep NAs) [864](#)
- string to scalar [837](#)
- sym to matrix [851](#)
- @convert [846](#)
- Coordinates for legend in graph [161](#)
- Copy
 - database [708](#)
 - objects [696](#)
 - workfile page [744](#)
- @cor [847](#), [884](#)
- cor [190](#), [576](#), [702](#)
- correl [43](#), [193](#), [363](#), [550](#)
- Correlation [847](#), [884](#)
 - cross [197](#), [706](#)
 - from matrix [249](#)
 - from sym [455](#)
 - matrix [190](#), [702](#)
 - matrix from vector object [576](#)
 - moving [887](#), [888](#)
- Correlogram [43](#), [193](#), [197](#), [363](#), [550](#), [706](#)
 - squared residuals [44](#)
- correlsq [44](#)
- @cos [884](#)
- Cosine [884](#)
- count [44](#), [703](#)
- Count models [44](#), [703](#)
- @cov [847](#), [884](#)
- cov [194](#), [249](#), [252](#), [455](#), [458](#), [576](#), [704](#)
- Covariance [847](#), [848](#), [884](#)
 - from matrix [252](#)
 - from sym [458](#)
 - from vector [576](#)
 - matrix [194](#), [704](#)
 - moving [888](#)
- @covp [848](#)
- @covs [848](#)
- @cpareto [884](#)
- @cpoisson [884](#)
- Cramer-von Mises test [365](#)
- Create
 - database [707](#), [708](#)
 - workfile [801](#)
 - workfile page [746](#)
- cross [197](#), [706](#)
- Cross correlations [197](#), [706](#)
- Cross product [858](#)
- Cross section member
 - add to pool [185](#), [298](#)
 - define list of in a pool [302](#)
 - text identifier [296](#)
- @csum [848](#)
- CSV
 - save table to file [515](#)
- @ctdist [884](#)
- @cumbmax [884](#)
- @cumbmean [884](#)
- @cumbmin [884](#)
- @cumbnas [884](#)
- @cumbobs [884](#)
- @cumbprod [884](#)
- @cumbstdev [884](#)
- @cumbstdevp [884](#)
- @cumbstdevs [884](#)
- @cumbsum [884](#)
- @cumbsumsq [885](#)
- @cumbvar [885](#)
- @cumbvarp [885](#)
- @cumbvars [885](#)
- @cummax [885](#)
- @cummean [885](#)
- @cummin [885](#)
- @cumnas [885](#)
- @cumobs [885](#)
- @cumprod [885](#)
- @cumstdev [885](#)
- @cumstdevp [885](#)
- @cumstdevs [885](#)
- @cumsum [885](#)
- @cumsumsq [885](#)
- Cumulative Distribution Function [615](#)
- Cumulative statistics
 - maximum [884](#), [885](#)
 - mean [884](#), [885](#)
 - min [885](#)
 - minimum [884](#)
 - NAs, number of [884](#), [885](#)
 - observations, number of [884](#), [885](#)
 - product [884](#), [885](#)
 - standard deviation [884](#), [885](#)

- sum [884](#), [885](#)
- sum of squares [885](#)
- variance [885](#)
- @cumvar [885](#)
- @cumvarp [885](#)
- @cumvars [885](#)
- @cunif [885](#)
- Current
 - date function [870](#)
 - time function [879](#)
- CUSUM test [84](#)
 - of squares [84](#)
- @cweib [885](#)
- D**
- d [885](#)
- Data
 - enter from keyboard [706](#)
 - import [802](#)
- data [706](#)
- Data members
 - alpha series [5](#)
 - coef [16](#)
 - equation [29](#)
 - factor [98](#)
 - group [184](#)
 - matrix [248](#)
 - pool [296](#)
 - rowvector [338](#)
 - series [356](#)
 - spool [410](#)
 - sspace [428](#)
 - sym [454](#)
 - system [476](#)
 - table [510](#)
 - var [544](#)
 - vector [576](#)
- Database
 - copy [708](#)
 - create [708](#)
 - delete [709](#)
 - fetch [717](#)
 - fetch using pool [306](#)
 - Haver Analytics [731](#), [732](#), [733](#)
 - open existing [709](#)
 - open or create [707](#)
 - pack [711](#)
 - rebuild [711](#)
 - rename [712](#)
 - repair [712](#)
 - store object in [787](#)
 - store pool in [325](#)
- Date
 - arithmetic [824](#), [825](#), [831](#)
 - current [870](#)
 - current date [833](#)
 - extract portion of [826](#)
 - format [21](#), [209](#), [264](#), [343](#)
 - make from formatted number [831](#)
- @date [870](#)
- Date format [379](#), [468](#), [519](#), [583](#)
- @dateadd [824](#)
- Dated data table [199](#), [282](#)
- @datediff [825](#)
- @datefloor [825](#)
- datelabel [152](#)
- @datepart [826](#)
- Dates
 - convert from observation number [833](#)
 - convert string to date [827](#)
 - convert to observation number [828](#)
 - current as string [836](#)
 - string representation [827](#)
 - strings representations [827](#)
 - workfile dates as strings [835](#)
- @datestr [827](#)
- @dateval [827](#)
- db [707](#)
- dbcopy [708](#)
- dbcreate [708](#)
- dbdelete [709](#)
- @dbeta [885](#), [886](#)
- @dbinom [885](#)
- dbopen [709](#)
- dbpack [711](#)
- dbrebuild [711](#)
- dbrename [712](#)
- dbrepair [712](#)
- @dchisq [885](#)
- decomp [551](#)
- define [302](#)
- Delete
 - columns in tables [511](#)

- database [709](#)
- objects [713](#)
- objects using pool identifiers [303](#)
- rows in tables [512](#)
- workfile page [750](#)
- delete [303](#), [713](#)
- deletecol [511](#)
- deleterow [512](#)
- Derivatives
 - examine derivs of specification [46](#), [485](#)
 - make series or group containing [65](#)
- derivs [46](#), [485](#)
- describe [303](#)
- Descriptive statistics [25](#), [216](#), [267](#), [347](#), [389](#), [471](#), [587](#), [785](#)
 - by category of dependent variable in equation [70](#)
 - by classification [387](#)
 - equation residuals [60](#)
 - functions [883](#)
 - make series from pool [315](#)
 - matrix functions [841](#)
 - pool series [303](#)
 - valmap [540](#)
- @det [848](#)
- Determinant [848](#)
- @dexp [886](#)
- @dfdist [886](#)
- @dgamma [886](#)
- @dged [886](#)
- Diagonal matrix [855](#)
- Dickey-Fuller test [219](#), [331](#), [396](#), [796](#)
- Difference operator [885](#)
- Directory
 - change working [689](#)
 - EViews executable [872](#)
 - temporary files [878](#)
- Display
 - action [3](#)
 - objects [777](#)
 - spreadsheet tables [13](#), [24](#), [214](#), [267](#), [324](#), [346](#), [383](#), [471](#), [529](#), [540](#), [586](#)
- display [411](#)
- displayname [7](#), [17](#), [47](#), [101](#), [154](#), [198](#), [223](#), [238](#), [255](#), [276](#), [305](#), [338](#), [349](#), [364](#), [412](#), [432](#), [461](#), [486](#), [512](#), [533](#), [538](#), [553](#), [579](#)
- distplot [615](#)

- Distribution graph [615](#)
- Divide
 - matrix element by element [849](#)
- @dlaplace [886](#)
- dlog [886](#)
- @dlogistic [886](#)
- @dlognorm [886](#)
- @dnegbin [886](#)
- @dnorm [886](#)
- do [714](#)
- dot [622](#)
- Dot plot [622](#)
- Double exponential smoothing [384](#), [778](#)
- @dpareto [886](#)
- @dpoisson [886](#)
- draw [154](#)
- Draw lines in graph [154](#)
- drawdefault [156](#)
- DRI database
 - convert to EViews database [714](#)
 - copy from [688](#)
 - fetch series [691](#)
 - read series description [692](#)
- driconvert [714](#)
- Drop
 - cross-section from pool definition [305](#)
 - series from group [199](#)
- drop [199](#), [305](#)
- dtable [199](#)
- @dtdist [886](#)
- @dtoc [828](#)
- Dummy variables
 - automatic creation [816](#)
- @dunif [886](#)
- @dweib [886](#)
- Dynamic forecast [52](#), [433](#), [722](#)

E

- ec [553](#)
- edftest [365](#)
- @ediv [849](#)
- EGARCH [31](#)
 - See also ARCH and GARCH.
- Eigenvalues [102](#), [462](#), [849](#)
- @eigenvalues [849](#)

-
- Eigenvectors [849](#)
 - @eigenvectors [849](#)
 - Elapsed time [791](#), [880](#)
 - Element by element
 - division [849](#)
 - inverse [850](#)
 - matrix functions [840](#)
 - multiplication [850](#)
 - raise to power [850](#)
 - else [871](#)
 - Else clause in if statement [871](#)
 - Empirical CDF [615](#)
 - Empirical distribution test [365](#)
 - Empty string [830](#)
 - @emult [850](#)
 - endif [871](#)
 - endog [277](#), [433](#), [486](#), [555](#)
 - Endogenous variables
 - make series or group [280](#), [436](#)
 - make series or group in system [494](#)
 - make series or group in VAR [562](#)
 - of specification [277](#), [433](#)
 - of specification in system [486](#)
 - of specification in VAR [555](#)
 - endsub [871](#)
 - Enter data from keyboard [706](#)
 - @epow [850](#)
 - @eqna [828](#), [886](#)
 - eqs [277](#)
 - Equation [27](#)
 - coefficient covariance [42](#)
 - command entries [31](#)
 - data members [29](#)
 - declare [47](#)
 - derivatives [46](#)
 - display gradients [58](#)
 - methods [27](#)
 - procs [29](#)
 - stepwise regression [85](#), [786](#)
 - views [27](#)
 - equation [47](#)
 - Equation view
 - of model [277](#)
 - errbar [626](#)
 - Error bar graph [626](#)
 - Error correction model
 - See VEC and VAR.
 - @errorcount [872](#)
 - Errors
 - count in programs [872](#)
 - Estimation methods
 - 2sls [88](#), [327](#), [503](#), [792](#)
 - 3sls [477](#)
 - for factor [97](#)
 - for pool [295](#)
 - for system [475](#)
 - for var [543](#)
 - generalized least squares [62](#), [311](#), [493](#), [735](#)
 - GMM [727](#)
 - least squares [62](#), [311](#), [493](#), [561](#), [735](#)
 - maximum likelihood [117](#), [233](#), [242](#), [441](#)
 - nonlinear least squares [62](#), [311](#), [493](#), [561](#), [735](#)
 - EViews
 - installation directory [872](#)
 - spawn process [783](#)
 - @evpath [872](#)
 - Excel file [802](#)
 - export data to file [25](#), [268](#), [335](#), [347](#), [472](#), [588](#), [812](#)
 - importing data into workfile [262](#), [319](#), [341](#), [466](#), [581](#), [766](#)
 - importing data into workfile coef vector [20](#)
 - exclude [277](#)
 - Exclude variables from model solution [277](#)
 - Exit
 - from EViews [715](#)
 - loop [872](#)
 - subroutine [876](#)
 - exit [715](#)
 - exitloop [872](#)
 - @exp [886](#)
 - exp [886](#)
 - @expand [816](#)
 - @explode [851](#)
 - Exponential function [886](#)
 - CDF [884](#)
 - density [886](#)
 - inverse CDF [890](#)
 - random number generator [891](#)
 - Exponential smoothing [384](#), [778](#)
 - Export
 - workfile data to file [25](#), [268](#), [335](#), [347](#), [472](#), [588](#), [812](#)

workfile page [752](#)
External program [783](#)

Extract
 row vector [861](#)
 submatrix from matrix [865](#)
extract [412](#)

F

facbreak [48](#), [715](#)
@fact [886](#)
factest [716](#)
@factlog [886](#)
factnames [103](#)
Factor [97](#)
 anti-image covariance [100](#)
 command entries [100](#)
 command for estimation [716](#)
 data members [98](#)
 declare [103](#)
 eigenvalue display [102](#)
 estimation output [121](#)
 factor names [103](#)
 fitted covariance [105](#)
 generalized least squares [105](#)
 goodness of fit [104](#)
 iterated principal factors [109](#)
 Kaiser's Measure of Sampling Adequacy [120](#)
 loadings [113](#)
 maximum absolute correlation [116](#)
 maximum likelihood [117](#)
 model [97](#)
 number of observations [121](#)
 observed covariance [121](#)
 partial correlation [125](#)
 partitioned covariance estimation [122](#)
 principal factors [125](#)
 reduced covariance [128](#)
 See also Factor rotation.
 See also Factor scores.
 squared multiple correlation [137](#)
 structure matrix [138](#)
 unweighted least squares [138](#)
factor [103](#)
Factor analysis [97](#)
 residual covariance [129](#)
 See also Factor.
Factor breakpoint test [48](#), [715](#)

Factor rotation [130](#)
 clear [134](#)
 display rotation output [134](#)
Factor scores [135](#)
 saving results [114](#)
Factorial [886](#)
 log of [886](#)
F-distribution function
 CDF [884](#)
 density [886](#)
 inverse CDF [890](#)
 random number generator [891](#)
fetch [306](#), [717](#)
Fetch object [306](#), [717](#)
Files
 open program or text file [737](#)
 temporary location [878](#)
Fill
 matrix [851](#)
 object [339](#), [366](#), [464](#)
 row vector [851](#)
 symmetric matrix [852](#)
 vector [852](#)
fill [256](#), [339](#), [366](#), [464](#), [580](#)
@filledmatrix [851](#)
@filledrowvector [851](#)
@filledsym [852](#)
@filledvector [852](#)
fiml [487](#)
Financial function
 future value [886](#)
 number of periods [890](#)
 payment amount [890](#)
 present value [890](#)
 rate for annuity [891](#)
@first [781](#)
First obs
 row-wise [891](#)
 row-wise index [891](#)
@firstmax [781](#)
@firstmin [781](#)
fit [49](#), [720](#)
fitted [105](#)
Fitted covariance [105](#)
Fitted values [49](#)
Fixed effects

test of joint significance in panel [51](#)
 test of joint significance in pool [308](#)
[fixedtest](#) [51](#), [308](#)
[flatten](#) [413](#)
[@floor](#) [886](#)
 Font selection [518](#)
[for](#) [873](#)
 For loop
 mark end [874](#)
 roundoff error in [877](#)
 start loop [873](#)
 step size [876](#)
 upper limit [879](#)
 Forecast
 dynamic (multi-period) [52](#), [433](#), [722](#)
 static (one-period ahead) [49](#), [720](#)
[forecast](#) [52](#), [433](#), [722](#)
 Formula series [368](#), [725](#)
 alpha [8](#)
[freeze](#) [724](#)
 Freeze view [724](#)
[freq](#) [7](#), [200](#), [367](#)
 Frequency conversion [225](#), [306](#), [696](#), [717](#)
 default method for series [377](#)
 using links [226](#)
 Frequency table
 one-way [7](#), [200](#), [367](#)
[frml](#) [8](#), [368](#), [725](#)
 Full information maximum likelihood [487](#)
 Future value [886](#)
[@fv](#) [886](#)

G

Gamma function
 CDF [884](#)
 density [886](#)
 inverse CDF [890](#)
 logarithm [887](#)
 random number generator [891](#)
[@gammalog](#) [887](#)
 GARCH
 display conditional standard deviation [53](#), [488](#)
 estimate equation [31](#), [680](#)
 generate conditional variance series [66](#), [495](#)
[garch](#) [53](#), [488](#)
 Gauss file [802](#)
 Generalized autoregressive conditional heteroskedasticity
 See ARCH and GARCH.
 Generalized error function
 CDF [884](#)
 density [886](#)
 inverse CDF [890](#)
 random number generator [891](#)
 Generalized inverse [859](#)
 Generalized least squares
 factor estimation [105](#)
 Generate series [370](#), [726](#)
 for pool [309](#)
[genr](#) [10](#), [309](#), [370](#), [726](#)
 See also alpha.
 See also series.
 Geometric mean [887](#)
[@getmaindiagonal](#) [852](#)
 Glejser heteroskedasticity [59](#)
[gls](#) [105](#)
 GLS (generalized least squares) [62](#), [311](#), [493](#), [735](#)
[@gmean](#) [887](#)
 GMM
 estimate by [727](#)
 estimate single equation by [54](#)
 estimate system by [489](#)
[gmm](#) [54](#), [489](#), [727](#)
 Gompit models [37](#), [686](#)
 Goodness of fit
 binary models [88](#)
 factor analysis [104](#)
 Gradients
 display [58](#), [238](#), [435](#), [490](#)
 saving in series [67](#), [241](#), [438](#)
[grads](#) [58](#), [238](#), [435](#), [490](#)
 Granger causality test [186](#), [570](#), [687](#)
 Graph [143](#)
 add text [145](#)
 align multiple graphs [148](#)
 area graph [603](#)
 axis [149](#)
 axis labeling [152](#), [175](#)
 band graph [606](#)
 bar graph [609](#)
 boxplot [613](#)
 change legend or axis name [163](#)
 command entries [145](#)

- commands [143](#)
- create using freeze [724](#)
- creation command entries [603](#)
- declaring [158](#)
- distribution graph [615](#)
- dot plot [622](#)
- drawing lines and shaded areas [154](#), [156](#)
- error bar [626](#)
- extract from spool [412](#)
- high-low-open-close [628](#)
- insert in spool [415](#)
- legend appearance and placement [161](#)
- line graph [630](#)
- merge multiple [163](#), [283](#)
- merging graphs [158](#)
- options for individual elements [171](#)
- options for individual elements of a boxplot [170](#)
- pie graph [633](#)
- place text in [179](#)
- procs [144](#)
- quantile-quantile graph [636](#)
- save to disk [168](#), [515](#), [773](#)
- scatterplot (pairs) graph [647](#)
- scatterplot graph [640](#)
- set axis scale [170](#)
- set options [164](#)
- sort [177](#)
- spike graph [652](#)
- templates [178](#)
- views [144](#)
- XY area graph [656](#)
- XY bar graph [659](#)
- XY line graph [661](#)
- XY pairs graph [664](#)
- graph [158](#)
- graphmode [413](#)
- Group [183](#)
 - add series [185](#)
 - command entries [185](#)
 - convert to matrix [389](#), [390](#), [857](#)
 - convert to matrix (drop NAs) [216](#)
 - convert to matrix (keep NAs) [217](#), [864](#)
 - data members [184](#)
 - declare [202](#)
 - descriptive statistics [216](#)
 - drop series [199](#)
 - graph views [183](#)

- procs [184](#)
- Sort [215](#)
- views [183](#)

- group [202](#)

H

- Harvey heteroskedasticity test [59](#)
- Hausman test [81](#), [318](#)
- Haver Analytics Database
 - convert to EViews database [731](#)
 - display series description [733](#)
 - fetch series from [732](#)
- hconvert [731](#)
- Heteroskedasticity [59](#)
 - quantile slope test [78](#)
 - tests for [59](#)
 - White test in VAR [573](#)
- hetttest [59](#)
- hfetch [732](#)
- High-Low-(Open-Close) graph [628](#)
- hilo [628](#)
- hist [60](#), [370](#), [732](#)
- Histogram [370](#), [615](#), [732](#)
 - equation residuals [60](#)
 - variable width [659](#)
- hlabel [733](#)
- Hodrick-Prescott filter [371](#), [734](#)
- Holt-Winters [384](#), [778](#)
- horizindent [414](#)
- Hosmer-Lemeshow test [88](#)
- hpf [371](#), [734](#)
- HTML
 - open page as workfile [802](#)
 - save table to file [515](#)

I

- @identity [853](#)
- Identity matrix [853](#)
 - extract column [867](#)
- if [873](#)
- If statement
 - else clause [871](#)
 - end of condition [871](#)
 - start of condition [873](#)
 - then [878](#)
- @iff [887](#)

@implode [853](#)
 Import data [802](#)
 Import data from file [20](#), [262](#), [319](#), [341](#), [466](#),
 [581](#), [766](#)
 impulse [555](#)
 Impulse response function [555](#)
 Include
 file in a program file [874](#)
 include [874](#)
 Indentation [12](#)
 Independence test [357](#)
 Initial parameter values [761](#)
 Initialize
 add factor [273](#)
 matrix object [256](#), [464](#), [580](#)
 series [366](#)
 @inner [853](#), [887](#)
 Inner product [853](#), [887](#)
 moving [888](#)
 innov [278](#)
 @insert [829](#)
 insert [415](#)
 insertcol [513](#)
 Inserting columns in tables [513](#)
 Inserting rows in tables [513](#)
 insertrow [513](#)
 @instr [829](#)
 Integer random number [770](#)
 @inv [850](#), [887](#)
 @inverse [854](#)
 Inverse of matrix [854](#)
 element by element [850](#)
 ipf [109](#)
 @isempty [830](#)
 @isna [887](#)
 @isobject [874](#)
 @issingular [855](#)
 Iterated principal factors [109](#)

J

Jarque-Bera
 multivariate normality test for a VAR [491](#), [558](#)
 jbera [491](#), [558](#)
 Johansen cointegration test [187](#), [300](#), [694](#)
 from a VAR [548](#)

K

Kaiser's Measure of Sampling Adequacy [120](#)
 Kalman filter [437](#)
 kdensity [371](#)
 Kendall's tau
 from matrix [249](#), [252](#)
 from sym [455](#), [458](#)
 from vector [576](#)
 kerfit [203](#)
 Kernel
 bivariate regression [203](#)
 density [371](#), [615](#)
 Kolmogorov-Smirnov test [365](#)
 KPSS unit root test [219](#), [331](#), [396](#), [796](#)
 @kronecker [855](#)
 Kronecker product [855](#)
 @kurt [887](#)
 Kurtosis [887](#)
 by category [887](#)
 moving [888](#), [889](#)
 @kurtsby [887](#)

L

label [10](#), [19](#), [61](#), [112](#), [160](#), [203](#), [224](#), [239](#), [257](#),
 [279](#), [310](#), [340](#), [350](#), [372](#), [416](#), [435](#), [465](#), [492](#),
 [514](#), [534](#), [539](#), [559](#), [580](#)
 Label object [7](#), [10](#), [17](#), [19](#), [47](#), [61](#), [101](#), [112](#), [154](#),
 [160](#), [198](#), [203](#), [223](#), [224](#), [238](#), [239](#), [255](#), [257](#),
 [276](#), [279](#), [305](#), [310](#), [338](#), [340](#), [349](#), [350](#), [364](#),
 [372](#), [412](#), [416](#), [432](#), [435](#), [461](#), [465](#), [486](#), [492](#),
 [512](#), [514](#), [533](#), [534](#), [538](#), [539](#), [553](#), [559](#), [579](#),
 [580](#)
 Label values [373](#)
 alpha [12](#)
 Lag
 exclusion test [571](#)
 specify as range [879](#)
 VAR lag order selection [560](#)
 laglen [560](#)
 Lagrange multiplier
 test for ARCH in residuals [59](#)
 Laplace function
 CDF [884](#)
 density [886](#)
 inverse CDF [890](#)
 random number generator [891](#)

- @last [781](#)
 - Last obs
 - row-wise [891](#)
 - row-wise index [891](#)
 - @lastmax [781](#)
 - @lastmin [781](#)
 - Least squares estimation [62](#), [311](#), [493](#), [561](#), [735](#)
 - stepwise [85](#), [786](#)
 - @left [830](#)
 - leftmargin [417](#)
 - Legend
 - appearance and placement [161](#)
 - rename [163](#)
 - legend [161](#)
 - @length [830](#)
 - Length of string [830](#)
 - Lilliefors test [365](#)
 - line [630](#)
 - Line drawing [154](#)
 - Line graph [630](#)
 - linefit [204](#)
 - Link [223](#)
 - command entries [223](#)
 - convert to ordinary series [293](#), [796](#)
 - declare [225](#)
 - procs [223](#)
 - specification [226](#)
 - link [225](#)
 - linkto [226](#)
 - Loadings [113](#)
 - @log [887](#)
 - Log normal function
 - CDF [884](#)
 - density [886](#)
 - inverse CDF [890](#)
 - random number generator [891](#)
 - @log10 [887](#)
 - Logarithm
 - arbitrary base [887](#)
 - base-10 [887](#)
 - difference [886](#)
 - natural [887](#)
 - Logistic function
 - CDF [884](#)
 - density [886](#)
 - inverse CDF [890](#)
 - random number generator [891](#)
 - logit
 - See binary and Logistic model. [62](#), [735](#)
 - Logit models [37](#), [686](#)
 - Logl [233](#)
 - append specification line [235](#)
 - check user-supplied derivatives [236](#)
 - coefficient covariance [237](#)
 - command entries [235](#)
 - data members [234](#)
 - declare [240](#)
 - display gradients [238](#)
 - method [233](#)
 - procs [98](#), [233](#)
 - statements [233](#)
 - views [97](#), [233](#)
 - logl [240](#)
 - @logx [887](#)
 - Loop
 - exit loop [872](#)
 - for [873](#)
 - if [873](#)
 - while [881](#)
 - Lotus file
 - export data to file [25](#), [268](#), [335](#), [347](#), [472](#), [588](#), [812](#)
 - @lower [831](#)
 - Lowercase [831](#)
 - ls [62](#), [311](#), [493](#), [561](#), [735](#)
 - @ltrim [831](#)
- ## M
- MA [818](#)
 - seasonal [822](#)
 - ma [818](#)
 - Make model object [68](#), [241](#), [314](#), [438](#), [496](#), [563](#)
 - makecoint [562](#)
 - @makedate [831](#)
 - makederivs [65](#)
 - @makediagonal [855](#)
 - makeendog [280](#), [436](#), [494](#), [562](#)
 - makefilter [437](#)
 - makegarch [66](#), [495](#)
 - makegrads [67](#), [241](#), [438](#)
 - makegraph [281](#)
 - makegroup [282](#), [314](#)

-
- `makelimits` [68](#)
 - `makemap` [11](#)
 - `makemodel` [68](#), [241](#), [314](#), [438](#), [496](#), [563](#)
 - `makeregs` [69](#)
 - `makesresids` [69](#), [315](#), [439](#), [497](#), [563](#)
 - `makesignals` [439](#)
 - `makestates` [440](#)
 - `makestats` [315](#)
 - `makesystem` [317](#), [564](#)
 - `map` [12](#), [373](#)
 - `Match merge` [225](#), [226](#)
 - `matplace` [856](#)
 - Matrix** [866](#)
 - Cholesky factorization [842](#)
 - columns, number of [845](#)
 - command entries [249](#)
 - condition number [845](#)
 - covert from series or group (drop NAs) [846](#), [864](#)
 - covert from series or group (keep NAs) [864](#)
 - data members [248](#)
 - declare [258](#)
 - descriptive statistics [267](#)
 - element division [849](#)
 - element inverse [850](#)
 - element multiply [850](#)
 - element power [850](#)
 - extract row [861](#)
 - fill values [256](#), [366](#), [851](#)
 - generalized inverse [859](#)
 - graph views [247](#)
 - initialize [256](#)
 - main diagonal [852](#)
 - norm [858](#)
 - of ones [858](#)
 - outer product [858](#)
 - permute rows of [859](#)
 - place submatrix [856](#)
 - place vector in column [844](#)
 - place vector into [861](#)
 - procs [248](#)
 - rank [859](#)
 - resample rows from [860](#)
 - rows, numbers of [861](#)
 - scale rows or columns [862](#)
 - singular value decomposition [865](#)
 - spreadsheet view [267](#)
 - stack columns [867](#)
 - stack lower triangular columns [867](#)
 - subtract submatrix [865](#)
 - trace [866](#)
 - views [247](#)
 - `matrix` [258](#)
 - Matrix commands and functions**
 - descriptive statistics [841](#)
 - element [840](#)
 - matrix algebra [840](#)
 - utility [839](#)
 - Matrix language command entries** [842](#)
 - `@mav` [887](#)
 - `@mavc` [887](#)
 - `@max` [887](#)
 - Maximum** [887](#)
 - by category [887](#)
 - cumulative [884](#), [885](#)
 - matrix columns [843](#)
 - matrix columns index [842](#)
 - moving [888](#), [889](#)
 - row-wise [891](#)
 - row-wise index [891](#)
 - Maximum absolute correlation** [116](#)
 - Maximum likelihood estimation** [242](#), [441](#)
 - factor [117](#)
 - logl [233](#)
 - state space [427](#)
 - `@maxsby` [887](#)
 - `@mcor` [887](#)
 - `@mcov` [888](#)
 - `@mcovp` [888](#)
 - `@mcovs` [888](#)
 - Mean** [888](#)
 - by category [888](#)
 - cumulative [884](#), [885](#)
 - equality test [218](#), [391](#)
 - geometric [887](#)
 - matrix columns [843](#)
 - moving [887](#), [888](#)
 - row-wise [891](#)
 - `means` [70](#)
 - `@meansby` [888](#)
 - Median** [888](#)
 - by category [888](#)
 - equality test [218](#), [391](#)
 - `@median` [888](#)

@mediansby [888](#)

Merge

- graph objects [158](#), [163](#), [283](#)
- into model [163](#), [283](#)
- using links [226](#)

merge [163](#), [283](#)

Messages

- model solution [284](#)

@mid [832](#)

@min [888](#)

Minimum [888](#)

- by category [888](#)
- cumulative [884](#), [885](#)
- matrix columns [844](#)
- matrix columns index [843](#)
- moving [888](#), [889](#)
- row-wise [891](#), [892](#)

@minner [888](#)

@minsby [888](#)

Missing values [887](#)

- code for missing [818](#)
- recoding [889](#)

@mkurt [888](#)

ml [117](#), [242](#), [441](#)

@mmax [888](#)

@mmin [888](#)

@mnas [888](#)

@mobs [888](#)

@mod [888](#)

Model [271](#)

- append specification line [275](#)
- break all model links [293](#), [796](#)
- command entries [272](#)
- declare [284](#)
- equation view [277](#)
- merge into [163](#), [283](#)
- procs [271](#)
- scenarios [286](#)
- update specification [294](#)
- variable view [294](#)
- views [271](#)

model [284](#)

Models

- add factor assignment and removal [272](#)
- add factor initialization [273](#)
- block structure [275](#)
- exclude variables from solution [277](#)

make from equation object [68](#)

make from logl object [241](#)

make from pool object [314](#)

make from sspace object [438](#)

make from system object [496](#)

make from var object [563](#)

make graph of model series [281](#)

make group of model series [282](#)

options for solving [288](#)

options for stochastic simulation [278](#)

options for stochastic solving [290](#)

overrides in model solution [285](#)

solution messages [284](#)

solve [287](#), [782](#)

solve control to match target [276](#)

text representation [291](#), [535](#)

trace [292](#)

trace iteration history [292](#)

track [292](#)

Modulus [888](#)

Moore-Penrose inverse of matrix [859](#)

@movav [888](#)

@movavc [888](#)

@movcor [888](#)

@movcov [888](#)

@movcovp [888](#)

@movcovs [888](#)

move [418](#)

Moving average (MA) [818](#)

Moving statistics

average [888](#)

average, centered [887](#)

centered mean [888](#)

correlation [887](#), [888](#)

covariance [888](#)

inner product [888](#)

kurtosis [888](#)

maximum [888](#), [889](#)

mean [887](#)

minimum [888](#), [889](#)

NAs, number of [888](#), [889](#)

observations, number of [888](#), [889](#)

skewness [889](#)

standard deviation [889](#)

sum [889](#)

sum of squares [889](#)

variance [889](#)

@movinner [888](#)
 @movkurt [889](#)
 @movmax [889](#)
 @movmin [889](#)
 @movnas [889](#)
 @movobs [889](#)
 @movskew [889](#)
 @movstdev [889](#)
 @movstdevp [889](#)
 @movstdevs [889](#)
 @movsum [889](#)
 @movsumsq [889](#)
 @movvar [889](#)
 @movvarp [889](#)
 @movvars [889](#)
 msa [120](#)
 msg [284](#)
 @mskew [889](#)
 @mstdev [889](#)
 @mstdevp [889](#)
 @mstdevs [889](#)
 @msumsq [889](#)
 mtos [857](#)
 Multiplication [890](#)
 Multiply
 matrix element by element [850](#)
 @mvar [889](#)
 @mvarp [889](#)
 @mvars [889](#)

N

NA
 recode [889](#)
 na [818](#)
 name [163](#), [419](#)
 @nan [889](#)
 NAs, number of
 by category [890](#)
 cumulative [884](#), [885](#)
 matrix columns [844](#)
 moving [888](#), [889](#)
 row-wise [892](#)
 @nasby [890](#)
 Nearest neighbor regression [206](#)
 Negative binomial count model [44](#), [703](#)

Negative binomial function
 CDF [884](#)
 density [886](#)
 inverse CDF [890](#)
 random number generator [892](#)
 @neqna [833](#), [890](#)
 next [874](#)
 nnfit [206](#)
 Nonlinear least squares [62](#), [311](#), [493](#), [561](#), [735](#)
 @norm [858](#)
 Norm of a matrix [858](#)
 Normal distribution
 random number [819](#)
 Normal function
 CDF [884](#)
 density [886](#)
 inverse CDF [891](#)
 random number generator [892](#)

@now [833](#)
 @nper [890](#)
 nrnd [819](#)
 Number
 evaluate a string [837](#)
 Number format [21](#), [209](#), [264](#), [343](#), [379](#), [468](#),
 [519](#), [583](#)
 Number of periods
 annuity [890](#)

O

Object
 copy [696](#)
 create using freeze [724](#)
 delete [713](#)
 fetch from database or databank [717](#)
 using pool [306](#)
 label [10](#), [19](#), [61](#), [112](#), [160](#), [203](#), [224](#), [239](#), [257](#),
 [279](#), [310](#), [340](#), [372](#), [416](#), [435](#), [465](#), [492](#),
 [514](#), [559](#), [580](#)
 print [762](#)
 rename [769](#)
 store [787](#)
 store pool [325](#)
 test for existence [874](#)
 @obs [890](#)
 @obsby [890](#)
 Observations, number of [890](#)

- by category [890](#)
- cumulative [884](#), [885](#)
- matrix columns [844](#)
- moving [888](#), [889](#)
- row-wise [892](#)
- observed [121](#)
- ODBC [802](#)
- OLS (ordinary least squares) [62](#), [311](#), [493](#), [561](#), [735](#)
- Omitted variables test [86](#), [326](#), [789](#)
- @ones [858](#)
- Ones matrix [858](#)
- One-way frequency table [7](#), [200](#), [367](#)
- Open
 - database [707](#), [709](#)
 - foreign source data [802](#)
 - text or program files [737](#)
- open [737](#)
- options [164](#), [420](#)
- or [890](#)
- ordered [71](#), [739](#)
- Ordered dependent variable
 - estimating models with [71](#), [739](#)
 - make vector of limit points from equation [68](#)
- @otod [833](#)
- @outer [858](#)
- Outer product [858](#)
- Output
 - display estimation results [72](#), [442](#), [740](#)
 - display factor results [121](#)
 - display Logl results [243](#)
 - display pool results [317](#)
 - display system results [498](#)
 - display VAR results [565](#)
 - redirection [72](#), [243](#), [317](#), [442](#), [498](#), [565](#), [740](#)
- output [72](#), [121](#), [243](#), [317](#), [442](#), [498](#), [565](#), [740](#)
- Output redirection [875](#)
- override [285](#)
- Override variables in model solution [285](#)

P

- pace [122](#)
- Page
 - contract [744](#)
 - copy from [744](#)
 - create new [746](#)
 - define structure [756](#)
 - delete page [750](#)
 - rename [751](#)
 - resize [756](#)
 - save or export [752](#)
 - set active [753](#)
 - stack [753](#)
 - subset from [744](#)
- pageappend [742](#)
- pagecontract [744](#)
- pagecopy [744](#)
- pagecreate [746](#)
- pagedelete [750](#)
- pageload [750](#)
- pagerename [751](#)
- pagesave [752](#)
- pageselect [753](#)
- pagestack [753](#)
- pagestruct [756](#)
- pageunstack [759](#)
- Panel
 - unit root test [219](#), [331](#), [396](#), [796](#)
- param [761](#)
- Parameters [761](#)
- PARCH
 - See ARCH.
- Pareto function
 - CDF [884](#)
 - density [886](#)
 - inverse CDF [891](#)
 - random number generator [892](#)
- partcor [125](#)
- Partial autocorrelation [43](#), [193](#), [363](#), [550](#)
- Partial correlation [43](#), [125](#), [193](#), [363](#), [550](#)
- Partitioned covariance estimation [122](#)
- Payment amount [890](#)
- @pc [890](#)
- @pca [890](#)
- @pch [890](#)
- @pcha [890](#)
- @pchy [890](#)
- @pcy [890](#)
- pdl [820](#)
- PDL (polynomial distributed lag) [820](#)
- Pearson correlation
 - from matrix [249](#), [252](#)

- from sym [455](#), [458](#)
 - from vector [576](#)
 - Percent change
 - one-period [890](#)
 - one-period annualized [890](#)
 - one-year [890](#)
 - @permute [859](#)
 - Permute rows of matrix [859](#)
 - pf [125](#)
 - Phillips-Perron test [219](#), [331](#), [396](#), [796](#)
 - pie [633](#)
 - Pie graph [633](#)
 - @pinverse [859](#)
 - plot [762](#)
 - @pmt [890](#)
 - poff [875](#)
 - Poisson
 - count model [44](#), [703](#)
 - Poisson function
 - CDF [884](#)
 - density [886](#)
 - inverse CDF [891](#)
 - random number generator [892](#)
 - Polynomial distributed lags [820](#)
 - pon [875](#)
 - Pool [295](#)
 - add cross section member [185](#), [298](#)
 - coefficient covariance [299](#)
 - command entries [297](#)
 - data members [296](#)
 - declare [318](#)
 - delete using identifiers [303](#)
 - generate series using identifiers [309](#)
 - make group of pool series [282](#), [314](#)
 - members [295](#)
 - procs [296](#)
 - views [295](#)
 - pool [318](#)
 - Power (raise to) [883](#)
 - matrix element by element [850](#)
 - predict [73](#)
 - Prediction table [73](#)
 - Present value [890](#)
 - Presentation table [199](#)
 - Principal components [204](#), [206](#), [259](#)
 - Principal factors [125](#)
 - iterated [109](#)
 - Print
 - automatic printing [875](#)
 - turn off in program [875](#)
 - print [420](#), [762](#)
 - probit [74](#), [763](#)
 - Probit models [37](#), [686](#)
 - @prod [890](#)
 - Product [890](#)
 - cumulative [884](#), [885](#)
 - Program
 - call subroutine [870](#)
 - counting execution errors [872](#)
 - create new file [763](#)
 - include file [874](#)
 - run [772](#)
 - stop execution [877](#)
 - program [763](#)
 - Programming language command entries [870](#)
 - Pseudoinverse [859](#)
 - @pv [890](#)
- ## Q
- @qbeta [890](#)
 - @qbinom [890](#)
 - @qchisq [890](#)
 - @qexp [890](#)
 - @qfdist [890](#)
 - @qgamma [890](#)
 - @qged [890](#)
 - @qlaplace [890](#)
 - @qlogistic [890](#)
 - @qlognorm [890](#)
 - @qnegbin [890](#)
 - @qnorm [891](#)
 - @qpareto [891](#)
 - @qpoisson [891](#)
 - qqplot [636](#)
 - qrprocess [76](#)
 - qrslope [78](#)
 - qrsymm [79](#)
 - Q-statistic [43](#), [193](#), [363](#), [550](#)
 - qstats [498](#), [565](#)
 - @qtdist [891](#)
 - Quantile [361](#), [615](#)

- by category [891](#)
- @quantile [891](#)
- Quantile function [891](#)
- Quantile regression
 - process estimation [76](#)
 - slope equality test [78](#)
 - symmetric quantiles test [79](#)
- Quantile-Quantile [615](#)
- Quantile-Quantile graph [636](#)
- @quantilesby [891](#)
- @qunif [891](#)
- @qweib [891](#)
- R**
- Ramsey's test [82](#)
- Random effects
 - test for correlated effects (Hausman) [81](#), [318](#)
- Random number
 - generator for normal [819](#)
 - integer [770](#)
 - seed [770](#)
 - uniform [821](#)
- range [766](#)
- ranhaus [81](#), [318](#)
- @rank [859](#)
- Rank (matrix) [859](#)
- Ranks
 - observations in series or vector [891](#)
- @ranks [891](#)
- @rate [891](#)
- Rate for annuity [891](#)
- @rbeta [891](#)
- @rbinom [891](#)
- @rchisq [891](#)
- Read
 - data from foreign file [262](#), [319](#), [341](#), [466](#), [581](#), [766](#)
 - coef [20](#)
- read [262](#), [319](#), [341](#), [466](#), [581](#), [766](#)
 - coef [20](#)
- Reciprocal [887](#)
- @recode [891](#)
- Recode values [887](#), [891](#)
- Recursive least squares [84](#)
 - CUSUM [84](#)
 - CUSUM of squares [84](#)

- Redirect output to file [72](#), [243](#), [317](#), [442](#), [498](#), [565](#), [740](#)
- reduced [128](#)
- Reduced covariance matrix [128](#)
- Redundant fixed effects test [51](#)
 - pool [308](#)
- Redundant variables test [87](#), [327](#), [790](#)
- Regressors
 - make group containing from equation [69](#)
- remove [421](#)
- Rename
 - database [712](#)
 - object [769](#)
- rename [769](#)
- Repair database [712](#)
- @replace [834](#)
- Representations view
 - equation [82](#)
 - pool [322](#)
 - VAR [566](#)
- Reproduced covariance [105](#)
- Resample
 - observations [207](#), [373](#)
 - rows from matrix [860](#)
- @resample [860](#)
- resample [207](#), [373](#)
- reset [82](#), [769](#)
- RESET test [82](#), [769](#)
- Reset timer [790](#)
- residcor [322](#), [443](#), [499](#), [566](#)
- residcov [322](#), [443](#), [500](#), [567](#)
- resids [83](#), [129](#), [323](#), [444](#), [500](#), [567](#)
- Residual covariance
 - display of [129](#)
- Residuals
 - correlation matrix of [322](#), [443](#)
 - system [499](#)
 - VAR [566](#)
 - covariance matrix
 - pool [322](#)
 - covariance matrix of [443](#)
 - covariance matrix of in system [500](#)
 - covariance matrix of in VAR [567](#)
 - display
 - equation [83](#)
 - pool [323](#)

- display of [444](#)
- display of in system [500](#)
- display of in VAR [567](#)
- make series or group containing [69](#), [315](#), [439](#), [497](#), [563](#)
- Resize page [756](#)
- Restricted VAR text [548](#)
- Results
 - display or retrieve [83](#), [243](#), [324](#), [444](#), [501](#), [568](#)
- results [83](#), [243](#), [324](#), [444](#), [501](#), [568](#)
- return [876](#)
- @rexp [891](#)
- @rfdist [891](#)
- @rfirst [891](#)
- @rfirsti [891](#)
- @rgamma [891](#)
- @RGB specification of colors [517](#)
- @rged [891](#)
- @right [834](#)
- @rlaplace [891](#)
- @rlast [891](#)
- @rlasti [891](#)
- @rlogistic [891](#)
- @rlognorm [891](#)
- rls [84](#)
- @rmax [891](#)
- @rmaxi [891](#)
- @rmean [891](#)
- @rmin [891](#)
- @rmini [892](#)
- @rnas [892](#)
- rnd [821](#)
- rndint [770](#)
- rndseed [770](#)
- @rnegbins [892](#)
- @rnorm [892](#)
- @robs [892](#)
- Roots of the AR polynomial in VAR [547](#)
- rotate [130](#)
- rotateclear [134](#)
- rotateout [134](#)
- Round [884](#), [886](#), [892](#)
- @round [892](#)
- Roundoff error in for loops [877](#)
- Row
 - numbers [861](#)
 - place in matrix [861](#)
- Row statistics
 - first non-missing obs [891](#)
 - first non-missing obs index [891](#)
 - last non-missing obs [891](#)
 - last non-missing obs index [891](#)
 - maximum [891](#)
 - maximum index [891](#)
 - mean [891](#)
 - minimum [891](#), [892](#)
 - NAs, number of [892](#)
 - observations matching value, number of [892](#)
 - Observations, number of [892](#)
 - standard deviation [892](#)
 - sum [892](#)
 - sum of squares [892](#)
 - variance [892](#)
- @rowextract [861](#)
- rowplace [861](#)
- @rows [861](#)
- Rowvector [337](#)
 - command entries [338](#)
 - data members [338](#)
 - declare [343](#)
 - extract from matrix [861](#)
 - filled rowvector function [851](#)
 - graph views [337](#)
 - procs [338](#)
 - views [337](#)
- rowvector [343](#)
- @rpareto [892](#)
- @rpoisson [892](#)
- @rstdev [892](#)
- @rstdevp [892](#)
- @rstdevs [892](#)
- @rsum [892](#)
- @rsumsq [892](#)
- @rtldist [892](#)
- RTF
 - save table to file [515](#)
- @rtrim [835](#)
- run [772](#)
- Run batch program [772](#)
- @runif [892](#)
- @rvalcount [892](#)
- @rvar [892](#)

@rvarp [892](#)

@rvars [892](#)

@rweib [892](#)

S

Sample

 @all [781](#)

 command entries [349](#)

 declare [351](#)

 earliest of first panel obs [781](#)

 earliest of last panel obs [781](#)

 @first [781](#)

 @last [781](#)

 latest of first panel obs [781](#)

 latest of last panel obs [781](#)

 procs [349](#)

 set current [780](#)

 set sample specification in [352](#)

 views [349](#)

sample [351](#)

sar [821](#)

SAS file [802](#)

save [168](#), [515](#), [773](#)

Scalar [353](#)

 command entries [353](#)

 declare [353](#)

scalar [353](#)

@scale [862](#)

scale [170](#)

Scale rows or columns of matrix [862](#)

scat [640](#)

scatmat [644](#)

scatpair [647](#)

Scatter diagrams

 matrix of [644](#)

Scatterplot [640](#)

 with kernel fit [203](#)

 with nearest neighbor fit [206](#)

 with regression line fit [204](#)

Scatterplot pairs graph [647](#)

scenario [286](#)

scores [114](#), [135](#)

Scree plot [102](#)

seas [375](#), [774](#)

Seasonal adjustment

 moving average [375](#), [774](#)

 Tramo/Seats [393](#)

 X11 [400](#)

 X12 [402](#)

Seasonal autoregressive error [821](#)

Seasonal line graphs [651](#)

seasplot [651](#)

Second moment matrix [853](#)

Seed random number generator [770](#)

Seemingly unrelated regression. *See* SUR.

Serial correlation

 Breusch-Godfrey LM test [36](#), [685](#)

 multivariate VAR LM test [546](#)

Series [355](#)

 alpha formula [8](#)

 auto-updating [368](#), [725](#)

 bin recoding [361](#)

 categorize [361](#)

 command entries [357](#)

 convert to matrix [389](#), [390](#), [857](#)

 convert to matrix (drop NAs) [216](#)

 convert to matrix (keep NAs) [217](#), [864](#)

 data members [356](#)

 declare [375](#)

 descriptive statistics [389](#)

 element function [356](#)

 fill values [366](#)

 formula [368](#), [725](#)

 frequency conversion default method [377](#)

 graph views [355](#)

 initialize [366](#)

 procs [356](#)

 smoothing [384](#), [734](#)

 Sort [386](#)

 value maps [373](#)

 views [355](#)

series [375](#)

set [352](#)

Set graph date labeling formats [152](#)

setbpelem [170](#)

setcell [774](#)

setcolwidth [776](#)

setconvert [377](#)

setelem [171](#)

setfillcolor [516](#)

setfont [518](#)

setformat [21](#), [209](#), [264](#), [343](#), [379](#), [468](#), [519](#),
[583](#)

-
- setheight [523](#)
 - setindent [12](#), [22](#), [212](#), [265](#), [344](#), [382](#), [469](#), [524](#), [584](#)
 - setjust [13](#), [23](#), [213](#), [266](#), [345](#), [382](#), [469](#), [524](#), [585](#)
 - setline [776](#)
 - setlines [525](#)
 - setmerge [526](#)
 - setobslabel [175](#)
 - settextcolor [528](#)
 - setwidth [24](#), [214](#), [266](#), [346](#), [383](#), [470](#), [529](#), [586](#)
 - Shade region of graph [154](#)
 - sheet [24](#), [214](#), [324](#), [346](#), [383](#), [529](#), [540](#)
 - alpha [13](#)
 - matrix [267](#)
 - sym [471](#)
 - vector [586](#)
 - show [777](#)
 - Show object view [777](#)
 - Signal variables
 - display graphs [445](#)
 - saving [439](#)
 - signalgraph [445](#)
 - @sin [892](#)
 - Sine [892](#)
 - Singular matrix
 - test for [855](#)
 - Singular value decomposition [865](#)
 - @skew [892](#)
 - Skewness [892](#)
 - by category [892](#)
 - moving [889](#)
 - @skewsby [892](#)
 - Slope equality test [78](#)
 - sma [822](#)
 - smooth [384](#), [778](#)
 - Smoothing [734](#)
 - exponential smooth series [384](#), [778](#)
 - signal series [439](#)
 - state series [440](#)
 - smpl [780](#)
 - Solve
 - linear system [863](#)
 - See also* Models.
 - simultaneous equations model [287](#), [782](#)
 - solve [287](#), [782](#)
 - solveopt [288](#)
 - @solvesystem [863](#)
 - Sort [177](#), [215](#), [386](#)
 - vector [863](#)
 - workfile [782](#)
 - @sort [863](#)
 - sort [177](#), [215](#), [386](#), [782](#)
 - spawn [783](#)
 - Spawn process within EViews [783](#)
 - Spearman rank correlation
 - from matrix [249](#), [252](#)
 - from sym [455](#), [458](#)
 - from vector [576](#)
 - spec [244](#), [290](#), [446](#), [501](#)
 - Special expression command entries [815](#)
 - Specification
 - of equation [82](#)
 - of pool [322](#)
 - of VAR [566](#)
 - Specification view [290](#), [446](#)
 - LogL [244](#)
 - system [501](#)
 - spike [652](#)
 - Spike graph [652](#)
 - Spool [409](#)
 - append objects [410](#)
 - command entries [410](#)
 - comment [411](#)
 - data members [410](#)
 - declare spool object [421](#)
 - display contents [411](#)
 - display mode [413](#), [422](#)
 - extract object [412](#)
 - flatten tree hierarchy [413](#)
 - graph display mode [413](#)
 - horizontal indentation [414](#)
 - insert object [415](#)
 - left margin [417](#)
 - move object [418](#)
 - name object [419](#)
 - options [420](#)
 - print object [420](#)
 - procs [409](#)
 - remove object [421](#)
 - table and text display mode [422](#)
 - top margin [422](#)

- vertical indentation [423](#)
- vertical spacing [424](#)
- views [409](#)
- widths [424](#)
- Spreadsheet
 - sort display order [215](#), [386](#)
- Spreadsheet view [13](#), [24](#), [214](#), [267](#), [346](#), [383](#), [471](#), [529](#), [586](#)
 - pool [324](#)
 - See also Table.
 - valmap [540](#)
- SPSS file [802](#)
- @sqrt [892](#)
- sqrt [892](#)
- Square root [892](#)
- Squared multiple correlation [137](#)
- Sspace [427](#)
 - append specification line [430](#)
 - coefficient covariance [432](#)
 - command entries [430](#)
 - data members [428](#)
 - declare [446](#)
 - display signal graphs [445](#)
 - make Kalman filter from [437](#)
 - method [427](#)
 - procs [427](#)
 - specification display [449](#)
 - state graphs [447](#)
 - views [427](#)
- sspace [446](#)
- Stability test [41](#), [48](#), [92](#), [691](#), [715](#), [795](#)
- Stack
 - matrix by column [867](#)
 - sym matrix by lower column triangle [867](#)
 - workfile page [753](#)
- Standard deviation [889](#), [892](#)
 - by category [892](#)
 - cumulative [884](#), [885](#)
 - moving [889](#)
 - row-wise [892](#)
- Starting values [761](#)
- Stata file [802](#)
- statby [387](#)
- State space
 - specification [449](#)
- State variables
 - display graphs of [447](#)
 - final one-step ahead predictions [447](#)
 - initial values [448](#)
 - smoothed series [440](#)
- statefinal [447](#)
- stategraphs [447](#)
- stateinit [448](#)
- Static forecast [49](#), [720](#)
- Statistics
 - compute for subgroups [387](#)
 - pool [303](#)
- stats [25](#), [216](#), [267](#), [347](#), [389](#), [471](#), [540](#), [785](#)
- Status line [785](#)
- statusline [785](#), [876](#)
- @stdev [892](#)
- @stdevpsby [892](#)
- @stdevs [892](#)
- @stdevsby [892](#)
- @stdevssby [892](#)
- step [876](#)
- steps [85](#), [786](#)
- Stepwise regression [85](#), [786](#)
- stochastic [290](#)
- stom [216](#), [389](#), [864](#)
- stomna [217](#), [390](#), [864](#)
- stop [877](#)
- Stop program execution [877](#)
- store [325](#), [787](#)
- Store object in database or databank [787](#)
 - pool [325](#)
- @str [835](#)
- @strdate [835](#)
- String
 - convert date value into [827](#)
 - convert from a number [835](#)
 - convert into date value [827](#)
 - convert to a scalar [837](#)
 - find substring in [829](#)
 - insert string into [829](#)
 - left substring [830](#)
 - length [830](#)
 - length of [836](#)
 - lowercase [831](#)
 - replace substring in [834](#)
 - right substring [834](#)
 - substring from location [832](#)
 - test for blank [830](#)

- test for equality [828](#)
 - test for inequality [833](#)
 - trim spaces from ends [836](#)
 - trim spaces from left end [831](#)
 - trim spaces from right end [835](#)
 - uppercase [837](#)
 - String series [5](#)
 - @strlen [836](#)
 - @strnow [836](#)
 - structure [449](#)
 - Structure workfile page [756](#)
 - @subextract [865](#)
 - Subroutine
 - call [870](#)
 - declare [878](#)
 - mark end [871](#)
 - return from [876](#)
 - subroutine [878](#)
 - Substring [829](#), [832](#)
 - Sum [893](#)
 - by category [893](#)
 - cumulative [884](#), [885](#)
 - matrix columns [848](#)
 - moving [889](#)
 - row-wise [892](#)
 - @sum [893](#)
 - Sum of squares [893](#)
 - by category [893](#)
 - cumulative [885](#)
 - moving [889](#)
 - row-wise [892](#)
 - @sumsby [893](#)
 - @sumsq [893](#)
 - @sumsqby [893](#)
 - SUR
 - estimating [501](#)
 - sur [501](#)
 - Survivor [615](#)
 - svar [568](#)
 - @svd [865](#)
 - Sym [453](#)
 - command entries [455](#)
 - create from lower triangle of square matrix [853](#)
 - create from scalar function [852](#)
 - create square matrix from [851](#)
 - data members [454](#)
 - declare [471](#)
 - descriptive statistics [471](#), [785](#)
 - graph views [453](#)
 - initialize [464](#)
 - procs [454](#)
 - scale rows and columns [862](#)
 - spreadsheet view [471](#)
 - stack columns [867](#)
 - views [453](#)
 - sym [471](#)
 - Symmetric matrix
 - See Sym.
 - Symmetry test [79](#)
 - System [475](#)
 - 3SLS [477](#)
 - append specification line [479](#)
 - coefficient covariance [484](#)
 - command entries [477](#)
 - create from pool [317](#)
 - create from var [564](#)
 - data members [476](#)
 - declare [503](#)
 - derivatives [485](#)
 - display gradients [490](#)
 - FIML estimation [487](#)
 - methods [475](#)
 - procs [476](#)
 - views [475](#)
 - weighted least squares [506](#)
 - system [503](#)
- ## T
- Table [509](#)
 - add comment to cell [510](#)
 - borders and line characteristics [525](#)
 - column widths [24](#), [214](#), [266](#), [346](#), [383](#), [470](#), [529](#), [586](#)
 - command entries [510](#)
 - commands [510](#)
 - create using freeze [724](#)
 - data members [510](#)
 - declare [530](#)
 - delete column [511](#)
 - delete row [512](#)
 - display format for cells [21](#), [209](#), [264](#), [343](#), [379](#), [468](#), [519](#), [583](#)
 - display justification for cells [13](#), [23](#), [213](#), [266](#), [345](#), [382](#), [469](#), [524](#), [585](#)
 - extract from spool [412](#)

- fill (background) color for cells [516](#)
- font for text in cells [518](#)
- horizontal line [776](#)
- indentation for cells [12](#), [22](#), [212](#), [265](#), [344](#),
[382](#), [469](#), [524](#), [584](#)
- insert column [513](#)
- insert in spool [415](#)
- insert row [513](#)
- merge cells [526](#)
- procs [509](#)
- row heights [523](#)
- save to disk [168](#), [515](#), [773](#)
- set and format cell contents [774](#)
- set column width [776](#)
- text color for cells [528](#)
- title [530](#)
- views [509](#)
- table [530](#)
- tablemode [422](#)
- @tan [893](#)
- Tangent [893](#)
- TARCH
 - See ARCH.
- t-distribution function
 - CDF [884](#)
 - density [886](#)
 - inverse CDF [891](#)
 - random number generator [892](#)
- Template [178](#)
- template [178](#)
- @temppath [878](#)
- Test
 - ARCH [59](#)
 - Chow [41](#), [48](#), [691](#), [715](#)
 - correlated random effects [81](#), [318](#)
 - CUSUM [84](#)
 - CUSUM of squares [84](#)
 - exogeneity [570](#)
 - for equality of values [886](#)
 - for inequality of values [890](#)
 - for missing value [887](#)
 - for serial correlation [36](#), [685](#)
 - for serial correlation in VAR [546](#)
 - Goodness of fit [88](#)
 - Granger causality [186](#), [687](#)
 - heteroskedasticity [59](#)
 - heteroskedasticity in VAR [573](#)
 - Johansen cointegration [187](#), [300](#), [694](#)
 - Johansen cointegration from a VAR [548](#)
 - lag exclusion (Wald) [571](#)
 - mean, median, variance equality [218](#)
 - mean, median, variance equality by classification [391](#)
 - omitted variables [86](#), [326](#), [789](#)
 - redundant fixed effects [51](#)
 - pool [308](#)
 - redundant variables [87](#), [327](#), [790](#)
 - RESET [82](#), [769](#)
 - simple mean, median, variance hypotheses [392](#)
 - unit root [219](#), [331](#), [396](#), [796](#)
 - Wald [93](#), [245](#), [334](#), [450](#), [505](#)
 - White [59](#)
- testadd [86](#), [326](#), [789](#)
- testbtw [218](#)
- testby [391](#)
- testdrop [87](#), [327](#), [790](#)
- testexog [570](#)
- testfit [88](#)
- testlags [571](#)
- teststat [392](#)
- Text [533](#)
 - command entries [533](#)
 - declare [291](#), [535](#)
 - extract from spool [412](#)
 - insert in spool [415](#)
 - procs [533](#)
 - views [533](#)
- text [291](#), [535](#)
- Text file
 - open as workfile [802](#)
- Text series [5](#)
- textdefault [179](#)
- then [878](#)
- Three stage least squares [477](#)
- tic [790](#)
- Time
 - current [879](#)
 - current as string [836](#)
- @time [879](#)
- Timer [790](#), [791](#), [880](#)
- title [530](#)
- to [879](#)
- Tobit models [40](#), [689](#)

@toc [880](#)
 toc [791](#)
 topmargin [422](#)
 Trace
 of matrix [866](#)
 @trace [866](#)
 trace [292](#)
 track [292](#)
 Tramo/Seats [393](#)
 tramoseats [393](#)
 Transpose
 matrix [866](#)
 @transpose [866](#)
 transpose [866](#)
 @trim [836](#)
 Truncated dependent variable models [40](#), [689](#)
 tsls
 command [792](#)
 pool [327](#)
 single equation [88](#)
 system [503](#)
 Two-stage least squares
 See 2sls.

U

ubreak [92](#), [795](#)
 uls [138](#)
 Uniform function
 CDF [885](#)
 density [886](#)
 inverse CDF [891](#)
 random number generator [821](#), [892](#)
 Unit root test [219](#), [331](#), [396](#), [796](#)
 Unit vector [867](#)
 @unitvector [867](#)
 Unknown breakpoint test [92](#), [795](#)
 unlink [293](#), [796](#)
 Unstack workfile page [759](#)
 Unweighted least squares [138](#)
 update [294](#)
 updatecoefs [93](#), [244](#), [330](#), [450](#), [505](#)
 @upper [837](#)
 Uppercase [837](#)
 uroot [219](#), [331](#), [396](#), [796](#)
 usage [540](#)

V

@val [837](#)
 Valmap [537](#)
 append specification line [538](#)
 apply to alpha series [12](#)
 apply to series [373](#)
 command entries [537](#)
 declare [541](#)
 find series that use map [540](#)
 make from alpha series [11](#)
 procs [537](#)
 views [537](#)
 valmap [541](#)
 VAR [543](#)
 append specification line [546](#)
 clear restrictions [548](#)
 command entries [546](#)
 data members [544](#)
 declare [572](#)
 estimate factorization matrix [568](#)
 estimating [800](#)
 impulse response [555](#)
 lag exclusion test [571](#)
 lag length test [560](#)
 methods [543](#)
 multivariate autocorrelation test [498](#), [565](#)
 procs [543](#)
 variance decomposition [551](#)
 views [543](#)
 @var [893](#)
 var [572](#)
 Variance [893](#)
 by category [893](#)
 cumulative [885](#)
 equality test [218](#), [391](#)
 moving [889](#)
 row-wise [892](#)
 Variance decomposition [551](#)
 Variance equation
 See ARCH and GARCH.
 @varp [893](#)
 @varpsby [893](#)
 @vars [893](#)
 vars [294](#)
 @varsby [893](#)
 @varssby [893](#)

VEC

- estimating [553](#), [800](#)

- @vec [867](#)

- @vech [867](#)

Vector [575](#)

- command entries [576](#)

- data members [576](#)

- declare [587](#)

- fill [852](#)

- fill values [580](#)

- graph views [575](#)

- initialize [580](#)

- outer product [858](#)

- procs [575](#)

- spreadsheet view [586](#)

- unit [867](#)

- views [575](#)

- vector [587](#)

Vector autoregression

- See VAR.

Vector error correction model

- See VEC and VAR.

- vertindent [423](#)

- vertspacing [424](#)

W

- wald [93](#), [245](#), [334](#), [450](#), [505](#)

- Wald test [93](#), [245](#), [334](#), [450](#), [505](#)

- Watson test [365](#)

Weibull function

- CDF [885](#)

- density [886](#)

- inverse CDF [891](#)

- random number generator [892](#)

- Weighted least squares [506](#)

- Weighted two-stage least squares [507](#)

- wend [880](#)

- wfcreate [801](#)

- wfopen [802](#)

- wfsave [810](#)

- wfselect [811](#)

- while [881](#)

While loop

- end of [880](#)

- start of [881](#)

- white [94](#), [573](#)

- White heteroskedasticity test [59](#)

- width [424](#)

Windows

- command shell [783](#)

- wls [506](#)

Workfile [756](#)

- append contents of workfile page to current page [742](#)

- contract page [744](#)

- convert repeated obs to repeated series [759](#)

- convert repeated series to repeated obs [753](#)

- copy from page [744](#)

- create [705](#)

- create new workfile [801](#)

- create page in [746](#)

- define structured page [756](#)

- delete page [750](#)

- export page [810](#)

- load workfile pages into [750](#)

- open existing [802](#)

- open foreign source into [802](#)

- panel to pool [759](#)

- pool page to panel page [753](#)

- rename page [751](#)

- save [810](#)

- save or export page [752](#)

- set active page [753](#)

- set active workfile and page [811](#)

- sort observations [782](#)

- stack page [753](#)

- unstack page [759](#)

- workfile [812](#)

Write

- coef vector to text file [25](#)

- data to text file [268](#), [335](#), [347](#), [472](#), [588](#), [812](#)

- write [25](#), [268](#), [335](#), [347](#), [472](#), [588](#), [812](#)

- wtsls [507](#)

X

- x11 [400](#)

- x12 [402](#)

- XY (area) graph [656](#)

- XY (bar) graph [659](#)

- XY (line) graph [661](#)

- XY (pairs) graph [664](#)

- xyarea [656](#)

- xybar [659](#)

xyline [661](#)

xypair [664](#)

