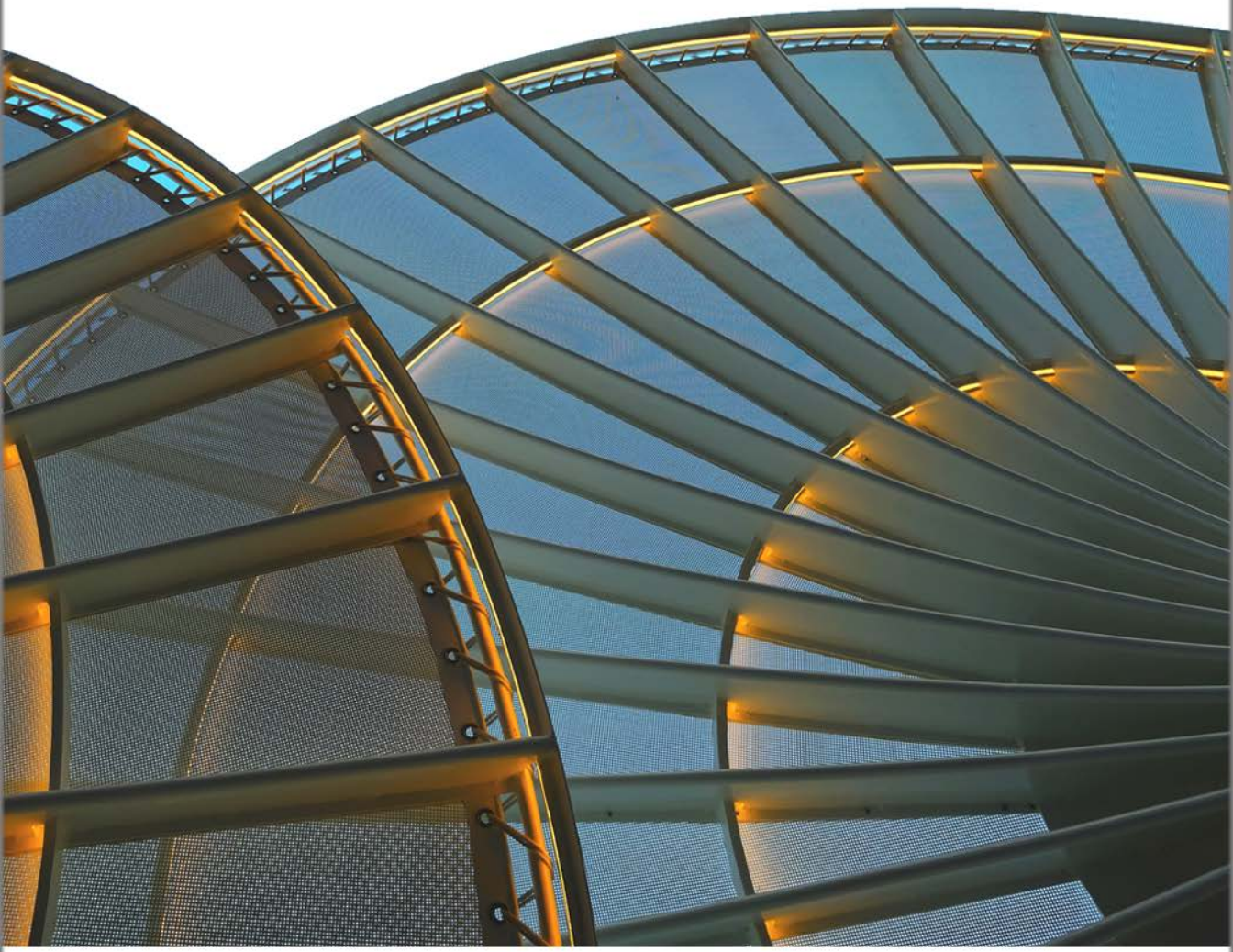
EViews® 14

Object Reference



S&P Global

EViews 14 Object Reference

S&P Global

EViews 14 Object Reference

Disclaimer

The authors and S&P Global Inc. 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

EViews® is a registered trademark of S&P Global Inc. Windows, Excel, PowerPoint, and Access are registered trademarks of Microsoft Corporation. PostScript is a trademark of Adobe Corporation. Bloomberg is a trademark of Bloomberg Finance L.P. All other product names mentioned in this manual may be trademarks or registered trademarks of their respective companies.

Third Party Licenses

This section contains third party notices or additional terms and conditions applicable to certain software technologies which may be used in one or more EViews products and/or services. Please be sure to consult the individual product files, about box, and/or install or manual documentation for specific copyright notices and author attributions. Notices on this page are current for EViews products released on or after October 1, 2017.

- diff template Library - Copyright © 2015 Tatsuhiko Kubo cubicdaiya@gmail.com. All rights reserved.
- FFmpeg Library - FFmpeg is a trademark of Fabrice Bellard, originator of the FFmpeg project.
- GZipHelper - Copyright © 1995-2002 Gao Dasheng dsgao@hotmail.com.
- Java SE Runtime Environment v1.8.0_401 licensed under Oracle Binary Code License Agreement.
- JDemetra+ v3 licensed under European Union Public License v1.2.
- jsonCPP Library - Copyright © 2007-2010 Baptiste Lepilleur and The JsonCPP Authors.
- openssl Library - Copyright © 1998-2016 The OpenSSL Project. All rights reserved.
- libcurl Library - Copyright © 1996-2013, Daniel Stenberg daniel@haxx.se.

S&P Global Inc.
Telephone: (949) 856-3368
e-mail: sales@eviews.com
web: www.eviews.com
June 24, 2024

Table of Contents

Introduction

The three chapters of the EViews *Object Reference* consist of reference material for working with views and procedures of objects in EViews.

- Chapter 1. "Object View and Procedure Reference," on page 3 provides a cross-referenced listing of the commands associated with each object, along with individual entries describing the syntax of each object command.
- Appendix A. "Graph Creation Commands," on page 1267 documents specialized object commands for producing graph views from various EViews data objects.
- Appendix B. "Object Command Summary," on page 1349 offers an alternative indexing of the object views and procedures discussed in the first two chapters, pairing each object command with a list of the objects to which it may be applied.

Chapter 1. Object View and Procedure Reference

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

In addition, there is a link object which, depending on its definition, may be used as an alpha or numeric series (see Link (p. 523)).

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_type_declaration 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 *pro-*

cedure 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:

action ..... is one of the four verb commands (`do`, `freeze`, `print`, `show`).

action_opt ..... an option that modifies the default behavior of the action.

object_name..... the name of the object to be acted upon.

view_or_proc ..... the object view or procedure to be 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 objects 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 1. "Object and Command Basics," on page 3 of the *Command and Programming Reference* for additional discussion of using commands in EViews.

Alpha

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

Alpha Declaration

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 empty (blank) values, "".

Alpha Views

Alpha Procs

Alpha Data Members

@attr("*arg*") ..... string containing the value of the *arg* attribute, where the argument is specified as a quoted string.

@description.....string containing the alpha object's description (if available).

@detailedtype .....string describing the object type: "ALPHA", if an ordinary alpha series, or "LINK", if defined by link.

@displayname .....string containing alpha object's display name. If the Alpha has no display name set, the name is returned.

@first .....string containing the date or observation number of the first non-blank observation of the alpha. In a panel workfile, the first date at which any cross-section has a non-blank observation is returned.

@firstall.....returns the same as `@first`, however in a panel workfile, the first date at which all cross-sections have a non-blank observation is returned.

@hilo .....string containing the alpha series object's high-to-low frequency conversion method.

@last .....string containing the date or observation number of the last non-blank observation of the alpha. In a panel workfile, the last date at which any cross-section has a non-blank observation is returned.

@lastall .....returns the same as `@last`, however in a panel workfile, the last date at which all cross-sections have a non-blank observation is returned.

@lohi .....string containing the alpha series object's low-to-high frequency conversion method.

@name.....string containing the alpha object's name.

@remarks.....string containing the alpha object's remarks (if available).

@type.....string describing the object type: "ALPHA".

@updatetime .....string representation of the time and date at which the alpha was last updated.

(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 alpha series, or an explicit or implicit expression for generating the 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 (optionally, you may use `Alpha::genr` (p. 14) with a previously created alpha series). 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 224 of *User's Guide I* for additional discussion.

See also Alpha::genr (p. 14).

clearhist	Alpha Procs

Clear the contents of the history attribute.

Removes the alpha's history attribute, as shown in the label view of the alpha.

Syntax

alpha_name.**clearhist**

Examples

```
a1.clearhist
a1.label
```

The first line removes the history from the alpha A1, and the second line displays the label view of A1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also Alpha::label (p. 15).

clearremarks	Alpha Procs

Clear the contents of the remarks attribute.

Removes the alpha's remarks attribute, as shown in the label view of the alpha.

Syntax

alpha_name.**clearremarks**

Examples

```
a1.clearremarks
a1.label
```

The first line removes the remarks from the alpha object A1, and the second line displays the label view of A1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also Alpha::label (p. 15).

copy	Alpha Procs

Creates a copy of the alpha series.

Creates either a named or unnamed copy of the alpha series.

Syntax

alpha_name.**copy**

alpha_name.**copy** *dest_name*

Examples

```
a1.copy
```

creates an unnamed copy of the alpha series A1.

```
a1.copy a2
```

creates A2, a copy of the alpha series A1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

display	Alpha Views

Display table, graph, or spool output in the alpha object window.

Display the contents of a table, graph, or spool in the window of the alpha object.

Syntax

alpha_name.**display** *object_name*

Examples

```
alpha1.display tab1
```

Display the contents of the table TAB1 in the window of the object ALPHA1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

displayname	Alpha Procs

Display name for an alpha object.

Attaches a display name to an alpha object. The display name 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 various 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 123 of *User's Guide I* for a discussion of labels and display names. See also `Alpha::label` (p. 15).

dups	Alpha Views

Duplicate observations display for observations in the series.

Syntax

series_name.**dups**(*opts*)

By default, EViews displays a summary table showing the number of duplicate groups of a given size, but you may use the options to display an alternative view.

Of particular note is that the spreadsheet and individual duplicates displays are interactive - clicking on rows in one will open the display to show the other. Thus, clicking on a duplicate in the spreadsheet view will jump to show all of the observations that share that duplicate. Similarly, clicking on an observation in the shared individual duplicates view will jump to the corresponding observation in the full spreadsheet.

Options

graph	Display observation graph showing duplicates.
sheet	Display spreadsheet view of duplicates.
individ	Display first individual duplicates.

Examples

```
alpha1.dups
```

displays the duplicates summary for the alpha series ALPHA1.

```
alpha1.dups(sheet)
```

displays a spreadsheet showing highlighted duplicates.

Cross-references

freq	Alpha Views

Compute frequency tables.

`freq` performs a one-way frequency tabulation.

Frequencies are computed for the current sample of observations. Observations with NAs (blank strings) are dropped unless included by option. You may use options to control automatic binning (grouping) of values and the order of the entries of the table.

Syntax

alpha_name.**freq***(options)*

Options

dropna *(default)* / keepna	[Drop/Keep] NA as a category.
n, obs, count *(default)*	Display frequency counts.
nocount	Do not display frequency counts.
prompt	Force the dialog to appear from within a program.
p	Print the table.
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.
sort = *arg* *(default* = "lohi"*)*	Sort order for entries in the frequency table: high data value to low ("hilo"), low data value to high ("lohi" – *default*), high frequency to low ("freqhilo"), low frequency to high ("freqlohi").
prompt	Force the dialog to appear from within a program.
p	Print the table.

Examples

```
names.freq
```

tabulates the values in NAMES in ascending alphabetical order, displaying counts, percentages, and cumulatives.

```
names.freq(sort=freqhilo)
```

tabulates NAMES with the table rows ordered from values with highest frequency to lowest.

Cross-references

See "One-Way Tabulation" on page 467 of *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 directly.

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 freezes the contents of the series at the current values.

Cross-references

See "Auto-Updating Series" on page 219 of *User's Guide I*.

See also `Link::link` (p. 528).

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. 8) 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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User's Guide I* for a discussion of labels.

See also `Alpha::displayname` (p. 11).

makemap — Alpha Procs

Create a 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

prompt	Force the dialog to appear from within a program.
nosort	Do not alphabetically sort the alpha series values 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 224 of *User's Guide I* for a discussion of alpha series and "Value Maps" on page 235 of *User's Guide I* for a discussion of valmaps.

map — Alpha Procs

Assign or remove value map to alpha series.

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 235 of *User's Guide I* for a discussion of valmap objects in EViews.

olepush	Alpha Procs

Push updates to OLE linked objects in open applications.

Syntax

alpha_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

setattr	Alpha Procs

Set the object attribute.

Syntax

alpha_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the "revised" attribute in the object A to the string "never", and extracts the attribute into the string object S.

Cross-references

See "Adding Custom Attributes in the Label View" on page 123 and "Adding Your Own Label Attributes" on page 70 in *User's Guide I*.

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 indentation setttings are taken from the Global Defaults for spreadsheet views ("Spreadsheet Data Display" on page 1019 of *User's Guide I*) at the time the spreadsheet was 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. 18) for details on setting spreadsheet justification.

setjust	Alpha Procs

Set the horizontal justification for all cells in the spreadsheet view of the alpha series.

Syntax

alpha_name.**setjust** *just_arg*

where *format_arg* may be set to left, center, right, or auto (strings are left-justified and numbers are right-justified). Default display settings can be set in General Options; see "Spreadsheet Data Display" on page 1019 of *User's Guide I*.

Examples

```
a1.setjust left
```

left-justifies the cells in the spreadsheet view of the alpha series A1.

Cross-references

See also `Alpha::setindent` (p. 18) for details on setting spreadsheet indentation.

sheet	Alpha Views

Spreadsheet view of an alpha series object.

Syntax

alpha_name.**sheet***(options)*

Options

w	Wide. In a panel this will switch to the unstacked form of the panel (dates along the side, cross-sections along the top).
t	Transpose.
a	All observations (ignore sample).
nl	Do not display labels.
p	Print the spreadsheet view.

Examples

```
a1.sheet
```

displays the spreadsheet view of the alpha series A1.

```
a1.sheet(t)
```

displays the observations in A1 in the current sample in a wide spreadsheet.

```
a1.sheet(nl)
```

shows the series without labels.

```
a1.sheet(a)
```

shows all of the observations in the workfile.

Cross-references

See "Alpha Series," beginning on page 224 of the *User's Guide I* for a discussion of the spreadsheet view of alpha series.

sort	Alpha Procs

Change display order for an alpha series spreadsheet.

The `sort` command changes the sort order settings for spreadsheet display of the alpha series.

Syntax

alpha_name.**sort***([opt])*

By default, EViews will sort the alpha series alphabetically. 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

```
a1.sort
```

changes the display order for the alpha series A1 so that spreadsheet rows are ordered alphabetically.

```
a1.sort(-)
```

sorts in descending order.

```
a1.sort(obs)
```

returns the display order for alpha series A1 to the original (by observation).

Cross-references

See "Spreadsheet" on page 642 of *User's Guide II* for additional discussion.

Coef

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

Coef Declaration

coef ..... declare coefficient vector (p. 26).

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, a new coef vector will be initialized to zero.

See also `param` (p. 564) in the *Command and Programming Reference* for information on initializing coefficients, and the entries for each of the estimation objects (**Equation**, **Logl**, **Pool**, **Sspace**, **System**, and **Var**) for additional methods of accessing coefficients.

Coef Views

display ..... display table, graph, or spool in object window (p. 27).
label ..... label view (p. 38).
sheet ..... spreadsheet view of the coefficient (p. 46).
stats ..... descriptive statistics (p. 47).

Coef Procs

clearcollabels ..... clear the column label in a coef object (p. 24).
clearhist ..... clear the contents of the history attribute (p. 25).
clearremarks ..... clear the contents of the remarks attribute (p. 25).
clearrowlabels ..... clear the row labels in a coef object (p. 26).
copy ..... creates a copy of the coef (p. 27).
displayname ..... set display name (p. 28).
export ..... save as Excel 2007 XLSX, CSV, tab-delimited ASCII text, RTF, HTML, Enhanced Metafile, PDF, TEX, or MD file on disk (p. 28).
fill ..... fill the elements of the coefficient vector (p. 32).
import ..... imports data from a foreign file into the object (p. 33).
olepush ..... push updates to OLE linked objects in open applications (p. 39).
read ..... import data into coefficient vector (p. 40).
resize ..... resize the coef object (p. 42).

Coef Graph Views

Graph creation views are discussed in detail in "Graph Creation Command Summary" on page 1267.

Coef Data Members

@attr("*arg*").....string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@collabels .....string containing the column label of the coef.
@description.....string containing the Coef object's description (if available).
@detailedtype .....string describing the object type: "COEF".
@displayname .....string containing the Coef object's display name. If the Coef has no display name set, the name is returned.
@droprow(*arg*) .....Returns the Coef with the rows defined by *arg* removed. *arg* may be an integer, vector of integers, string, or svector of strings. Integer values correspond to rows and string values correspond to previously defined row labels.
@name.....string containing the Coef object's name.
@remarks.....string containing the Coef object's remarks (if available).

@row(*arg*) ..... Returns the rows defined by *arg*. *arg* may be an integer, vector of integers, string, or svector of strings. Integer values correspond to rows and string values correspond to previously defined row labels.

@rowlabels ..... string containing the row labels of the coef.

@rows ..... number of rows in the coef.

@type ..... string describing the object type: "COEF".

@updatetime ..... string representation of the time and date at which the Coef was last updated.

(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.

clearcollabels	Coef Procs

Clear the column label in a coef object.

Syntax

coef_name.**clearcollabels**

Examples

```
c1.clearcollabels
```

clears the custom column label from the coef C1.

Cross-references

See also Coef::clearrowlabels (p. 26).

clearhist

Coef Procs

Clear the contents of the history attribute.

Removes the coef's history attribute, as shown in the label view of the coef.

Syntax

coef_name.**clearhist**

Examples

```
c1.clearhist
c1.label
```

The first line removes the history from the coef C1, and the second line displays the label view of C1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also Coef::label (p. 38).

clearremarks

Coef Procs

Clear the contents of the remarks attribute.

Removes the coef's remarks attribute, as shown in the label view of the coef.

Syntax

coef_name.**clearremarks**

Examples

```
c1.clearremarks
c1.label
```

The first line removes the remarks from the coef C1, and the second line displays the label view of C1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Coef::label` (p. 38).

clearrowlabels	Coef Procs

Clear the row labels in a coef object.

Syntax

coef_name.**clearrowlabels**

Examples

```
c1.clearrowlabels
```

clears the custom row labels from the coef C1.

Cross-references

See also `Coef::clearcollabels` (p. 24).

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. 1263).

copy	Coef Procs

Creates a copy of the coef.

Creates either a named or unnamed copy of the coef.

Syntax

coef_name.**copy**

coef_name.**copy** *dest_name*

Examples

```
c1.copy
```

creates an unnamed copy of the coef C1.

```
c1.copy c2
```

creates C2, a copy of the coef C1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

display	Coef Views

Display table, graph, or spool output in the coef object window.

Display the contents of a table, graph, or spool in the window of the coef object.

Syntax

coef_name.**display** *object_name*

Examples

```
coef1.display tab1
```

Display the contents of the table TAB1 in the window of the object COEF1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

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 123 of *User's Guide I* for a discussion of labels and display names. See also `Coef::label` (p. 38).

export	Coef Procs

Export coef vector to disk as an Excel 2007 XLSX, CSV, tab-delimited ASCII text, RTF, HTML, Enhanced Metafile, LaTeX, PDF, or Markdown file.

Syntax

coef_name.**export***(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 set in the **File Locations** global options.

The base syntax for writing Excel 2007 files is:

coef_name.**export**(*options*) *[path\]file_name [table_description]*

where *table_description* contains:

- "range = *arg*", where arg is top left cell of the destination 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 defined inside the Excel workbook to refer to a range or cell may specify the cells to read.

Options

t = *file_type (default* = "csv"*)*	Specifies the file type, where *file_type* may be: "excelxml" (Excel 2007 (xml)),"csv" (CSV - comma-separated), "rtf" (Rich-text format), "txt" (tab-delimited text), "html" (HTML - Hypertext Markup Language), "emf" (Enhanced Metafile), "pdf" (PDF - Portable Document Format), "tex" (LaTeX), or "md" (Markdown). Files will be saved with the ".xlsx", ".csv", ".rtf", ".txt", ".htm", ".emf", ".pdf", ".tex", or ".md" extensions, respectively.
s = *arg*	Scale size, where *arg* is from 5 to 200, representing the percentage of the original table size (only valid for HTML or RTF files).
n = *string*	Replace all cells that contain NA values with the specified string. "NA" is the default.
h / -h	Include(/do not include) column and row headers. The default is to not include the headers
prompt	Force the dialog to appear from within a program.

PDF Options

landscape	Save in landscape mode (the default is to save in portrait mode).
size = *arg* (*default* = "letter")	Page size: "letter", "legal", "a4", and "custom".
width = *number* (*default* = 8.5)	Page width in inches if "size = custom".
height = *number* (*default* = 11)	Page height in inches if "size = custom".
leftmargin = *number (default* = 0.5*)*	Left margin width in inches.
rightmargin = *number (default* = 0.5*)*	Right margin width in inches.
topmargin = *number (default* = 1*)*	Top margin width in inches.
bottommargin = *number (default* = 1*)*	Bottom margin width in inches.

LaTeX Options

texspec / -texspec	[Include / Do not include] the full LaTeX documentation specification in the LaTeX output. The default behavior is taken from the global default settings.

Excel Options

mode = *arg*	Specify whether to create a new file, overwrite an existing file, or update an existing file. arg may be "create" (create new file only; error on attempt to overwrite) or "update" (update an existing file, only overwriting the area specified by the range = *table_description*). If the "mode = " option is not used, EViews will create a new file, unless the file already exists in which case it will overwrite it. Note that the "mode = update" option is only available for Excel in 1) Excel versions through 2003, if Excel is installed, and 2) Excel 2007 (xml). Note: Excel does not need to be installed for Excel 2007 writing.

Excel 2007 Options

mode = *arg*	Specify whether to create a new file, overwrite an existing file, or update an existing file. arg may be "create" (create new file only; error on attempt to overwrite) or "update" (update an existing file, only overwriting the area specified by the range = *table_description*). If the "mode = " option is not used, EViews will create a new file, unless the file already exists in which case it will overwrite it. Note that the "mode = update" option is only available for Excel in 1) Excel versions through 2003, if Excel is installed, and 2) Excel 2007 (xml). Note: Excel does not need to be installed for Excel 2007 writing.
cellfmt = *arg*	Specify whether to use EViews, pre-existing, or remove cell formatting (colors, font, number formatting when possible, column widths and row heights) for the written range. *arg* may be "eviews" (replace current formatting in the file with the same cell formatting in EViews), "preserve" (leave current cell formatting already in the Excel file), or "clear" (remove current formatting and do not replace).
strlen = *arg* (default = 256)	Specify the maximum the number of characters written for cells containing text. Strings in cells which are longer the max, will be truncated.

Examples

The command:

```
coef1.export myvector
```

exports data in COEF1 to a CSV file named "myvector.CSV" in the default directory.

```
coef1.export(h,t=csv, n="NaN") myvector
```

saves the contents of COEF1 along with the column and row headers to a CSV file named "myvector.CSV" and writes all NA values as "NaN".

```
coef1.export(h,t=html, s=50) myvector
```

writes the data of COEF1 along with the column and row headers to a HTML file named "myvector.HTM" at half of the original size.

```
coef1.export(n=".", r=B) myvector
```

exports the data in the second column to a CSV file named "myvector.CSV", and writes all NA values as ".".

```
coef1.export(t=excelxml, cellfmt=clear, mode=update) myvector
    range=Country!b5
```

writes the data in COEF1 to the “Country” sheet at cell B5 in the preexisting “myvector.XLSX” Excel file. All cell formatting is cleared.

Cross-references

See also `Coef::write` (p. 48).

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 not be modified unless the “l” option is specified. However, if you list more values than the coef vector 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 coef vector.
o = *integer* *(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,l) -1
```

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

Cross-references

See “Fill assignment” on page 281 of the *Command and Programming Reference* for further discussion of the fill procedure.

import	Coef Procs

Imports data from a foreign file into the coef object.

Syntax

coef_name.**import***([type =]) source_description import_specification*

- *source_description* should contain a description of the file from which the data is to be imported. The specification of the description is usually just the path and file name of the file, however you can also specify more precise information. See `wfopen` (p. 640) of the *Command and Programming Reference* for more details on the specification of *source_description*.
- The optional "type = " option may be used to specify a source type. For the most part, you should not need to specify a "type = " option as EViews will automatically determine the type from the filename. The following table summaries the various source formats and along with the corresponding "type = " keywords:

	Option Keywords
Excel (through 2003)	"excel"
Excel 2007 (xml)	"excelxml"
HTML	"html"
Text / ASCII	"text"

- *import_specification* can be used to provide additional information about the file to be read. The details of *import_specification* will depend upon the type of file being imported.

Excel Files

The syntax for reading Excel files is:

coef_name.**import(type = excel***[xml]*) *source_description [table_description] [variables_description]*

The following *table_description* elements may be used when reading Excel data:

- "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.

- "byrow", transpose the incoming data. This option allows you to read files where the series are contained in rows (one row per series) rather than columns.

The optional *variables_description* may be formed using the elements:

- "colhead = *int*", number of table rows to be treated as 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 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 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.

Excel Examples

```
coef_obj.import "c:\data files\data.xls"
```

loads the active sheet of "data.XLSX" into the VEC_NAME vector object.

```
coef_obj.import "c:\data files\data.xls" range="GDP data"
```

reads the data contained in the "GDP data" sheet of "data.XLS" into the COEF_OBJ object.

HTML Files

The syntax for reading HTML pages is:

coef_name.**import(type = html)** *source_description [table_description] [variables_description]*

The following *table_description* elements may be used when reading an HTML file or page:

- "table = *arg*", where *arg* specifies which HTML 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.
- "byrow", transpose the incoming data. This option allows you to import files where the series are contained in rows (one row per series) rather than columns.

The optional *variables_description* may be formed using the elements:

- "colhead = *int*", number of table rows to be treated as 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.

HTML Examples

```
coef_obj.import "c:\data.html"
```

loads into the COEF_OBJ the data located in the HTML file "Data.HTML" located on the C:\ drive

```
coef_obj.import(type=html) "http://www.tradingroom.com.au/apps/
    mkt/forex.ac" colhead=3
```

loads into a coef object called COEF_OBJ the data with the given URL located on the website site "http://www.tradingroom.com.au". The column header is set to three rows.

Text and Binary Files

The syntax for reading text or binary files is:

coef_name.**import(type** = *arg***)** *source_description [table_description] [variables_description]*

If a *table_description* is not provided, EViews will attempt to read the file as a free-format text file. The following *table_description* elements may be used when reading a text or binary file:

- "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.
- “byrow”, transpose the incoming data. This option allows you to import files where the series are contained in rows (one row per series) rather than columns.
- “lastcol”, include implied last column. For lines that end with a delimiter, this option adds an additional column. When importing a CSV file, lines which have the delimiter as the last character (for example: “name, description, date”), EViews normally determines the line to have 3 columns. With the above option, EViews will determine the line to have 4 columns. Note this is not the same as a line containing “name, description, date”. In this case, EViews will always determine the line to have 3 columns regardless if the option is set.

A central component of the *table_description* element is the format statement. You may specify the data format using the following table descriptors:

- Fortran Format:

  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 Format:

  rformat = "*[n1]Type[Width][.Precision], [n2]Type[Width][.Precision], ...*)"

  where optional type is "$" for string or "#" for number, and *n1*, *n2*, *n3*, *n4*, *etc*. are the range of columns containing the data.

- C printf/scanf Format:

  cformat = "*fmt*"

  where *fmt* follows standard C language (printf/scanf) format rules.

The optional *variables_description* may be formed using the elements:

- "colhead = *int*", number of table rows to be treated as 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.

Text and Binary File Examples (.txt, .csv, etc.)

```
coef_obj.import c:\data.csv skip=5
```

reads "Data.CSV" into a `coef_obj`, skipping the first 5 rows.

```
coef_obj.import(type=text) c:\date.txt delim=comma
```

loads the comma delimited data "Date.TXT" into the COEF_OBJ matrix object.

Cross-references

See also `Coef::read` (p. 40).

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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User's Guide I* for a discussion of labels.

See also `Coef::displayname` (p. 28).

olepush	Coef Procs

Push updates to OLE linked objects in open applications.

Syntax

coef_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

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

prompt	Force the dialog to appear from within a program.

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 *default* 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.

skipcol = *integer*	Number of columns to skip. Must be used with the "*rect*" option.
skiprow = *integer*	Number of rows to skip. Must be used with the "rect" option.
comment = *symbol*	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.
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

```
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 152 of *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 51 of *User's Guide I* and `wfopen` (p. 640) in the *Command and Programming Reference*.

See also `Coef::write` (p. 48).

resize	Coef Procs

Resize the coef object.

Syntax

coef_name.**resize** *rows*

Examples

```
c1.resize 20
```

resizes the coef C1 to 20 rows, retaining the contents of any existing elements and initializing new elements to 0.

setattr	Coef Procs

Set the object attribute.

Syntax

coef_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the “revised” attribute in the object A to the string “never”, and extracts the attribute into the string object S.

Cross-references

See “Adding Custom Attributes in the Label View” on page 123 and “Adding Your Own Label Attributes” on page 70 of *User’s Guide I*.

setcollabels	Coef Procs

Set the column label in a coef object.

Syntax

coef_name.**setcollabels** *label*

Follow the keyword with the column label. Note that the column label should not contain spaces unless it is enclosed in quotes.

Examples

```
c1.setcollabels beta
```

sets the column label to “beta”.

Cross-references

See also Coef::setrowlabels (p. 45).

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:

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

To specify a format that groups digits into thousands using a comma separator, place a “t” after the format character. For example, to obtain a fixed number of decimal places with commas used to separate thousands, use “ft*[.precision]*”.

To use the period character to separate thousands and commas to denote decimal places, use “..” (two periods) when specifying the precision. For example, 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)”), 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. 46), `Coef::setindent` (p. 44), and `Coef::setjust` (p. 45) 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 indentation setttings are taken from the Global Defaults for spreadsheet views ("Spreadsheet Data Display" on page 1019 of *User's Guide I*) at the time the spreadsheet was 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. 46) and `Coef::setjust` (p. 45) for details on setting spreadsheet widths and justification.

setjust	Coef Procs

Set the horizontal justification for all cells in the spreadsheet view of the coef object.

Syntax

coef_name.**setjust** *format_arg*

where format_arg may be set to left, center, right, or auto (strings are left-justified and numbers are right-justified). Default display settings can be set in General Options; see "Spreadsheet Data Display" on page 1019 of *User's Guide I*.

Examples

```
c1.setjust left
```

left-justifies the cells in the spreadsheet view of the coef C1.

Cross-references

See `Coef::setwidth` (p. 46) and `Coef::setindent` (p. 44) for details on setting spreadsheet widths and indentation.

setrowlabels	Coef Procs

Set the row labels in a coef object.

Syntax

coef_name.**setrowlabels** *label1 label2 label3...*

Follow the keyword with a space delimited list of row labels. Note that each row label should not contain spaces unless it is enclosed in quotes. If you provide fewer labels than there are rows, EViews will keep the corresponding default row names ("R11", "R12", etc…).

Examples

```
c1.setrowlabels USA UK FRANCE
```

sets the row label for the first row in coef C1 to USA, the second to UK, and the third to FRANCE.

Cross-references

See also `Coef::setcollabels` (p. 42).

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. 44) and `Coef::setjust` (p. 45) 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.

showlabels	Coef Procs

Displays the custom row and column labels of a coef spreadsheet.

Syntax

coef_name.**showlabels** *mode*

where *mode* is either 0 or 1 where 0 displays the default row and column labels and 1 displays the custom row and column labels (if present).

Examples

```
c1.showlabels 1
```

displays the custom row and column labels for the C1 spreadsheet. If custom labels have not been set the default labels will be displayed.

```
c1.showlabels 0
```

displays the default row and column labels for the C1 spreadsheet.

Cross-references

See `Coef::setcollabels` (p. 42) and `Coef::setrowlabels` (p. 45).

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 450 and "Descriptive Statistics" on page 667 of *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.

This routine should realistically only be used in the oft-hand chance that you wish to write into a Lotus file. Improved Excel, text, and other format writing is available in `Coef::export` (p. 28).

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

prompt	Force the dialog to appear from within a program.

File type

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

If you omit the "*t* = " option, EViews will determine the type based on the file extension. Unrecognized extensions will be treated as ASCII files. For Lotus and Excel spreadsheet files specified without the "*t* = " option, EViews will automatically append the appropriate extension if it is not otherwise specified.

ASCII text files

na = *string*	Specify text string for NAs. Default is "NA".
d = *arg*	Specify delimiter (*default* is tab): "s" (space), "c" (comma).

Spreadsheet (Lotus, Excel) files

letter_number	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 171 of *User's Guide I* for a discussion.

See also `Coef::export` (p. 28) and `Coef::read` (p. 40).

Equation

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

Equation Declaration

equation.....declare equation object (p. 133).

To declare an equation object, enter the keyword `equation`, followed by a name:

```
equation ecoefq01
```

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. 64).

ardl.....autoregressive distributed lag models (p. 71).

binary.....binary dependent variable models (includes probit, logit, gompit) models (p. 79).

breakls..... least squares with breakpoints and breakpoint determination (p. 158).

censored .....censored and truncated regression (includes tobit) models (p. 89).

cointreg .....cointegrating regression using FMOLS, CCR, or DOLS, or panel FMOLS or DOLS (p. 102).

count .....count data modeling (includes poisson, negative binomial and quasi-maximum likelihood count models) (p. 113).

did.....estimate a panel equation using the difference-in-difference estimator (p. 118).

enet .....elastic net regression (including Lasso and ridge regression) (p. 126).

funcoef .....functional coefficients regression (p. 146).

glm.....estimate a Generalized Linear Model (GLM) (p. 153).

gmm.....estimate an equation using generalized method of moments (GMM) (p. 158).

heckit .....estimate a selection equation using the Heckman ML or 2-step method (p. 166).

liml.....estimate an equation using Limited Information Maximum Likelihood and K-class estimation (p. 179).

logit.....logit (binary) estimation (p. 181).

ls .....estimation using least squares or nonlinear least squares (p. 181).

midas ..... Mixed Data Sampling (MIDAS) regression (p. 201).
ordered ..... ordinal dependent variable models (includes ordered probit, ordered logit, and ordered extreme value models) (p. 210).
probit ..... probit (binary) estimation (p. 215).
qreg ..... estimate an equation using quantile regression (p. 218).
robustls ..... robust regression (M-estimation, S-estimation and MM-estimation) (p. 234).
switchreg ..... exogenous and Markov switching regression (p. 242).
threshold ..... threshold least squares, including threshold autoregression (p. 249).
tsls ..... estimate an equation using two-stage least squares regression (p. 254).
varsel ..... equation estimation using least squares with variable selection (uni-directional, stepwise, swapwise, combinatorial, Auto-GETS, Lasso) (p. 260).

Equation Views

abtest ..... test for serial correlation in a panel GMM equation using the Arellano-Bond test (p. 64).
archtest ..... LM test for the presence of ARCH in the residuals (p. 70).
arma ..... Examine ARMA structure of estimated equation (p. 77).
auto ..... Breusch-Godfrey serial correlation Lagrange Multiplier (LM) test (p. 78).
boundstest ..... perform the Pesaran, Shin and Smith (2001) bounds test of long-run relationships from an ARDL estimated equation (p. 81).
breakspec ..... display the breakpoint specification for an equation estimated by least squares with breakpoints (p. 85).
breaktest ..... perform breakpoint test for TSLS and GMM equations (p. 86).
cdtest ..... test for the presence of cross-sectional dependence of errors in panel equations (p. 86).
cellipse ..... confidence ellipses for coefficient restrictions (p. 87).
chow ..... Chow breakpoint and forecast tests for structural change (p. 91).
cinterval ..... confidence interval for coefficients (p. 92).
coefcov ..... coefficient covariance matrix (p. 94).
coeflabel ..... display coefficients associated with variables in the equation (p. 94).
coefmatrix ..... display matrix of lambda and coefficients for elastic net, ridge, and Lasso models (p. 95).

rls ..... recursive residuals least squares (only for non-panel equations estimated by ordinary least squares, without ARMA terms) (p. 233).
signbias ..... perform the Sign bias test (Engle and Ng, 1993) of misspecification in equations estimated using GARCH (p. 238).
similarity ..... compute Hausman tests for Pooled mean group ARDL equations (p. 239).
srcoefs ..... displays a spool object with the results of error-correction regressions for each cross-section in PMG estimation (p. 240).
strconstant ..... tests for constancy of the base specification coefficients against a smoothly varying alternative in a smooth threshold regression (p. 240).
strlinear ..... compute tests for linearity of the base specification against the smooth threshold alternative in a smooth threshold regression (p. 241).
strnonlin ..... compute various tests for additional additive or encapsulated nonlinearity in a smooth threshold regression (p. 241).
strwgts ..... compute and display the transition weights in a smooth threshold regression (p. 242).
symmtest ..... compute symmetry test for nonlinear distributed lag variables in nonlinear ARDL models (p. 246).
testadd ..... likelihood ratio test for adding variables to equation (p. 246).
testdrop ..... likelihood ratio test for dropping variables from equation (p. 247).
testfit ..... performs Hosmer and Lemeshow and Andrews goodness-of-fit tests (only for equations estimated using binary) (p. 248).
transprobs ..... display the state transition probabilities in a switching regression equation (p. 252).
ubreak ..... Andrews-Quandt test for unknown breakpoint (p. 258).
varinf ..... display Variance Inflation Factors (VIFs) (p. 260).
wald ..... Wald test for coefficient restrictions (p. 267).
weakinst ..... display the weak instruments summary (p. 268).
white ..... White test for heteroskedasticity (p. 268).

Equation Procs

clearhist ..... clear the contents of the history attribute (p. 93).
clearremarks ..... clear the contents of the remarks attribute (p. 93).
copy ..... creates a copy of the equation (p. 111).
didmakeeq ..... create an equation object with the underlying fixed-effects estimation of a difference-in-difference equation (p. 120).
displayname ..... set display name (p. 122).
fit ..... static forecast (p. 134).

Equation Data Members

Scalar Values

@aic .....Akaike information criterion.
@bylist .....returns 1 or 0 depending on whether the equation was estimated by list.
@coefcov(i,j) .....covariance of coefficient estimates i and j.
@coefs(i) .....i-th coefficient value.
@deviance .....deviance (for Generalized Linear Models).
@deviancestat.....deviance statistic: deviance divided by degrees-of-freedom (for Generalized Linear Models).
@df .....degrees-of-freedom for equation.
@dispersion.....estimate of dispersion (for Generalized Linear Models).
@dw.....Durbin-Watson statistic.

@f ..... F-statistic.

@finalbw ..... returns the final bandwidth used in functional coefficient estimation.

@fixeddisp ..... indicator for whether the dispersion is a fixed value (for Generalized Linear Models).

@fprob..... probability value of the F-statistic.

@hacbw..... bandwidth for HAC estimation of GMM weighting matrix or long-run covariance in cointegrating regression (if applicable).

@hq..... Hannan-Quinn information criterion.

@instrank ..... rank of instruments (if applicable).

@jstat ..... J-statistic — value of the GMM objective function (for GMM and TSLS).

@jprob..... probability value of the J-statistic.

@lambdamin..... minimum lambda value from ENET cross-validation.

@limlk..... estimate of LIML k (if applicable).

@logl ..... value of the log likelihood function.

@lrprob ..... probability value of likelihood ratio statistic (if applicable).

@lrstat ..... likelihood ratio statistic (if applicable).

@lrvar ..... long-run variance estimate for cointegrating regression (if applicable).

@meandep ..... mean of the dependent variable.

@nbreaks..... number of breaks in breakpoint least squares and thresholds in threshold regression.

@ncases..... number of cases.

@nclusters ..... number of clusters used in computing cluster robust covariances.

@ncoef ..... number of estimated coefficients.

@ncross..... number of cross-sections used in estimation (equal to 1 for non-panel workfiles).

@npers ..... number of workfile periods used in estimation (same as `@regobs` for non-panel workfiles).

@nregimes..... number of regimes in a switching and breakpoint regression.

@nthresholds..... number of thresholds in threshold regression.

@ntreatment ..... difference-in-difference number of cross sections receiving treatment.

@objective ..... quasi-likelihood objective function (if applicable).

@pearsonssr..... Pearson sum-of-squared residuals (for Generalized Linear Models).

@pearsonstat..... Pearson statistic: Pearson SSR divided by degrees-of-freedom (for Generalized Linear Models).

@pilotbw.....returns the pilot bandwidth used in functional coefficient estimation.
@qlrprob .....probability value of quasi-likelihood ratio statistic (if applicable).
@qlrstat.....quasi-likelihood ratio statistic (if applicable).
@quantdep .....quantile of dependent variable (for quantile regression).
@r2 .....R-squared statistic.
@rbar2 .....adjusted R-squared statistic.
@rdeviance.....restricted (constant only) deviance (for Generalized Linear Models).
@regobs .....number of observations in regression.
@rlogl .....restricted (constant only) log-likelihood (if applicable).
@rmse.....root MSE.
@rn2 .....Rn-squared statistic.
@robf .....robust F-statistic (Wald-test form).
@robfprob .....robust F-statistic (Wald-test form) p-value.
@robjective.....restricted (constant only) quasi-likelihood objective function (if applicable).
@rw2 .....Rw-squared.
@schwarz .....Schwarz information criterion.
@sddep.....standard deviation of the dependent variable.
@se .....standard error of the regression.
@sparsity.....estimate of sparsity (for quantile regression).
@ssr .....sum of squared residuals.
@ssr2 .....second-stage SSR.
@stderrs(i) .....standard error for coefficient i.
@thresholds.....number of thresholds (for threshold regression).
@tstats(i).....t-statistic or z-statistic value for coefficient i.
@wmeandep.....weighted mean of dependent variable (if applicable).
@wgtscale .....scaling factor for weights (if applicable).
c(i) .....i-th element of default coefficient vector for equation (if applicable).

Vectors and Matrices

@ardlceccoefs.....returns a vector of coefficient estimates from the conditional error-correction (CEC) regression in univariate (N)ARDL estimation.
@ardleccoefs.....returns a vector of coefficient estimates from the traditional error-correction (EC) regression in univariate (N)ARDL estimation.
@ardlcoint.....returns a coef containing coefficients from the cointegrating relationship form of an ARDL estimation.
@ardllrcoefs.....returns a coef containing coefficients from the long run relationship form of a non-panel ARDL estimation.

@ardlsrcoefs ..... returns a matrix where each row corresponds to an individual cross-section's short-run coefficients. Only applicable for PMG/ARDL estimation.

@ardlsrses ..... returns a matrix where each row corresponds to an individual cross-section's short-run coefficient standard errors. Only applicable for PMG/ARDL estimation.

@coefcov ..... covariance matrix for coefficient estimates.

@coefs ..... coefficient vector.

@contempcov ..... symmetric matrix containing the contemporaneous covariance Σ for cointegrating regression residuals estimated with CCR.

@cvconverge..... Elastic net path cross-validation convergence test values (lambda values in rows; lambda in first column, training-test sample results in remaining columns).

@cvisvalid ..... Elastic net path cross-validation valid results indicators (lambda values in rows; lambda in first column, training-test sample results in remaining columns).

@cviters..... Elastic net path cross-validation iterations (lambda values in rows; lambda in first column, training-test sample results in columns).

@cvobjective..... Elastic net path cross-validation objective values (lambda values in rows; lambda in first column, training-test sample results in remaining columns).

@effects ..... vector of fixed and random effects estimates (if applicable).

@fcgrid ..... returns a vector of unique grid values over which functional coefficients are evaluated in functional coefficient estimation.

@initprobs ..... matrix containing initial probabilities for switching regression equations.

@instwgt ..... symmetric matrix containing the final sample instrument weighting matrix used during GMM or TSLS estimation (*e.g.*, $\hat{s}^2(Z'Z)$ for 2SLS and $\sum \hat{\epsilon}_t^2 Z_t Z_t'$ for White weighting).

@lambdacoefs..... Elastic net lambda path coefficients matrix (lambda values in rows; variables in columns). Full set of variables including those with zero coefficients along the path.

@lambdaest ..... Elastic net lambda path estimation measures matrix (lambda values in rows; columns contain the lambda values, number of non-zero coefficients, estimation objective, sums-of-squares portion of the objective, L_1 portion of the objective, L_2^2 portion of the objective).

@lambdafit ..... Elastic net lambda path fit measures matrix (lambda values in rows; columns contain the lambda values, number of non-zero coefficients, R-squared, adjusted R-squared, and sums-of-squared residuals).

@lambdapath.....Elastic net lambda path vector.

@lambda2cov .....symmetric matrix Λ_2 containing the long run covariances of u_2 with u_1 and u_2 for cointegrating regression equations estimated with CCR and FMOLS.

@lrcov.....symmetric matrix containing the long-run covariance Ω for cointegrating regression equations estimated with CCR and FMOLS.

@modselresults.....Elastic net path model selection summary (lambda values in rows; lambda in first column, followed by model selection objective, number of non-zero coefficients, and the fit statistics (sum-of-squared residuals, mean-square error, R-squared, and adjusted R-squared) associated with the estimated model.

@pmgcxcoefs.....returns a matrix of coefficient estimates from the error-correction regressions for each cross-section in PMG estimation. Each column corresponds to a single cross-section and each column corresponds to a coefficient estimate from the traditional error-correction regression, in order of appearance.

@pmgcxses.....returns a matrix of coefficient standard error estimates from the error-correction regressions for each cross-section in PMG estimation. Each column corresponds to a single cross-section and each column corresponds to a coefficient standard error estimate from the traditional error-correction regression, in order of appearance.

@pmglrcoefs.....returns a vector of long-run (pooled) coefficient estimates in PMG estimation.

@pmgsrcoefs .....returns a vector of short-run (mean-group) coefficient estimates in PMG estimation.

@pvals .....vector containing the coefficient probability values.

@stderrs .....vector of standard errors for coefficients.

@thresholds.....vector of threshold values for threshold estimation.

@tstats .....vector of *t*-statistic or *z*-statistic values for coefficients.

String Values

@ardlcointsubst ....returns string representation of the cointegration form of an ARDL equation with substituted coefficients.

@attr("*arg*").....string containing the value of the *arg* attribute, where the argument is specified as a quoted string.

@breaks .....string representation of the breakpoints in breakpoint least squares or thresholds in threshold regression.

@coeflabels.....coefficient labels used in regression output table.

@coeflist ..... returns a string containing a space delimited list of the coefficients used in estimation (*e.g.*, "C(1) C(2) C(3)"). This function always returns the list of actual coefficients used, irrespective of whether the original equation was specified by list or by expression.

@command..... full command line form of the estimation command. Note this is a combination of `@method`, `@options` and `@spec`.

@depends ..... string containing a list of the series in the current workfile on which this equation depends.

@description..... string containing the Equation object's description (if available).

@detailedtype ..... returns a string with the object type: "EQUATION".

@displayname..... returns the equation's display name. If the equation has no display name set, the name is returned.

@esteq..... returns string representation of the estimation equation.

@extralist..... space delimited list of the equation's extra regressors. For equation's estimated by ARCH, `@extralist` contains the variance equation terms. For equations estimated by CENSORED, this contains the error distribution terms. For all other equation methods it returns an empty string.

@instlist..... space delimited list of the equation instruments (if applicable).

@method ..... command line form of estimation method ("ARCH", "LS", *etc.*...).

@name ..... returns the name of the Equation.

@options..... command line form of estimation options.

@remarks ..... string containing the equation object's remarks (if available).

@smpl ..... description of the sample specified for estimation.

@spec..... original equation specification. Note this will be different from `@varlist` if the equation specification contains groups, or is specified by expression.

@subst..... returns string representation of the equation with substituted coefficients.

@type ..... returns a string with the object type: "EQUATION".

@updatetime..... returns a string representation of the time and date at which the equation was last updated.

@varlist ..... space delimited list of the equation's dependent variable and regressors if the equation was specified by list, or the equation's underlying variables (both dependent and independent) if the equation was specified by expression.

@varselkept ..... space delimited list of variables kept by model selection.

@varselrejected.... space delimited list of the variables dropped by model selection.

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.

abtest	Equation Views

Test for serial correlation in a panel GMM equation using the Arellano-Bond test.

Tests for first and second order autocorrelation amongst the residuals of an equation estimated by GMM with first differences in a panel workfile. If the underlying errors are *i.i.d*, we would expect the first differences to be negatively first order serially correlated, and not display second order correlation.

Syntax

eq_name.**abtest***(options)*

Options

p	Print output from the test.

Examples

```
equation eq1.gmm(cx=fd, per=f, gmm=perwhite, iter=oneb, levelper)
   n n(-1) n(-2) w w(-1) k ys ys(-1) @ @dyn(n,-2) w w(-1) k ys ys(-
   1)
eq1.abtest
```

estimates an equation using GMM with first difference fixed effects, and then tests for first and second order autocorrelation.

Cross-references

See "Arellano-Bond Serial Correlation Testing" on page 1370 of *User's Guide II* for discussion.

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 first two options specify the order of the GARCH model:

- The **arch** estimation method specifies a GARCH(p, q) model with p ARCH terms and q GARCH terms. *Note the order of the arguments in which the ARCH and GARCH terms are entered.*

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.

- For CGARCH, FIEGARCH and MIDAS-GARCH models, EViews only estimates (1,1) models. For these specifications, p and q options should not be specified, and if provided, will be ignored.

After the "ARCH" keyword and options, specify the dependent variable followed by a list of regressors in the mean equation.

- By default, only the intercept is included in the conditional variance equation. If you wish to specify 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 both 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.
- For MIDAS-GARCH models, the low-frequency permanent component regressor are entered after the mean equation regressors and an "@"-sign. The regressor should be specified as *pagename\seriesname*.

Options

Type Options

The default is to estimate a standard GARCH model. You may specify one of the followings keywords to estimate a different model:

egarch	Exponential GARCH.
parch*[= arg]*	Power ARCH. If the optional *arg* is provided, the power parameter will be set to that value, otherwise the power parameter will be estimated.
cgarch	Component (permanent and transitory) ARCH.
figarch	Fractional GARCH (FIGARCH).
fiegarch	Fractional Exponential GARCH (FIEGARCH(1,1)).
midas	MIDAS GARCH(1,1)

General Options

thrsh	For Component GARCH models, include a threshold term.
thrsh = *integer* *(default* = 0*)*	Number of threshold terms for GARCH models. The maximum number of terms allowed is 9.
vt	Variance target of the constant term for GARCH models. (May not be used with integrated specifications.)
integrated	Restrict GARCH model to be integrated, *i.e.* IGARCH. (May not be used with variance targeting.)
asy = *integer* *(default* = 1*)*	Number of asymmetric terms in Power ARCH or EGARCH models. The maximum number of terms allowed is 9.
trunclag = *integer* *(default* = 1000*)*	Number of terms in the expansion approximation for FIGARCH and FIEGARCH models.
archm = *arg*	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.
tdist *[= number]*	Estimate the model assuming that the residuals follow a conditional Student's *t*-distribution (the default is the conditional normal distribution). Providing the optional number greater than two will fix the degrees of freedom to that value. If the argument is not provided, the degrees of freedom will be estimated.
ged *[= number]*	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.
z	Turn of backcasting for both initial MA innovations and initial variances.
backcast = *n*	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.
optmethod = *arg*	Optimization method: "bfgs" (BFGS); "newton" (Newton-Raphson), "opg" or "bhhh" (OPG or BHHH), "legacy" (EViews legacy). "bfgs" is the default for new equations.

optstep = *arg*	Step method: "marquardt" (Marquardt - default); "dogleg" (Dogleg); "linesearch" (Line search). (Applicable when "optmethod = bfgs", "optmethod = newton" or "optmethod = opg".)
b	Use Berndt-Hall-Hall-Hausman (BHHH) as maximization algorithm. The default is Marquardt. (Applicable when "optmethod = legacy".)
cov = *arg*	Covariance method: "ordinary" (default method based on inverse of the estimated information matrix), "huber" or "white" (Huber-White sandwich method), "bollerslev" (Bollerslev-Wooldridge method).
covinfo = *arg*	Information matrix method: "opg" (OPG); "hessian" (observed Hessian), " (Applicable when non-legacy "optmethod = " with "cov = ordinary".)
h	Bollerslev-Wooldridge robust quasi-maximum likelihood (QML) covariance/standard errors. (Applicable for "optmethod = legacy" when estimating assuming normal errors.)
m = *integer*	Set maximum number of iterations.
c = *scalar*	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.
s	Use the current coefficient values in "C" as starting values (see also `param` (p. 564) of the *Command and Programming Reference*).
s = *number*	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*").
numericderiv / -numericderiv	[Do / do not] use numeric derivatives only. If omitted, EViews will follow the global default.
fastderiv / -fastderiv	[Do / do not] use fast derivative computation. If omitted, EViews will follow the global default. Available only for legacy estimation ("optmeth = legacy").
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.

coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
prompt	Force the dialog to appear from within a program.
p	Print estimation results.

MIDAS Options

lag = *arg*	Specify the number of lags of the low frequency regressor to include. Default value is 32.
beta = *arg*	Beta function restriction: none ("none"), trend coefficient equals 1 ("trend"), endpoints coefficient equals 0 ("endpoint"), both trend and endpoints restriction ("both"). For use when "midwgt = beta". The default is "beta = none".
thrsh	Include a threshold term.
optmethod = *arg*	Optimization method for nonlinear estimation: "bfgs" (BFGS); "newton" Newton-Raphson), "opg", "bhhh" (OPG or BHHH), or "hybrid" (initial BHHH followed by BFGS). Hybrid is the default method.
optstep = *arg*	Step method for nonlinear estimation: "marquardt" (Marquardt); "dogleg" (Dogleg); "linesearch" (Line search). Marquardt is the default method.
cov = *arg*	Covariance method for nonlinear models: "ordinary" (default method based on inverse of the estimated information matrix), "huber" or "white" (Huber-White sandwich).
covinfo = *arg*	Information matrix method for nonlinear models: "opg" (OPG); "hessian" (observed Hessian).
nodf	Do not perform degree of freedom corrections in computing coefficient covariance matrix. The default is to use degree of freedom corrections.
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 estimator coefficient vector as starting values in nonlinear estimation. If the "s = number" or "s" options are not used, EViews will use random starting values.

s = *number*	Determine starting values for nonlinear estimation. Specify a number between zero and oSpecify the number of lags of the low frequency regressor to include. Default value is 32.ne representing the fraction of preliminary EViews chosen values. Note that out of range values are set to "s = 1". Specifying "s = 0" initializes coefficients to zero. If the "s = number" or "s" options are not used, EViews will use random starting values.
seed = *positive integer* from 0 to 2,147,483,647	Seed the random number generator used in random starting values. If not specified, EViews will seed random number generator with a single integer draw from the default global random number generator.
showopts/-showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
coef = *arg*	Specify the name of the coefficient vector; the default behavior is to use the "C" coefficient vector.
prompt	Force the dialog to appear from within a program.
p	Print estimation results.

Saved results

Most of the results saved for the `ls` estimation method are also available after ARCH estimation; see `Equation::ls` (p. 181) for details.

Examples

```
equation arc1.arch(4, 0, m=1000, cov=bollerslev) 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 arc1.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 arc1.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.

```
arc1.makegarch garch1
```

Cross-references

See Chapter 27. "ARCH and GARCH Estimation," on page 273 of *User's Guide II* for a discussion of ARCH models. See also `Equation::garch` (p. 153) and `Equation::makegarch` (p. 193).

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:

prompt	Force the dialog to appear from within a program.
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` as a view off of an estimated `arch` equation, EViews will use 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 226 of *User's Guide II* for further discussion of testing ARCH and Chapter 27. "ARCH and GARCH Estimation," on page 273 of *User's Guide II* for a general discussion of working with ARCH models in EViews.

See also `Equation::hettest` (p. 168) for a more full-featured version of this test.

ardl	Equation Methods

Estimate an equation with autoregressive distributed lags using linear and nonlinear least squares or quantile regression.

Syntax

equation.**ardl**(*options*) *linear_regs [@ static_regs] [@***asy** *dual_asymmetric_regs] [@***asylr** *long_run_asymmetric_regs] [@***asysr** *short_run_asymmetric_regs]*

The *linear_regs* specification is required:

- The *linear_regs* list should be the dependent variable followed by a list of linear distributed-lag regressors.

The remaining specifications are optional

- *static_regs* should be a list of static regressors, not including a constant or trend term.
- *dual_asymmetric_regs* are distributed-lag regressors which are asymmetric both in the short-run and long-run.
- *long_run_asymmetric_regs* regressors are distributed lag-regressors which are asymmetric in the long-run but symmetric in the short-run.
- *short_run_asymmetric_regs* are asymmetric regressors which are distributed lag-regressors which are asymmetric in the short-run but symmetric in the long-run.

You may specify the lag for an individual distributed-lag variable using the "`@fl(variable, lag)`" syntax. For instance, if the variable X should use 3 lags, irrespective of the fixed or automatic lag settings, you may specify this by entering "@fl(x, 3)" in the regressor list.

Options

Least Squares ARDL Options

method = *arg* *(default = "ls")*	Set the method of estimation: "ls" (least-squares regression, default) or "qreg" (quantile regression).
determ = *arg* *(default = "rconst")*	Johansen deterministic trend type: "none" (no deterministics), "rconst" (restricted constant and no trend), "uconst" (unrestricted constant and no trend), "rtrend" (unrestricted constant and restricted trend, "utrend" (unrestricted constant and unrestricted trend).
trend = *arg* *(deprecated)*	Johansen deterministic trend type: "none" (no deterministics), "const" (restricted constant and no trend, *default*), "uconst" (unrestricted constant and no trend), "linear" (unrestricted constant and restricted trend, "ulinear" (unrestricted constant and unrestricted trend). Note: this is a *deprecated* s option which handles a subset of cases covered by the "determ = " option
fixed	Do not use automatic selection for lag lengths. This option must be used with the "deplags = " and "reglags = " options.
deplags = *int* *(default = 4)*	Set the number of lags for the dependent variable to *int*. If automatic selection is used, this sets the maximum number of possible lags. If fixed lags are used (the *fixed* option is set), this fixes the number of lags.
reglags = *int (default = 4)*	Set the number of lags for the explanatory variables (dynamic regressors) to *int*. If automatic selection is used, this sets the maximum number of possible lags. If fixed lags are used (the *fixed* option is set), this fixes the number of lags for each regressor.
ic = *key* *(default = "aic")*	Set the method of automatic model selection. *key* may take values of "aic" (Akaike information criterion, *default*), "bic" (Schwarz criterion), "hq" (Hannan-Quinn criterion) or "rbar2" (Adjusted R-squared, not applicable in panel workfiles).
cov = *arg*	Covariance method: "ordinary" (default method based on inverse of the estimated information matrix), "huber" or "white" (Huber-White sandwich method), "hac" (Newey-West HAC, available for nonlinear least squares or ARMA estimated by CLS)..
nodf	Do not perform degree of freedom corrections in computing coefficient covariance matrix. The default is to use degree of freedom corrections.

covlag = *arg* *(default* = 1*)*	Whitening lag specification: *integer* (user-specified lag value), "a" (automatic selection).
covinfosel = *arg* *(default* = "aic"*)*	Information criterion for automatic selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn) (if "lag = a").
covmaxlag = *integer*	Maximum lag-length for automatic selection *(optional)* (if "lag = a"). The default is an observation-based maximum of $T^{1/3}$.
covkern = *arg* *(default* = "bart"*)*	Kernel shape: "none" (no kernel), "bart" (Bartlett, *default*), "bohman" (Bohman), "daniell" (Daniel), "parzen" (Parzen), "parzriesz" (Parzen-Riesz), "parzgeo" (Parzen-Geometric), "parzcauchy" (Parzen-Cauchy), "quadspec" (Quadratic Spectral), "trunc" (Truncated), "thamm" (Tukey-Hamming), "thann" (Tukey-Hanning), "tparz" (Tukey-Parzen).
covbw = *arg* *(default* = "fixednw" *)*	Kernel Bandwidth: "fixednw" (Newey-West fixed), "andrews" (Andrews automatic), "neweywest" (Newey-West automatic), *number* (User-specified bandwidth).
covnwlag = *integer*	Newey-West lag-selection parameter for use in nonparametric kernel bandwidth selection (if "covbw = neweywest").
covbwint	Use integer portion of bandwidth.
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
prompt	Force the dialog to appear from within a program.
p	Print results.

Quantile ARDL Options

quant = *number* *(default* = 0.5*)*	Quantile to be fit (where *number* is a value between 0 and 1).
w = *arg*	Weight series or expression. Note: *we recommend that, absent a good reason, you employ the default settings* **Inverse std. dev. weights** *("wtype = istdev") with* **EViews default** *scaling ("wscale = eviews") for backward compatibility with versions prior to EViews 7.*
wtype = *arg* *(default* = "istdev"*)*	Weight specification type: inverse standard deviation ("istdev"), inverse variance ("ivar"), standard deviation ("stdev"), variance ("var").

wscale = *arg*	Weight scaling: EViews default ("eviews"), average ("avg"), none ("none"). The default setting depends upon the weight type: "eviews" if "wtype = istdev", "avg" for all others.
cov = *arg* (*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 = *arg* (*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 = *number*	Use user-specified bandwidth value in place of automatic method specified in "bwmethod = ".
bwsize = *number* (*default* = 0.05)	Size parameter for use in computation of bandwidth (used when "bw = hs" and "bw = bf").
spmethod = *arg* (*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 = *arg* (*default* = "pair")	Bootstrap method: "resid" (residual bootstrap), "pair" (xy-pair bootstrap), "mcmb" (MCMB bootstrap), "mcmba" (MCMB-A bootstrap).
btreps = *integer* (*default* = 100)	Number of bootstrap repetitions
btseed = *positive integer*	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.
btrnd = *arg* (*default* = "kn" or method previously set using `rndseed` (p. 577) in the *Command and Programming Reference*).	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").

btobs = *integer*	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.
btout = *name*	*(optional)* Matrix to hold results of bootstrap simulations.
k = *arg* *(default* = "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).
m = *integer*	Maximum number of iterations.
s	Use the current coefficient values in estimator coefficient vector as starting values (see also `param` (p. 564) in the *Command and Programming Reference*).
s = *number (default* = 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.
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
prompt	Force the dialog to appear from within a program.
p	Print estimation results.

Examples

```
wfopen http://www.stern.nyu.edu/~wgreene/Text/Edition7/TableF5-
    2.txt
```

opens example data from Greene (2008, page 685), containing quarterly US macroeconomic variables between 1950 and 2000.

The following command

```
equation eq01.ardl(deplags=8, reglags=8) log(realcons)
    log(realgdp) @ @expand(@quarter, @droplast)
```

creates an equation object and estimates an ARDL model with the log of real consumption as the dependent variable, and the log of real GDP as a dynamic regressor. Quarterly dummy variables are included as static regressors. Automatic model selection is used to determine the number of lags of LOG(REALCONS) and LOG(REALGDP).

The command

```
equation eq02.ardl(deplags=3, reglags=3, fixed) log(realcons)
    log(realgdp) @ @expand(@quarter, @droplast)
```

estimates a second model, replicating Example 20.4 from Greene, with a fixed three lags of the dependent variable and three lags of the regressor.

```
equation eq03.ardl(deplags=1, reglags=1, fixed) log(realcons)
    log(realgdp) @asy log(realgovt)
```

The line above estimates an ARDL(1,1,1) model with the log of real consumption as the dependent variable, the log of real GDP as a linear regressor, and log of real government expenditures as a dual asymmetric regressor.

```
equation eq04.ardl(deplags=1, reglags=1, fixed) log(realcons)
    log(realgdp) @asy log(realgovt) @asysr log(realinvs)
```

extends the previous model and estimates an ARDL(1,1,1,1) model by including the log of real investments as a long-run asymmetric regressor.

```
equation eq05.ardl(deplags=1, reglags=1, fixed) log(realcons)
    log(realgdp) @asy log(realgovt) @asysr log(realinvs) @asylr
    log(tbilrate)
```

The line above extends the previous model and estimates an ARDL(1,1,1,1,1) model by including the log of treasury bill rates as a short-run asymmetric regressor.

```
wfopen oecd.wf1
equation eq06.ardl(fixed, deplags=1, reglags=1) log(cons) log(inf)
    log(inc)
```

This example estimates a panel ARDL model using the workfile "OECD.wf1". This model replicates that given in the original Pesaran, Shin and Smith 1999 paper. Model selection is not used to choose the optimal lag lengths, rather a fixed single lag of both the dependent variable and the regressor is employed.

```
equation eq07.ardl(method=qreg, ls=fixed, deplags=1, reglags=1,
    quant=0.4) log(realcons) log(realgdp)
```

This command estimates a QARDL(1,1) model where lag selection is fixed for both the dependent and independent regressors, and the quantile value is 0.4.

Cross-references

See Chapter 29. "ARDL and Quantile ARDL" of *User's Guide II* for further discussion.

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 four views types available: roots, correlogram, impulse response, and frequency spectrum.

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

type = *arg*	Required "type = " option selects the type of ARMA structure output: "root" displays the inverse roots of the AR/MA characteristic polynomials, "acf" displays the second moments (autocorrelation and partial autocorrelation) for the data in the estimation sample and for the estimated model, "imp" displays the impulse responses., "freq" displays the frequency spectrum.
t	Displays the table view of the results for the view specified by the "type = " option. By default, EViews will display a graphical view of the ARMA results.
hrz = *arg*	Specifies the maximum lag length for "type = acf", and the maximum horizon (periods) for "type = imp".
imp = *arg*	Specifies the size of the impulse for the impulse response ("type = imp") view. By default, EViews will use the regression estimated standard error.
save = *arg*	Stores the results as a matrix object with the specified name. The matrix holds the results roughly as displayed in the table view of the corresponding type. For "type = root", roots for the AR and MA polynomials will be stored in separate matrices as NAME_AR and NAME_MA, where "NAME" is the name given by the "save = " option.
prompt	Force the dialog to appear from within a program.
p	Print the table or graph output.

Examples

```
eq1.arma(type=root, save=root)
```

displays and saves the ARMA roots from the estimated equation EQ1. The roots will be placed in the matrix object ROOT.

```
eq1.arma(type=acf, hrz=25, save=acf)
```

computes the second moments (autocorrelation and 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.

```
eq1.arma(type=freq)
```

displays the frequency spectrum in graph form.

Cross-references

See "ARMA Structure" on page 150 of *User's Guide II* for details. See also Chapter 24. "Time Series Regression," on page 121 of *User's Guide II*.

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

prompt	Force the dialog to appear from within a program.
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 130 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").
optmethod = *arg*	Optimization method: "bfgs" (BFGS); "newton" (Newton-Raphson), "opg" or "bhhh" (OPG or BHHH), "legacy" (EViews legacy). Newton-Raphson is the default method.
optstep = *arg*	Step method: "marquardt" (Marquardt); "dogleg" (Dog-leg); "linesearch" (Line search). Marquardt is the default method.

cov = *arg*	Covariance method: "ordinary" (default method based on inverse of the estimated information matrix), "huber" or "white" (Huber-White sandwich method), "glm" (GLM method), "cr" (cluster robust).
covinfo = *arg*	Information matrix method: "opg" (OPG); "hessian" (observed Hessian - default). (Applicable when non-legacy "optmethod = ".)
df	Degree-of-freedom correct the coefficient covariance estimate.(For non-cluster robust methods estimated using non-legacy estimation).
h	Huber-White quasi-maximum likelihood (QML) standard errors and covariances. (Legacy option applicable when "optmethod = legacy").
g	GLM standard errors and covariances. (Legacy option applicable when "optmethod = legacy").
crtype = *arg (default "cr1")*	Cluster robust weighting method: "cr0" (no finite sample correction), "cr1" (finite sample correction), when "cov = cr".
crname = *arg*	Cluster robust series name, when "cov = cr".
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 (see also `param` (p. 564) of the *Command and Programming Reference*).
s = *number*	Specify a number between zero and one to determine starting values as a fraction of EViews default values (out of range values are set to "*s* = 1").
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
prompt	Force the dialog to appear from within a program.
p	Print results.

Examples

To estimate a logit model of Y using a constant, WAGE, EDU, and KIDS, and computing Huber-White standard errors, you may use the command:

```
equation eq1.binary(d=l,cov=huber) 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 425 of *User's Guide II* for additional discussion.

boundstest	Equation Views

Perform the Pesaran, Shin and Smith (2001) bounds test of long-run relationships from an ARDL estimated equation.

This view displays a spool object with the ARDL bounds test diagnostics. The first table is a summary of the test along with statistic values. The second table summarizes the bound test critical values associated with the F-statistic. When appropriate (the deterministic case does not include a restricted constant (cases 3 and 5), a third table summarizes the bound test critical values associated with the t-statistic.

Syntax

eq_name.**boundstest**(*options*)

Options

p	Print output.

Examples

```
wfopen http://www.stern.nyu.edu/~wgreene/Text/Edition7/TableF5-
    2.txt
```

```
equation eq02.ardl(deplags=3, reglags=3, fixed) log(realcons)
    log(realgdp) @ @expand(@quarter, @droplast)
show eq02.boundstest
```

This example uses data from Greene (2008, page 685), containing quarterly US macroeconomic variables between 1950 and 2000. The first line of this example downloads the data set, the second line creates an equation object and estimates an ARDL model with the log of real consumption as the dependent variable. Three lags of the dependent variable, and three lags of the log of real GDP are used as dynamic regressors. Quarterly dummy variables are included as static regressors.

The final line performs the Pesaran, Shin and Smith (2001) bounds test to test for a long-run relationship between the log of real consumption and the log of real GDP.

Cross-references

See Chapter 29. “ARDL and Quantile ARDL,” on page 341 of *User’s Guide II* for further discussion.

breakls	Equation Methods

Estimation by linear least squares regression with breakpoints.

Syntax

eq_name.**breakls***(options) y z1 [z2 z3 ...] [***@nv** *x1 x2 x3 ...]*

List the dependent variable first, followed by a list of the independent variables that have coefficients which are allowed to vary across breaks, followed optionally by the keyword **@nv** and a list of non-varying coefficient variables.

Options

Breakpoint Options

method = *arg* (*default* = "seqplus1")	Breakpoint selection method: "seqplus1" (sequential tests of single $l+1$ versus l breaks), "seqall" (sequential test of all possible $l+1$ versus l breaks), "glob" (tests of global l vs. no breaks), "globplus1" (tests of $l+1$ versus l globally determined breaks), "globinfo" (information criteria evaluation),"user" (user-specified break dates).
select = *arg*	Sub-method setting (options depend on "method = "). (1) if "method = glob": Sequential ("seq") (default), Highest significant ("high"), $UDmax$ ("udmax"), $WDmax$ ("wdmax"). (2) if "method = globinfo": Schwarz criterion ("bic" or "sic") (default), Liu-Wu-Zidek criterion ("lwz").
trim = *arg* (*default* = 5)	Trimming percentage for determining minimum segment size (5, 10, 15, 20, 25).
maxbreaks = *integer* (*default* = 5)	Maximum number of breakpoints to allow (not applicable if "method = seqall").
maxlevels = *integer* (*default* = 5)	Maximum number of break levels to consider in sequential testing (applicable when "method = sequall").
breaks = "*arg*"	User-specified break dates entered in double quotes. For use when "method = user".
size = *arg* (*default* = 5)	Test sizes for use in sequential determination and final test evaluation (10, 5, 2.5, 1) corresponding to 0.10, 0.05, 0.025, 0.01, respectively
heterr	Assume regimes specific error distributions in variance computation.
commondata	Assume a common distribution for the data across segments (only applicable if original equation is estimated with a robust covariance method, "heterr" is not specified).

General Options

w = *arg*	Weight series or expression.
wtype = *arg* *(default* = "istdev"*)*	Weight specification type: inverse standard deviation ("istdev"), inverse variance ("ivar"), standard deviation ("stdev"), variance ("var").
wscale = *arg*	Weight scaling: EViews default ("eviews"), average ("avg"), none ("none"). The default setting depends upon the weight type: "eviews" if "wtype = istdev", "avg" for all others.
cov = *keyword*	Covariance type *(optional)*: "white" (White diagonal matrix), "hac" (Newey-West HAC).
nodf	Do not perform degree of freedom corrections in computing coefficient covariance matrix. The default is to use degree of freedom corrections.
covlag = *arg* *(default* = 1*)*	Whitening lag specification: *integer* (user-specified lag value), "a" (automatic selection).
covinfosel = *arg* *(default* = "aic"*)*	Information criterion for automatic selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn) (if "lag = a").
covmaxlag = *integer*	Maximum lag-length for automatic selection *(optional)* (if "lag = a"). The default is an observation-based maximum of $T^{1/3}$.
covkern = *arg* *(default* = "bart"*)*	Kernel shape: "none" (no kernel), "bart" (Bartlett, *default*), "bohman" (Bohman), "daniell" (Daniel), "parzen" (Parzen), "parzriesz" (Parzen-Riesz), "parzgeo" (Parzen-Geometric), "parzcauchy" (Parzen-Cauchy), "quadspec" (Quadratic Spectral), "trunc" (Truncated), "thamm" (Tukey-Hamming), "thann" (Tukey-Hanning), "tparz" (Tukey-Parzen).
covbw = *arg* *(default* = "fixednw" *)*	Kernel Bandwidth: "fixednw" (Newey-West fixed), "andrews" (Andrews automatic), "neweywest" (Newey-West automatic), *number* (User-specified bandwidth).
covnwlag = *integer*	Newey-West lag-selection parameter for use in nonparametric kernel bandwidth selection (if "covbw = neweywest").
covbwoffset = *integer (default* = 0*)*	Apply integer offset to bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").
covbwint	Use integer portion of bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").

coef = *arg*	Specify the name of the coefficient vector; the default behavior is to use the "C" coefficient vector.
prompt	Force the dialog to appear from within a program.
p	Print basic estimation results.

Examples

```
equation eq1.breakls m1 c unemp
```

uses the Bai-Perron sequential $L+1$ versus L tests to determine the optimal breaks in a model regressing M1 on the breaking variables C and UNEMP.

```
equation eq2.breakls(method=glob, select=high) m1 c unemp
```

uses the global Bai-Perron L versus none tests to determine the breaks. The selected break will be the highest significant number of breaks.

```
equation eq3.breakls(size=5, trim=10) m1 c unemp
```

lowers the sequential test size from 0.10 to 0.05, and raises the trimming to 10 percent.

```
equation eq4.breakls(method=user, breaks="1990q1 2010q4") m1 c @nv
   unemp
```

estimates the model with two user-specified break dates. In addition, the variable UNEMP is restricted to have common coefficients across the regimes.

Cross-references

See Chapter 34. "Least Squares with Breakpoints," beginning on page 535 of *User's Guide II* for discussion. See also "Multiple Breakpoint Tests" on page 238 of *User's Guide II*.

See `Equation::multibreak` (p. 207) for multiple breakpoint testing.

breakspec	Equation Views

Display the breakpoint specification results for an equation estimated using `breakls`.

Syntax

eq_name.**breakspec**

Options

p	Print basic estimation results.

Examples

```
equation eq1.breakls m1 c unemp
```

```
eq1.breakspec(p)
```

displays and prints the breakpoint determination results for the equation EQ1 estimated using Bai-Perron sequential $L + 1$ versus L tests to determine the optimal breaks.

Cross-references

See Chapter 34. "Least Squares with Breakpoints," beginning on page 535 of *User's Guide II* for discussion.

breaktest	Equation Views

Breakpoint test.

Carries out a breakpoint test for parameter stability in equations estimated using TSLS and GMM.

See `chow` for related tests in equations estimated using least squares.

Syntax

eq_name.**breaktest** *obs1 [obs2 obs3….]*

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.

Examples

The commands

```
equation eq1.gmm m1 c gdp cpi @ gdp(-1) cpi(-1)
eq1.breaktest 1960 1970
```

perform a GMM estimation of M1 on a constant, GDP and CPI, with lagged values of GDP and CPI used as instruments, and then perform a breakpoint test to test whether the parameter estimates for the periods prior to 1960, during the 1960s, and then after 1970 are stable.

Cross-references

See "GMM Breakpoint Test" on page 118 of the *User's Guide II* for discussion.

cdtest	Equation Views

Test for the presence of cross-sectional dependence in the residuals of panels equations.

Computes the Breusch-Pagan (1980) LM, Pesaran (2004) scaled LM, Pesaran (2004) CD, and Baltagi, and Feng and Kao (2012) bias-corrected scaled LM test for the residuals of a panel or pool equation, or panel series.

Syntax

eq_name.**cdtest**

Options

p	Print test results

Examples

```
equation eq1.ls(cx=f) @log(gsp) c @log(p_cap) @log(pc) @log(emp)
   unemp
eq1.cdtest
```

will estimate a panel model using the fixed effect estimator (EQ1) and then will compute and display the panel residual dependence test results.

Cross-references

See "Panel Cross-section Dependence Test" on page 1365 of *User's Guide II* for 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

ind = *arg*	Specifies whether and how to draw the individual coefficient intervals. The default is "ind = line" which plots the individual coefficient intervals as dashed lines. "ind = none" does not plot the individual intervals, while "ind = shade" plots the individual intervals as a shaded rectangle.
size = *number* (*default* = 0.95)	Set the size (level) of the confidence ellipse. You may specify more than one size by specifying a space separated list enclosed in double quotes.
dist = *arg*	Select the distribution to use for the critical value associated with the ellipse size. The default depends on estimation object and method. If the parameter estimates are least-squares based, the $F(2, n-2)$ distribution is used; if the parameter estimates are likelihood based, the $\chi^2(2)$ distribution will be employed. "dist = f" forces use of the F-distribution, while "dist = c" uses the χ^2 distribution.
prompt	Force the dialog to appear from within a program.
p	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 Intervals and Confidence Ellipses" on page 203 of the *User's Guide II* for discussion.

See also `Equation::wald` (p. 267).

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

l = *number* *(default* = 0*)*	Set value for the left censoring limit.
r = *number* *(default* = none*)*	Set value for the right censoring limit.
l = *series_name*, i	Set series name of the indicator variable for the left censoring limit.
r = *series_name*, i	Set series name of the indicator variable for the right censoring limit.
t	Estimate truncated model.
d = *arg* *(default* = "n"*)*	Specify error distribution: normal ("n"), logistic ("l"), Type I extreme value ("x").
optmethod = *arg*	Optimization method: "bfgs" (BFGS); "newton" (Newton-Raphson), "opg" or "bhhh" (OPG or BHHH), "legacy" (EViews legacy). Newton-Raphson is the default method.
optstep = *arg*	Step method: "marquardt" (Marquardt); "dogleg" (Dogleg); "linesearch" (Line search). Marquardt is the default method.
cov = *arg*	Covariance method: "ordinary" (default method based on inverse of the estimated information matrix), "huber" or "white" (Huber-White sandwich methods)., "cr" (cluster robust).
covinfo = *arg*	Information matrix method: "opg" (OPG); "hessian" (observed Hessian - default). (Applicable when non-legacy "optmethod = ").

df	Degree-of-freedom correct the coefficient covariance estimate.(For non-cluster robust methods estimated using non-legacy estimation).
h	Huber-White quasi-maximum likelihood (QML) standard errors and covariances. (Legacy option applicable when "optmethod = legacy").
crtype = *arg (default "cr1")*	Cluster robust weighting method: "cr0" (no finite sample correction), "cr1" (finite sample correction), when "cov = cr".
crname = *arg*	Cluster robust series name, when "cov = cr".
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 (see also `param` (p. 564) of the *Command and Programming Reference*).
s = *number*	Specify a number between zero and one to determine starting values as a fraction of EViews default values (out of range values are set to "*s = 1*").
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
prompt	Force the dialog to appear from within a program.
p	Print results.

Examples

The command:

```
eq1.censored(cov=huber) 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 31. "Discrete and Limited Dependent Variable Models," on page 425 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 234 of *User's Guide II* for further discussion.

See also Equation::facbreak (p. 133), Equation::breaktest (p. 86), Equation::ubreak (p. 258), and Equation::rls (p. 233).

cinterval	Equation Views

Confidence interval.

The confidence interval view displays a table of confidence intervals for each of the coefficients in the equation.

Syntax

eq_name.**cinterval***(options) arg*

where *arg* is a list of confidence levels, or the name of a scalar or vector in the workfile containing confidence levels.

Options

prompt	Force the dialog to appear from within a program.
nopair	Display the intervals concentrically. The default is to display them in pairs for each probability value

Examples

The set of commands:

```
equation eq1.ls lwage c edu edu^2 union
eq1.cinterval .95 .9 .75
```

displays the 95% confidence intervals followed by the 90% confidence levels, followed by the 75% confidence levels.

```
eq1.cinterval(nopair) .95 .9 .75
```

displays the 75% confidence interals nested inside the 90% intervals which in turn are nested inside the 95% intervals.

Cross-references

See also "Confidence Intervals and Confidence Ellipses" on page 203 of *User's Guide II*.

clearhist	Equation Procs

Clear the contents of the history attribute for equation objects.

Removes the equation's history attribute, as shown in the label view of the equation.

Syntax

equation_name.**clearhist**

Examples

```
eq1.clearhist
eq1.label
```

The first line removes the history from the equation EQ1, and the second line displays the label view of EQ1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names.

See also `Equation::label` (p. 174).

clearremarks	Equation Procs

Clear the contents of the remarks attribute.

Removes the equations's remarks attribute, as shown in the label view of the equation.

Syntax

equation_name.**clearremarks**

Examples

```
e1.clearremarks
e1.label
```

The first line removes the remarks from the equation E1, and the second line displays the label view of E1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names.

See also `Equation::label` (p. 174).

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. 26).

coeflabel	Equation Views

Displays the coefficients associated with variables in the equation specification.

Syntax

equation_name.**coeflabel***(options)*

Options

p	Print the view.

Examples

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

displays and prints a view showing the coefficients associated with each variable in the equation specification.

Cross-references

See also `Equation::results` (p. 231).

coefmatrix	Equation Views

Display the matrix of lambda and coefficients for elastic net, ridge, and Lasso models.

Syntax

eq_name.**coefmatrix**(*options*)

Options

p	Print output.

Cross-references

For further discussion, see Chapter 37. "Elastic Net and Lasso," on page 597 of *User's Guide II*.

See also `Equation::coefpath` (p. 95).

coefpath	Equation Views

Display graphs of the paths of the coefficients plotted against lambda, fit measures, and estimation values.

This view is only available for equations estimated with elastic net, ridge regression, Lasso, and variable selection using Lasso.

You may plot the coefficient values against the paths of: the penalty parameter lambda, R-squared, adjusted R-square, standard error of the regression, sum-of-squared residuals, L1-norm coefficient penalty, L2-norm squared coefficient penalty, and the estimation objective.

A vertical line will be included to identify the selected optimal lambda.

Only coefficients that have non-zero values for at least one lambda in the path will be displayed.

By default, EViews will display a spool object containing all of the plots. You may use the "type = " option to produce a specific graph.

Syntax

eq_name.**coefpaths**(*options*)

Options

type = *arg*	Graph of coefficient against the path of: "lambda" (log lambda), "r2" (R-squared), "rbar2" (Adjusted R-squared), "se" (standard error of regression), "ssr" (sum-of-squared residuals), "l1" (L1 coefficient penalty, if applicable), "l2" (L2-squared coefficient penalty, if applicable), "estobj" (estimation objective). If "type = " is not provided, EViews will display the spool object all of the graphs.
p	Print output.

Examples

Consider the estimated elastic net equation

```
equation my_eq.enet(xtrans=none, lambdaratio=.0001,
    cvseed=513255899) lpsa c lcavol_s lweight_s age_s lbph_s svi_s
    lcp_s gleason_s pgg45_s
```

Then the command

```
my_eq.coefpath
```

displays estimates a spool containing graphs of all of the coefficients plotted against the paths of lambda, fit measures, and estimation measures.

```
my_eq.coefpath(type=lambda, p)
```

displays and prints a single graph of the coefficient lambda path, while

```
my_eq_coefpath(type=r2)
```

plots the coefficients against the path of the R-squared.

Cross-references

For further discussion, see Chapter 37. "Elastic Net and Lasso" in *User's Guide II*.

See also `Equation::coefmatrix` (p. 95).

The data underlying these graphs are available via the data members `@lambdacoefs`, `@lambdaest` and `@lambdafit`.

coefscale	Equation Views

Scaled coefficients.

Displays the coefficient estimates, the standardized coefficient estimates and the elasticity at means.

Syntax

eq_name.**coefscale**

Examples

The set of commands:

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

produces the coefficient scale table view of EQ1.

Cross-references

See also "Scaled Coefficients" on page 203 of *User's Guide II*.

coint	Equation Views

Test for cointegration between series in an equation.

Test for cointegration between series in an equation estimated by `Equation::cointreg` (p. 102). You may perform a Hansen Instability Test, Park Added Variable (Spurious Trends) Test, or between a residual-based Engle-Granger or Phillips-Ouliaris test.

Johansen tests for cointegration may be performed from a group or a VAR object (see `Group::coint` (p. 442) and `Var::coint` (p. 1149)).

The cointegrating equation specification is taken from the equation. Additional test specification components are specified as options and arguments.

Syntax

Equation View: eq_name.**coint**(*options*) *[arg]*

where

method = *arg* (*default* = "hansen")	Test method: Hansen's Instability test ("hansen"), Park's Added Variable ("park"), Engle-Granger residual test ("eg"), Phillips-Ouliaris residual test ("po").

and *arg* is an optional list describing additional regressors to include in the Park Added Regressors test (when "method = park" is specified).

The Park, Engle-Granger, and Phillips-Ouliaris tests all have options which control various aspects of the test.

Options

Options for the Park Test

The following option, along with the optional argument described above, determines the additional regressors to include in the test equation.

trend = *arg* *(default* = two orders higher than trend in estimated equation*)*	Specification for the powers of trend to include in the test equation: None ("none"), Constant ("const"), Linear trend ("linear"), Quadratic trend ("quadratic"), Cubic trend ("cubic"), Quartic trend ("quartic"), *integer* (user-specified power). Note that the specification implies all trends up to the specified order so that choosing a quadratic trend instructs EViews to include a constant and a linear trend term along with the quadratic. Only trend orders higher than those specified in the original equation will be considered.
p	Print results.

Options for the Engle-Granger Test

The following options determine the specification of the Engle-Granger test (Augmented Dickey-Fuller) equation and the calculation of the variances used in the test statistic.

lag = *arg* *(default* = "a"*)*	Method of selecting the 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).
lagtype = *arg* *(default* = "sic"*)*	Information criterion or method 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), "tstat" (*t*-statistic).
maxlag = *integer*	Maximum lag length to consider when performing automatic lag-length selection *default* = $\mathrm{int}(\min((T-k)/3, 12)\cdot(T/100)^{1/4})$ where k is the number of coefficients in the cointegrating equation. Applicable when "lag = a".
lagpval = *number* *(default* = 0.10*)*	Probability threshold to use when performing automatic lag-length selection using a *t*-test criterion. Applicable when both "lag = a" and "lagtype = tstat".
nodf	Do not degree-of-freedom correct estimates of the variances.
p	Print results.

Options for the Phillips-Ouliaris Test

The following options control the computation of the symmetric and one-sided long-run variances in the Phillips-Ouliaris test.

Basic Options

nodf	Do not degree-of-freedom correct the coefficient covariance estimate.
p	Print results.

HAC Whitening Options

lag = *arg (default* = 0*)*	Lag specification: *integer* (user-specified lag value), "a" (automatic selection).
infosel = *arg (default* = "aic"*)*	Information criterion for automatic selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn) (if "lag = a").
maxlag = *integer*	Maximum lag-length for automatic selection *(optional)* (if "lag = a"). The default is an observation-based maximum.

HAC Kernel Options

kern = *arg (default* = "bart"*)*	Kernel shape: "none" (no kernel), "bart" (Bartlett, *default*), "bohman" (Bohman), "daniell" (Daniel), "parzen" (Parzen), "parzriesz" (Parzen-Riesz), "parzgeo" (Parzen-Geometric), "parzcauchy" (Parzen-Cauchy), "quadspec" (Quadratic Spectral), "trunc" (Truncated), "thamm" (Tukey-Hamming), "thann" (Tukey-Hanning), "tparz" (Tukey-Parzen).
bw = *arg (default* = "nwfixed"*)*	Bandwidth: "fixednw" (Newey-West fixed), "andrews" (Andrews automatic), "neweywest" (Newey-West automatic), *number* (User-specified bandwidth).
nwlag = *integer*	Newey-West lag-selection parameter for use in nonparametric bandwidth selection (if "bw = neweywest").
bwoffset = *integer (default* = 0*)*	Apply integer offset to bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").
bwint	Use integer portion of bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").

Examples

Hansen

```
equation base_eq.cointreg(trend=linear, bw=andrews, kern=quadspec)
```

```
base_eq.coint
```

estimates the cointegrating equation BASE_EQ using FMOLS and performs the Hansen cointegration test.

Park

```
base_eq.coint(method=park)
```

conducts the default Park test, which for BASE_EQ involves testing the significance of the quadratic and cubic trend coefficients.

```
base_eq.coint(method=park, trend=quartic) mytrend
```

performs a test which evaluates the significance of the quadratic, cubic, and quartic terms, and user trend variable MYTREND.

```
base_eq.coint(method=eg, trend=6)
```

estimates the test equation with trend powers up to 6.

Engle-Granger

```
base_eq.coint(method=eg)
```

performs the default Engle-Granger test using SIC and an observation-based maximum number of lags to determine the lags for an ADF equation.

```
base_eq.coint(method=eg, lag=a, lagtype=tstat, lagpval=.15,
    maxlag=10)
```

uses a sequential *t*-test starting at lag 10 with threshold probability 0.15 to determine the number of lags.

```
base_eq.coint(method=eg, lag=5)
```

conducts an Engle-Granger cointegration test with a fixed lag of 5.

Phillips-Ouliaris

```
base_eq.coint(method=po)
```

performs the default Phillips-Ouliaris test using a Bartlett kernel and Newey-West fixed bandwidth.

```
base_eq.coint(method=po, bw=andrews, kernel=quadspec, nodf)
```

estimates the long-run covariances using a Quadratic Spectral kernel, Andrews automatic bandwidth, and no degrees-of-freedom correction.

```
base_eq.coint(method=po, lag=1, bw=4)
```

constructs the long-run covariances using AR(1) prewhitening, a fixed bandwidth of 4, and the Bartlett kernel.

Cross-references

See Chapter 59. "Cointegration Testing," beginning on page 1461 of *User's Guide II*. See also `Group::coint` (p. 442) for testing from a group object.

cointgraph	Equation Views

View a graph of the estimated cointegrating relation form of an ARDL estimated equation.

This view is only available for non-panel equations estimated using the ARDL method.

Syntax

equation_name.**cointgraph**

Examples

```
wfopen http://www.stern.nyu.edu/~wgreene/Text/Edition7/TableF5-
    2.txt
equation eq02.ardl(deplags=3, reglags=3, fixed) log(realcons)
    log(realgdp) @ @expand(@quarter, @droplast)
show eq02.cointgraph
```

This example uses data from Greene (2008, page 685), containing quarterly US macroeconomic variables between 1950 and 2000. The first line of this example downloads the data set, the second line creates an equation object and estimates an ARDL model with the log of real consumption as the dependent variable. Three lags of the dependent variable, and three lags of the log of real GDP are used as dynamic regressors. Quarterly dummy variables are included as static regressors.

The final line views a graph of the cointegration representation of the estimation.

Cross-references

See Chapter 29. "ARDL and Quantile ARDL," on page 341 of *User's Guide II* for further discussion.

See also `Equation::cointrep` (p. 110).

cointreg	Equation Methods

Estimate a cointegrating equation using Fully Modified OLS (FMOLS), Canonical Cointegrating Regression (CCR), or Dynamic OLS (DOLS) in single time series settings, and Panel FMOLS and DOLS in panel workfiles.

Syntax

eq_name.**cointreg**(*options*) *y x1 [x2 x3 ...] [*@**determ** *determ_spec] [*@**regdeterm** *regdeterm_spec]*

List the `cointreg` keyword, followed by the dependent variable and a list of the cointegrating variables.

Cointegrating equation specifications that include a constant, linear, or quadratic trends, should use the "trend = " option to specify those terms. If any of those terms are in the stochastic regressors equations but not in the cointegrating equation, they should be specified using the "regtrend = " option.

Deterministic trend regressors that are not covered by the list above may be specified using the keywords **@determ** and **@regdeterm**. To specify deterministic trend regressors that enter into the regressor and cointegrating equations, you should add the keyword **@determ** followed by the list of trend regressors. To specify deterministic trends that enter in the regressor equations but not the cointegrating equation, you should include the keyword **@regdeterm** followed by the list of trend regressors.

Basic Options

method = *arg* *(default* = "fmols"*)*	Estimation method: Fully Modified OLS ("fmols"), Canonical Cointegrating Regression ("ccr"), Dynamic OLS ("dols") Note that CCR estimation is not available in panel settings.
trend = *arg* *(default* = "const"*)*	Specification for the powers of trend to include in the cointegrating and regressor equations: None ("none"), Constant ("const"), Linear trend ("linear"), Quadratic trend ("quadratic"). Note that the specification implies all trends up to the specified order so that choosing a quadratic trend instructs EViews to include a constant and a linear trend term along with the quadratic.

regtrend = *arg* *(default* = "none"*)*	Additional trends to include in the regressor equations (but not the cointegrating equation): None ("none"), Constant ("const"), Linear trend ("linear"), Quadratic trend ("quadratic"). Only trend orders higher than those specified by "trend = " will be considered. Note that the specification implies all trends up to the specified order so that choosing a quadratic trend instructs EViews to include a constant and a linear trend term along with the quadratic.
regdiff	Estimate the regressor equation innovations directly using the difference specifications.
coef = *arg*	Specify the name of the coefficient vector; the default behavior is to use the "C" coefficient vector.
btwcoefs = *arg*	Save the cross-section specific deterministic coefficient estimates in a matrix object (one row per cross-section).
btwcovs = *arg*	Save the covariances of the cross-section specific deterministic coefficient estimates in a matrix object (one row per cross-section, with each row holding the vech of the covariance).
prompt	Force the dialog to appear from within a program.
p	Print results.

In addition to these options, there are specialized options for each estimation method.

Panel Options

panmethod = *arg* *(default* = "pooled"*)*	Panel estimation method: pooled ("pooled"), pooled weighted ("weighted"), grouped ("grouped")
pancov = sandwich	Estimate the coefficient covariance using a sandwich method that allows for cross-section heterogeneity.

Options for FMOLS and CCR

To estimate FMOLS or CCR use the "method = fmols" or "method = ccr" options. The following options control the computation of the symmetric and one-sided long-run covariance matrices and the estimate of the coefficient covariance.

HAC Whitening Options

lag = *arg (default* = 0*)*	Lag specification: *integer* (user-specified lag value), "a" (automatic selection).

infosel = *arg* *(default* = "aic"*)*	Information criterion for automatic selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn) (if "lag = a").
maxlag = *integer*	Maximum lag-length for automatic selection *(optional)* (if "lag = a"). The default is an observation-based maximum.

HAC Kernel Options

kern = *arg* *(default* = "bart"*)*	Kernel shape: "none" (no kernel), "bart" (Bartlett, *default*), "bohman" (Bohman), "daniell" (Daniell), "parzen" (Parzen), "parzriesz" (Parzen-Riesz), "parzgeo" (Parzen-Geometric), "parzcauchy" (Parzen-Cauchy), "quadspec" (Quadratic Spectral), "trunc" (Truncated), "thamm" (Tukey-Hamming), "thann" (Tukey-Hanning), "tparz" (Tukey-Parzen).
bw = *arg* *(default* = "nwfixed"*)*	Bandwidth:: "fixednw" (Newey-West fixed), "andrews" (Andrews automatic), "neweywest" (Newey-West automatic), *number* (User-specified bandwidth).
nwlag = *integer*	Newey-West lag-selection parameter for use in nonparametric bandwidth selection (if "bw = neweywest").
bwoffset = *integer* *(default* = 0*)*	Apply integer offset to bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").
bwint	Use integer portion of bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").

Coefficient Covariance

nodf	Do not degree-of-freedom correct the coefficient covariance estimate.

Panel Options

hetfirst	Estimate the first-stage regression assuming heterogeneous coefficients. For FMOLS specifications estimated using pooled or pooled weighted methods ("panmethod = pooled", "panmethod = weighted")

Options for DOLS

To estimate using DOLS use the "method = dols" option. The following options control the specification of the lags and leads and the estimate of the coefficient covariance.

lltype = *arg* *(default* = "fixed"*)*	Lag-lead method: fixed values ("fixed"), automatic selection - Akaike ("aic"), automatic - Schwarz ("sic"), automatic - Hannan-Quinn ("hqc"), None ("none").
lag = *arg*	Lag specification: *integer* (*required* user-specified number of lags if "lltype = fixed").
lead = *arg*	Lead specification: *integer* (*required* user-specified number of lags if "lltype = fixed").
maxll = *integer*	Maximum lag and lead-length for automatic selection (*optional* user-specified integer if "lltype = " is used to specify automatic selection). The default is an observation-based maximum.
cov = *arg*	Coefficient covariance method: (default) long-run variance scaled OLS, unscaled OLS ("ols"), White ("white"), Newey-West ("hac").
nodf	Do not degree-of-freedom correct the coefficient covariance estimate.

For the default covariance calculation or "cov = hac", the following options control the computation of the long-run variance or robust covariance:

HAC Covariance Whitening Options (if default covariance or "cov=hac")

covlag = *arg* *(default* = 0*)*	Lag specification: *integer* (user-specified lag value), "a" (automatic selection).
covinfosel = *arg* *(default* = "aic"*)*	Information criterion for automatic selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn) (if "lag = a").
covmaxlag = *integer*	Maximum lag-length for automatic selection *(optional)* (if "lag = a"). The default is an observation-based maximum.

HAC Covariance Kernel Options (if default covariance or "cov=hac")

covkern = *arg* (*default* = "bart")	Kernel shape: "none" (no kernel), "bart" (Bartlett, *default*), "bohman" (Bohman), "daniell" (Daniel), "parzen" (Parzen), "parzriesz" (Parzen-Riesz), "parzgeo" (Parzen-Geometric), "parzcauchy" (Parzen-Cauchy), "quadspec" (Quadratic Spectral), "trunc" (Truncated), "thamm" (Tukey-Hamming), "thann" (Tukey-Hanning), "tparz" (Tukey-Parzen).
covbw = *arg* *(default* = "nwfixed"*)*	Bandwidth: "fixednw" (Newey-West fixed), "andrews" (Andrews automatic), "neweywest" (Newey-West automatic), *number* (User-specified bandwidth).
covnwlag = *integer*	Newey-West lag-selection parameter for use in nonparametric bandwidth selection (if "covbw = neweywest").
covbwoffset = *integer* *(default* = 0*)*	Apply integer offset to bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").
covbwint	Use integer portion of bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").

Panel Options

Weighted coefficient or coefficient covariance estimation in panel DOLS requires individual estimates of the long-run variances of the residuals. You may compute these estimates using the standard default long-run variance options, or you may choose to estimate it using the ordinary variance.

For weighted estimation we have:

panwgtlag = *arg* *(default* = 0*)*	Lag specification: *integer* (user-specified lag value), "a" (automatic selection).
panwgtinfosel = *arg* *(default* = "aic"*)*	Information criterion for automatic selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn) (if "lrvarlag = a").
panwgtmaxlag = *integer*	Maximum lag-length for automatic selection *(optional)* (if "lrvarlag = a"). The default is an observation-based maximum.
panwgtkern = *arg* *(default* = "bart"*)*	Kernel shape: "none" (no kernel), "bart" (Bartlett, *default*), "bohman" (Bohman), "daniell" (Daniell), "parzen" (Parzen), "parzriesz" (Parzen-Riesz), "parzgeo" (Parzen-Geometric), "parzcauchy" (Parzen-Cauchy), "quadspec" (Quadratic Spectral), "trunc" (Truncated), "thamm" (Tukey-Hamming), "thann" (Tukey-Hanning), "tparz" (Tukey-Parzen).

panwgtbw = *arg* *(default* = "nwfixed"*)*	Bandwidth:: "fixednw" (Newey-West fixed), "andrews" (Andrews automatic), "neweywest" (Newey-West automatic), *number* (User-specified bandwidth).
panwgtnwlag = *integer*	Newey-West lag-selection parameter for use in nonparametric bandwidth selection (if "bw = neweywest").
panwgtbwoffset = *integer* *(default* = 0*)*	Apply offset to automatically selected bandwidth. For settings where "cov = hac", "covkern = " is specified, and "covbw = " is not user-specified.
panwgtbwint	Use integer portion of bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").

For the coefficient covariance estimation we have:

lrvar = ordinary	Compute DOLS estimates of the long-run residual variance used in covariance calculation using the ordinary variance.
lrvarlag = *arg* *(default* = 0*)*	For DOLS estimates of the long-run residual variance used in covariance calculation, lag specification: *integer* (user-specified lag value), "a" (automatic selection).
lrvarinfosel = *arg* *(default* = "aic"*)*	For DOLS estimates of the long-run residual variance used in covariance calculation, information criterion for automatic selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn) (if "lrvarlag = a").
lrvarmaxlag = *integer*	For DOLS estimates of the long-run residual variance used in covariance calculation, maximum lag-length for automatic selection *(optional)* (if "lrvarlag = a"). The default is an observation-based maximum.
lrvarkern = *arg* *(default* = "bart"*)*	For DOLS estimates of the long-run residual variance used in covariance calculation, Kernel shape: "none" (no kernel), "bart" (Bartlett, *default*), "bohman" (Bohman), "daniell" (Daniell), "parzen" (Parzen), "parzriesz" (Parzen-Riesz), "parzgeo" (Parzen-Geometric), "parzcauchy" (Parzen-Cauchy), "quadspec" (Quadratic Spectral), "trunc" (Truncated), "thamm" (Tukey-Hamming), "thann" (Tukey-Hanning), "tparz" (Tukey-Parzen).
lrvarbw = *arg* *(default* = "nwfixed"*)*	For DOLS estimates of the long-run residual variance used in covariance calculation, bandwidth:: "fixednw" (Newey-West fixed), "andrews" (Andrews automatic), "neweywest" (Newey-West automatic), *number* (User-specified bandwidth).

lrvarnwlag = *integer*	For DOLS estimates of the long-run residual variance used in covariance calculation, Newey-West lag-selection parameter for use in nonparametric bandwidth selection (if "bw = neweywest").
lrvarbwoffset = *integer (default = 0)*	For DOLS estimates of the long-run residual variance used in covariance calculation, apply offset to automatically selected bandwidth. For settings where "cov = hac", "covkern = " is specified, and "covbw = " is not user-specified.
lrvarbwint	For DOLS estimates of the long-run residual variance used in covariance calculation, use integer portion of bandwidth.

Examples

FMOLS and CCR

To estimate, by FMOLS, the cointegrating equation for LC and LY including a constant, you may use:

```
equation fmols.cointreg(nodf, bw=andrews) lc ly
```

The long-run covariances are estimated nonparametrically using a Bartlett kernel and a bandwidth determined by the Andrews automatic selection method. The coefficient covariances are estimated with no degree-of-freedom correction.

To include a linear trend term in a model where the long-run covariances computed using the Quadratic Spectral kernel and a fixed bandwidth of 10, enter:

```
equation fmols.cointreg(trend=linear, nodf, bw=10, kern=quadspec)
    lc ly
```

A model with a cubic trend may be estimated using:

```
equation fmols.cointreg(trend=linear, lags=2, bw=neweywest,
    nwlag=10, kernel=parzen) lc ly @determ @trend^3
```

Here, the long-run covariances are estimated using a VAR(2) prewhitened Parzen kernel with Newey-West nonparametric bandwidth determined using 10 lags in the autocovariance calculations.

```
equation fmols.cointreg(trend=quadratic, bw=andrews, lags=a,
    infosel=aic, kernel=none, regdiff) lc ly @regdeterm @trend^3
```

estimates a restricted model with a cubic trend term that does not appear in the cointegrating equation using a parametric VARHAC with automatic lag length selection based on the AIC. The residuals for the regressors equations are obtained by estimating the difference specification.

To estimate by CCR, we provide the "method = ccr" option. The command

```
equation ccr.cointreg(method=ccr, lag=2, bw=andrews,
    kern=quadspec) lc ly
```

estimates, by CCR, the constant only model using a VAR(2) prewhitened Quadratic Spectral and Andrews automatic bandwidth selection.

```
equation ccr.cointreg(method=ccr, trend=linear, lag=a, maxlag=5,
    bw=andrews, kern=quadspec) lc ly
```

modifies the previous estimates by adding a linear trend term to the cointegrating and regressors equations, and changing the VAR prewhitening to automatic selection using the default SIC with a maximum lag length of 5.

```
equation ccr.cointreg(method=ccr, trend=linear,
    regtrend=quadratic, lag=a, maxlag=5, bw=andrews) lc ly
```

adds a quadratic trend term to the regressors equations only, and changes the kernel to the default Bartlett.

DOLS

```
equation dols.cointreg(method=dols, trend=linear, nodf, lag=4,
    lead=4) lc ly
```

estimates the linear specification using DOLS with four lags and leads. The coefficient covariance is obtained by rescaling the no d.f.-correction OLS covariance using the long-run variance of the residuals computed using the default Bartlett kernel and default fixed Newey-West bandwidth.

```
equation dols.cointreg(method=dols, trend=quadratic, nodf, lag=4,
    lead=2, covkern=bohman, covbw=10) lc ly @determ @trend^3
```

estimates a cubic specification using 4 lags and 2 leads, where the coefficient covariance uses a Bohman kernel and fixed bandwidth of 10.

```
equation dols.cointreg(method=dols, trend=quadratic, nodf, lag=4,
    lead=2, cov=hac, covkern=bohman, covbw=10) lc ly @determ
    @trend^3
```

estimates the same specification using a HAC covariance in place of the scaled OLS covariance.

```
equation sols.cointreg(method=dols, trend=quadratic, lltype=none,
    cov=ols) lc ly @determ @trend^3
```

computes the static OLS estimates with the usual OLS d.f. corrected coefficient covariance.

Cross-references

See Chapter 28. "Cointegrating Regression," beginning on page 313 of *User's Guide II* for a discussion of single equation cointegrating regression. See Chapter 57. "Panel Cointegration

Estimation," beginning on page 1407 of *User's Guide II* for discussion of estimation in panel settings.

See "Technical Discussion" on page 980 of *User's Guide II* for a discussion of VEC estimation.

See also `Group::coint` (p. 442).

cointrel	Equation Views

Display information about the cointegrating relation specification and the coefficients in an ARDL estimated equation.

Syntax

eq_name.**cointrel**(*options*)

Options

p	Print output.

Example

```
ardl_eq.cointrel
```

displays a spool object with the table and graph showing the cointegrating relation.

Cross-references

See Chapter 29. "ARDL and Quantile ARDL," on page 341 of *User's Guide II* for further discussion.

See also `Equation::cointgraph` (p. 101) and `Equation::coint` (p. 97).

cointrep	Equation Views

View the estimated cointegration form and the long-run coefficients table of an ARDL estimated equation.

This view is only available for non-panel equations estimated using the ARDL method.

Syntax

equation_name.**cointrep**

Examples

```
wfopen http://www.stern.nyu.edu/~wgreene/Text/Edition7/TableF5-
    2.txt
```

```
equation eq02.ardl(deplags=3, reglags=3, fixed) log(realcons)
    log(realgdp) @ @expand(@quarter, @droplast)
show eq02.cointrep
```

This example uses data from Greene (2008, page 685), containing quarterly US macroeconomic variables between 1950 and 2000. The first line of this example downloads the data set, the second line creates an equation object and estimates an ARDL model with the log of real consumption as the dependent variable. Three lags of the dependent variable, and three lags of the log of real GDP are used as dynamic regressors. Quarterly dummy variables are included as static regressors.

The final line views the cointegration representation of the estimation, as well as the long-run form of the coefficient estimates.

Cross-references

See Chapter 29. "ARDL and Quantile ARDL," on page 341 of *User's Guide II* for further discussion.

See also `Equation::cointgraph` (p. 101).

copy	Equation Procs

Creates a copy of the equation.

Creates either a named or unnamed copy of the equation.

Syntax

equation_name.**copy**

equation_name.**copy** *dest_name*

Examples

```
eq1.copy
```

creates an unnamed copy of the equation EQ1.

```
eq1.copy eq2
```

creates EQ2, a copy of the equation EQ1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

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 489 and "Partial Autocorrelations (PAC)" on page 490 of *User's Guide I* for a discussion of autocorrelation and partial correlation functions, respectively.

See also `Equation::correlsq` (p. 112).

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 489 and "Partial Autocorrelations (PAC)" on page 490 of *User's Guide I* for a discussion of autocorrelation and partial correlation functions, respectively.

See also `Equation::correl` (p. 112).

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 or provide a linear specification.

Options

d = *arg* *(default* = "p"*)*	Likelihood specification: Poisson likelihood ("p"), normal quasi-likelihood ("n"), exponential likelihood ("e"), negative binomial likelihood or quasi-likelihood ("b").
v = *positive_num* *(default* = 1*)*	Specify fixed value for QML parameter in normal and negative binomial quasi-likelihoods.
optmethod = *arg*	Optimization method: "bfgs" (BFGS); "newton" (Newton-Raphson), "opg" or "bhhh" (OPG or BHHH), "legacy" (EViews legacy). Newton-Raphson is the default method.
optstep = *arg*	Step method: "marquardt" (Marquardt); "dogleg" (Dog-leg); "linesearch" (Line search). Marquardt is the default method.
cov = *arg*	Covariance method: "ordinary" (default method based on inverse of the estimated information matrix), "huber" or "white" (Huber-White sandwich methods)., "glm" (GLM method),""cr" (cluster robust).

covinfo = *arg*	Information matrix method: "opg" (OPG); "hessian" (observed Hessian). (Applicable when non-legacy "optmethod = ".)
df	Degree-of-freedom correct the coefficient covariance estimate.(For non-cluster robust methods estimated using non-legacy estimation).
h	Huber-White quasi-maximum likelihood (QML) standard errors and covariances. (Legacy option Applicable when "optmethod = legacy").
g	GLM standard errors and covariances. (Legacy option Applicable when "optmethod = legacy").
crtype = *arg* *(default "cr1")*	Cluster robust weighting method: "cr0" (no finite sample correction), "cr1" (finite sample correction), when "cov = cr".
crname = *arg*	Cluster robust series name, when "cov = cr".
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 (see also `param` (p. 564) of the *Command and Programming Reference*).
s = *number*	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").
prompt	Force the dialog to appear from within a program.
p	Print the result.

Examples

The command:

```
equation eq1.count(d=n,v=2,cov=glm) 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 471 of *User's Guide II* for additional discussion.

cvardecomp	Equation Views

Displays the coefficient covariance decomposition table.

Syntax

equation_name.**cvardecomp**

Examples

```
equation e1.ls y c x
eq1.cvardecomp
```

creates and estimates an equation named E1, and then displays the coefficient covariance decomposition table.

Cross-references

See "Coefficient Variance Decomposition" on page 208 of *User's Guide II* for a discussion.

cvgraph	Equation Views

Display a graph of the cross-validation objective against the lambda path.

This view is only available for equations estimated with elastic net, ridge regression, Lasso, and variable selection using Lasso.

Show a graph of the cross-validation objective for each value of lambda, along with bars representing +/– one standard deviation of the mean (for cross-validation methods that produce multiple training samples for each lambda).

A vertical line will be included to identify the selected optimal lambda.

Syntax

eq_name.**cvgraph**(*options*)

Options

p	Print output.

Examples

Consider the estimated elastic net equation

```
equation my_eq.enet(xtrans=none, lambdaratio=.0001,
   cvseed=513255899) lpsa c lcavol_s lweight_s age_s lbph_s svi_s
   lcp_s gleason_s pgg45_s
```

Then the command

```
my_eq.cvgraph
```

displays a graph of the cross-validation measures of fit against the path of log lambda. The cross-validation selected optimal value of lambda is marked by a vertical line.

Cross-references

For further discussion, see Chapter 37. "Elastic Net and Lasso" in *User's Guide II*.

See also `Equation::lambdacoefs` (p. 175), `Equation::lambdapath` (p. 178).

The data underlying this graph are available via the data members `@lambdafit`, and `@cvobjective`.

depfreq — Equation Views

Dependent variable frequency table.

Displays the frequency table for the dependent variable in binary, count, and ordered equations.

Syntax

equation_name.**depfreq**(*options*)

Options

p	Print the frequency table.

Examples

```
eq1.depfreq(p)
```

displays and prints the dependent variable frequency.

Cross-references

See also "Views of Binary Equations" on page 433, "Views of Ordered Equations" on page 448, and "Views of Count Models" on page 475 of *User's Guide II*.

See also `Equation::means` (p. 201).

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.

You may optionally choose a tabular or graphical display of the derivatives. 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

t	Display spreadsheet view of the values of the derivatives with respect to the coefficients evaluated at each observation.
g	Display multiple graphs showing the derivatives of the equation specification 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" on page 1537 of *User's Guide II* for details on the computation of derivatives.

See also Equation::makederivs (p. 190) for additional routines for examining derivatives, and Equation::grads (p. 165), and Equation::makegrads (p. 194) for corresponding routines for gradients.

did	Equation Methods

Estimate a equation in a panel structured workfile using the difference-in-difference estimator.

Syntax

equation.**did***(options) y [x1] [@ treatment]*

List the dependent variable, followed by an optional list of exogenous regressors, followed by an "@" and then the binary treatment variable. You should not include a constant in the specification.

Options

coef = *arg*	Specify the name of the coefficient vector. The default behavior is to use the "C" coefficient vector.
prompt	Force the dialog to appear from within a program.
p	Print results.

Examples

```
equation eq1.did asmrs @ post
```

estimates an equation by difference-in-difference with ASMRS as the outcome variable, and POST as the treatment variable.

```
equation eq2.did lemp lpop @ treated
```

estimates an equation by difference-in-difference with LEMP as the outcome variable, TREATED as the treatment variable, and LPOP as an exogenous regressor.

Cross-references

See Chapter 56. "Difference-in-Difference Estimation," on page 1383 of *User's Guide II* for a discussion of difference-in-difference models.

didcs	Equation Views

Display the Callaway-Sant'Anna decomposition for difference-in-difference estimation.

For panel equations estimated using the difference-in-difference method.

Syntax

eq_name.**didcs**(*options*) *[additional_regs]*

You should follow the `didcs` keyword by an optional list of additional regressors added to the Callaway-Sant'Anna estimation.

Options

notyet	Use observations where an individual is not yet treated as the comparison group. Default is to only use individuals that are never treated as the comparison.
both	Use both observations where an individual is never treated or has not yet been treated as the comparison group. Default is to only use individuals that are never treated as the comparison.
p	Print output.

Example

```
equation eq1.did lemp @ treated
eq1.didcs lpop
```

estimates an equation by difference-in-difference with LEMP as the outcome variable, and TREATED as the treatment variable, and then displays the Callaway-Sant'Anna decomposition, adding LPOP as an additional exogenous regressor.

Cross-references

See Chapter 56. "Difference-in-Difference Estimation," on page 1383 of *User's Guide II* for a discussion of difference-in-difference models.

See also `Equation::didgbdecomp` (p. 120), `Equation::didmakeeq` (p. 120), and `Equation::didtrends` (p. 121).

didgbdecomp	Equation Views

Display the Goodman-Bacon decomposition for difference-in-difference estimation.

For panel equations estimated using the difference-in-difference method.

Syntax

eq_name.**didgbdecomp**(*options*)

Options

p	Print output.

Example

```
equation eq1.did asmrs @ post
eq1.didgbdecomp
```

estimates an equation by difference-in-difference with ASMRS as the outcome variable, and POST as the treatment variable, and then displays the Goodman-Bacon decomposition.

Cross-references

See Chapter 56. "Difference-in-Difference Estimation," on page 1383 of *User's Guide II* for a discussion of difference-in-difference models.

See also `Equation::didcs` (p. 119), `Equation::didmakeeq` (p. 120), and `Equation::didtrends` (p. 121).

didmakeeq	Equation Procs

Create an equation object with the underlying fixed-effects estimation of a difference-in-difference equation.

For panel equations estimated using the difference-in-difference method.

Syntax

eq_name.**didmakeeq** *new_eqname*

You should follow the `didmakeeq` keyword with the name of the new estimated equation with an equivalent specification to be created in the workfile.

Example

```
equation eq1.did asmrs @ post
eq1.didmakeeq eq_underlying
```

estimates an equation by difference-in-difference with ASMRS as the outcome variable, and POST as the treatment variable, and then creates the underlying fixed effects estimation in the equation object EQ_UNDERLYING

Cross-references

See Chapter 56. "Difference-in-Difference Estimation," on page 1383 of *User's Guide II* for a discussion of difference-in-difference models.

See also `Equation::didgbdecomp` (p. 120), `Equation::didcs` (p. 119), and `Equation::didtrends` (p. 121).

didtrends	Equation Views

Display difference-in-difference trends summary in graph or tabular form.

Syntax

eq_name.**didtrends**(*options*)

For panel equations estimated using the difference-in-difference method.

Options

t	Display results in a table.
p	Print output.

Example

```
equation eq1.did asmrs @ post
eq1.didtrends
eq1.didtrends(t)
```

estimates an equation by difference-in-difference with ASMRS as the outcome variable, and POST as the treatment variable, and then displays the trend summary graph, then as a table.

Cross-references

See Chapter 56. "Difference-in-Difference Estimation," on page 1383 of *User's Guide II* for a discussion of difference-in-difference models.

See also `Equation::didgbdecomp` (p. 120), `Equation::didmakeeq` (p. 120), and `Equation::didcs` (p. 119).

display	Equation Views

Display table, graph, or spool output in the equation object window.

Display the contents of a table, graph, or spool in the window of the equation object.

Syntax

equation_name.**display** *object_name*

Examples

```
equation1.display tab1
```

Display the contents of the table TAB1 in the window of the object EQUATION1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

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 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Equation::label` (p. 174).

dynmult	Equation Views

Dynamic multipliers for long-run regressors in ARDL equations.

Displays a spool object with the cumulative dynamic multiplier curve for each of the long-run regressors. The argument is a positive integer denoting the horizon length, and defaults to 15.

Syntax

eq_name.**dynmult**(*options*) *[horizon]*

horizon is a positive integer denoting the horizon length, and defaults to 15.

Options

noci	Do not generate confidence intervals for asymmetric regressors. Note that confidence intervals can only be generated for asymmetric regressors.
noshade	Display confidence interval using lines instead of shaded bands.
level = *number* *(default = 0.95)*	Number between 0 and 1 representing the confidence interval level.
reps = *integer* *(default = 999)*	Number of Monte Carlo repetitions used in the generation of confidence intervals (if applicable).
f = *number*	Fraction of failed repetitions before stopping. Only applicable if a se_pattern is provided.
prompt	Force the dialog to appear from within a program.
p	Print output.

Example

```
ardl_eq.dynmult
```

generates cumulative dynamic multiplier curves for each long-run regressor. The horizon length is 15, and the 95% confidence intervals (if they exist), are shaded, and derived from 999 Monte Carlo replications.

```
ardl_eq.dynmult(noshade) 30
```

generates cumulative dynamic multiplier curves for each long-run regressor. The horizon length is 30, and the 95% confidence intervals (if they exist), are not shaded.

```
ardl_eq.dynmult(noci)
```

produces cumulative dynamic multiplier curves for each long-run regressor. The horizon length is 15, and no confidence intervals are displayed.

```
ardl_eq.dynmult(level=0.99, reps=499) 10
```

shows cumulative dynamic multiplier curves for each long-run regressor. The horizon length is 10, and the 99% confidence intervals (if they exist), are shaded, and derived from 499 Monte Carlo replications.

Cross-references

See Chapter 29. "ARDL and Quantile ARDL," on page 341 of *User's Guide II* for a discussion of ARDL equation models.

See also `Equation::ardl` (p. 71).

ecresults	Equation Views

Display a spool object showing tables with the conditional error correction (CEC) and error correction (EC) regression results in ARDL equations.

Syntax

eq_name.**ecresults**(*options*)

Options

p	Print output.

Example

```
ardl_eq.ecresults
```

displays a spool object with the CEC and EC regressions from the ARDL equation ARDL_EQ.

Cross-references

See Chapter 29. "ARDL and Quantile ARDL," on page 341 of *User's Guide II* for a discussion of ARDL equation models.

See also `Equation::cointreg` (p. 102) and `Equation::cointrel` (p. 110).

effects	Equation Views

Display the estimates of the fixed and/or random effects.

The `effects` view of a panel equation shows the estimates of the fixed and/or random effects associated with the estimated equation. These effects are expressed as deviations from the overall intercept displayed in the main equation output..

Syntax

eq_name.**effects**

Options

p	Print view.

Examples

```
equation eq1.ls(cx=f) y c x1 x2
e1.effects
```

estimates the equation EQ1 with fixed effects, and displays a view showing the estimated cross-section deviations from the overall intercept.

Cross-references

See Chapter 55. "Panel Estimation," on page 1321 of *User's Guide II* for a discussion of panel equation estimation.

endogtest	Equation Views

Performs the regressor endogeneity test

The `endogtest` view of an equation carries out the Regressor Endogeneity/Donald-Wu Test for equations estimated via TSLS or GMM.

Syntax

eq_name.**endogtest** *regressors*

Options

prompt	Force the dialog to appear from within a program.

Regressors

A list of equation regressors to be tested for endogeneity. Note the regressors must have been included in the original equation.

Examples

```
equation eq1.gmm y c x1 x2 @ z1 z2 z3 z4
e1.endogtest x1
```

estimates an equation, called EQ1, and estimates it via GMM, and then performs the Endogeneity Test, where X1 is tested for endogeneity.

Cross-references

See "Regressor Endogeneity Test" on page 115 of *User's Guide II* for a discussion.

enet	Equation Methods

Estimation of an elastic net model, including options for Lasso and ridge regression.

Syntax

equation_name.**enet***(options) y x1 [x2 x3 ...] [@vw(...)]*

List the dependent variable first, followed by a list of the independent variables. Use a "C" if you wish to include an unpenalized intercept term.

Note that PDL and ARMA terms are not permitted in elastic net specifications.

If you wish to specify regressors with an individual penalty weight ω_j, or to place inequality restrictions on the coefficient values, you may do so using special expressions of the form:

@vw(*series_name*, *weight_value)*

or

@vw(*series_name[,* **wgt** = *weight_value,* **cmin** = *coef_min,* **cmax** = *coef_max]*)

where *weight_value* is a non-negative value, *coef_min* is a non-positive minimum coefficient value, and *coef_max* is a non-negative maximum coefficient value.

There are two forms of the special expression.

In the abbreviated form, you specify the variable name followed by the penalty weight value.

In the more general form, you specify the variable name followed by one or more keyword expressions in arbitrary order, with the "`wgt=`" argument specifying the penalty weight, "`cmin=`" with a non-positive minimum coefficient value, and "`cmax=`" with a non-negative maximum coefficient value.

When specifying individual regressor behavior using `@vw`, keep in mind that:

- The special intercept variable "C" is always non-penalized and has an implicit weight $\omega = 0.0$.
- Individual penalty weights may be also specified using a vector in the **Individual lambda wgts. vector** edit field on the **Penalty** dialog page (or using the command estimation option "**lambdawgt** = *vector_name*"). If the vector weights are specified and individual weights are specified using the `@vw` keyword, the vector weights will be applied first, followed by the individual variable weights.
- Individual coefficient limit values may also be specified using vectors in the **Min values vector** and **Max values vector** edit fields on the **Options** dialog page (or the command estimation options "**coefmin** = *vector_name*" and "**coefmax** = *vector_name*"). If vector coefficient limits are specified and individual regressor limits are specified using the `@vw` keyword, the vector limits will be applied first, followed by the individual limits weights.

EViews will normalize the individual penalty weights so that they sum to the number of coefficients.

Options

Specification Options

penalty = *arg* (*default* = "el")	Type of threshold estimation: "enet" (elastic net), "ridge" (ridge), "lasso" (Lasso).
alpha = *arg* (*default* = ".5")	Value of the mixing parameter. Must be a value from zero to one.
lambda = *arg*	Value(s) of the penalty parameter. Can be one or more numbers or vector objects.Values must be zero or greater. If left blank (default) EViews will generate a list.

Penalty Options

ytrans = *arg* (*default* = "none")	Scaling of the dependent variable: "none" (none), "L1" (L1), "L2" (L2), "stdsmpl" (sample standard deviation), "stdpop" (population standard deviation), "minmax" (min-max).
xtrans = *arg* (*default* = "stdpop")	Scaling of the regressor variables: "none" (none), "L1" (L1 norm), "L2" (L2 norm), "stdsmpl" (sample standard deviation), "stdpop" (population standard deviation), "minmax" (min-max).
nlambdas = *integer* (*default = 100*)	Number of penalty values for EViews-supplied list.

lambdaratio = *arg*	Ratio of minimum to maximum lambda for EViews-supplied list. You may specify a value for the ratio parameter, or you may leave the edit field blank to let EViews specify a default value based on the number of observations T and the number of potential regressors p. By default, EViews will set the ratio to 0.001.
lambdawgt = *vector_name*	Vector of individual penalty weights, containing non-negative values sized to and matching the order of the variables in the specification. If a vector is provided and individual weights are specified using one or more `@vw` regressors, the vector weights will be applied first, then overwritten by the individual variable weights. For comparability purposes, we normalize the final weights so that they sum to p^* where p^* the number of non-zero ω_j.
nlambdamin = *integer (default = 5)*	Minimum number of lambda values in the path before applying stopping rules.
minddev = *arg (default* = 1e-05*)*	Minimum change in deviance fraction to continue estimation. Truncate path estimation if relative change in deviance is smaller than this value.
maxedev = *arg (default* = 0.99*)*	Maximum of deviance explained fraction attained to terminate estimation. Truncate path estimation if fraction of null deviance explained is larger than this value.
maxvars = *arg*	Maximum number of regressors in the model. Truncate path estimation if the number of coefficients (including those for non-penalized variables like the intercept) reaches this value.
maxvarsratio = *arg*	Maximum number of regressors in the model as a fraction of the number of observations. Truncate path estimation if the number of coefficients (including those for non-penalized variables like the intercept) divided by the number of observations reaches this value.

Cross Validation Options

cvmethod = *arg* (*default* = "kfold_cv")	Cross-validation method: "kfold" (k-fold), "simple" (simple split), "mcarlo" (Monte Carlo), "leavepout" (leave-P-out), "leave1out" (leave-1-out), "rolling" (rolling window), "expanding" (expanding window).
cvmeasure = *arg* (*default* = "mse")	Cross-validation fit measure: "mse" (mean-squared error), "r2" (R-squared), "mae" (mean absolute error), "mape" (mean absolute percentage error), "smape" (symmetric mean absolute percentage error).
cvnfolds = *arg* (*default* = 5)	Number of folds for K-fold cross-validation. For "cvmethod = kfold".
cvftrain = *arg* (*default* = 0.8)	Proportion of data for split and Monte Carlo methods. For "cvmethod = simple" and "cvmethod = mcarlo".
cvnreps = *arg* (*default* = 1)	Number of Monte Carlo method repetitions. For "cvmethod = mcarlo".
cvleaveout = *arg* (*default* = 2)	Number of data points left out for leave-p-out method. For "cvmethod = leavepout".
cvnwindows = *arg* (*default* = 4)	Number of windows for rolling window cross-validation method. For "cvmethod = rolling".
cvinitial = *arg* (*default* = 12)	Number of initial data points in the training set for expanding cross-validation. For "cvmethod = expanding".
cvpregap = *arg* (*default* = 0)	Number of observations between end of training set and beginning of test set. For "cvmethod = simple", "cvmethod = rolling" and "cvmethod = expanding".
cvhorizon = *arg* (*default* = 1)	Number of observation in the test set. For "cvmethod = rolling" and "cvmethod = expanding".
cvpostgap = *arg* (*default* = 0)	Number of observations between end of test set and beginning of next training set for rolling window or between end of test set and end of next training set for expanding window. For "cvmethod = rolling" and "cvmethod = expanding"

Random Number Options

seed = *positive_integer* from 0 to 2,147,483,647	Seed the random number generator. If not specified, EViews will seed random number generator with a single integer draw from the default global random number generator.
rnd = *arg* (*default* = "kn" or method previously set using `rndseed` (p. 577) in the *Command and Programming Reference*).	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").

Other Options

coefmin = *vector_name, number*	Vector of individual coefficient minimum values, containing negative or missing values sized to and matching the order of the variables in the specification, or a negative value for the minimum for all coefficients. Missing values in the vector should be used to indicate that the coefficient is unrestricted. If a vector of values is provided and individual minimums are specified using one or more `@vw` regressors, the vector values will be applied first, then overwritten by the individual values.
coefmax = *vector_name, number*	Vector of individual coefficient maximum values, containing positive or missing values sized to and matching the order of the variables in the specification, or a positive value for the maximum for all coefficients. Missing values in the vector should be used to indicate that the coefficient is unrestricted. If a vector of values is provided and individual maximums are specified using one or more `@vw` regressors, the vector values will be applied first, then overwritten by the individual values.
maxit = *integer*	Maximum number of iterations.
conv = *scalar*	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.
w = *arg*	Weight series or expression.

wtype = *arg* *(default* = "istdev"*)*	Weight specification type: inverse standard deviation ("istdev"), inverse variance ("ivar"), standard deviation ("stdev"), variance ("var").
wscale = *arg*	Weight scaling: EViews default ("eviews"), average ("avg"), none ("none"). The default setting depends upon the weight type: "eviews" if "wtype = istdev", "avg" for all others.
showopts / -showopts	[Do / do not] display estimation options in the output.
prompt	Force the dialog to appear from within a program.
p	Print basic results view after estimation.

Examples

The command

```
equation enet1.enet(xtrans=none, lambdaratio=.0001,
    cvseed=513255899) lpsa c lcavol lweight age lbph svi lcp
    gleason pgg45
```

estimates an elastic net model with α equal to the default of 0.9, no regressor or dependent variable scaling, automatically determined 100-element lambda path with minimum lambda of 0.0001 times the maximum value, using the default K-fold cross-validation with 5 folds with an MSE objective and a random generator seed of 513255899 to determine the optimal value.

Similarly,

```
equation lasso1.enet(penalty=lasso, lambdaratio=.0001,
    cvseed=513255899) lpsa c lcavol lweight age lbph svi lcp
    gleason pgg45
```

estimates a Lasso model with regressor population standard deviation scaling, with the remaining settings as before, while

```
equation ridge1.enet(penalty=ridge, lambdaratio=.0001,
    cvseed=513255899) lpsa c lcavol lweight age lbph svi lcp
    gleason pgg45
```

estimates the equivalent ridge regression specification.

The command

```
equation enet2.enet(alpha=0.75, lambdaratio=.0001,
    cvmethod=rolling, cvmeasure=smape) lpsa c lcavol lweight age_s
    lbph svi lcp gleason pgg45
```

estimates an elastic net model with α equal to the 0.75 using rolling window cross-validation and SMAPE cross-validation.

We may use the @vw specifications to assign individual penalties and coefficient restrictions

```
equation enet3.enet(alpha=0.75, lambdaratio=.0001,
    cvmethod=rolling, cvseed=513255899) lpsa c @vw(lcavol, cmax=.4)
    lweight age lbph svi lcp gleason @vw(pgg45, cmax=0.075, w=1.2)
```

estimates an elastic net model with the coefficient of LCAVOL restricted to be less than or equal to 0.4, and the coefficient of PGG45 having a relative penalty weight of 1.2, and a maximum value of 0.075.

Identical specifications may be estimated using vectors of penalty weights and coefficient restrictions,

```
vector(9) cmax = na
cmax(2) = 0.4
cmax(9) = 1.2
vector(9) lwgt = 1
lwgt(9) = 1.2
equation enet4.enet(alpha=0.75, coefmax=cmax, lambdawgt=lwgt,
    lambdaratio=.0001, cvmethod=rolling, cvseed=513255899) lpsa c
    lcavol lweight age lbph svi lcp gleason pgg45
```

and

```
vector(9) cmax = na
cmax(2) = 0.4
vector(9) lwgt = 1
lwgt(9) = 50
equation enet5.enet(alpha=0.75, coefmax=cmax, lambdawgt=lwgt,
    lambdaratio=.0001, cvmethod=rolling, cvseed=513255899) lpsa c
    lcavol lweight age lbph svi lcp gleason @vw(pgg45, cmax=0.075,
    w=1.2)
```

since the penalty weight for PGG45 in the vector is overwritten by the individual weight specified using the @vw.

Note that in neither case is the intercept penalized, even though the corresponding element of LWGT is equal to 1 since the specification of "C" is always implicitly treated as "@VW(C, 0)".

Cross-references

See Chapter 37. "Elastic Net and Lasso," on page 597 of *User's Guide II* for a discussion of elastic net, ridge regression, and Lasso models.

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 estimation method, and the equation specification. Valid estimation methods are given in "Equation Methods" on page 51. 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 20. "Basic Regression Analysis," on page 5 of *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 219 of *User's Guide II* for further discussion.

See also `Equation::chow` (p. 91), `Equation::breaktest` (p. 86), and `Equation::rls` (p. 233).

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. 196) 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

Basic 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.
ga	Graph the forecasts along with the actuals (if available).
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.
n	Ignore coef uncertainty in standard error calculations that use them.
forcsmpl = *smpl*	Fit sample (optional). If forecast sample is not provided, the workfile sample will be employed.
f = *arg* (*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).
prompt	Force the dialog to appear from within a program.
p	Print view.

Stochastic Options

Options for forecasting from a functional coefficients estimated equation.

stochastic = arg (default = "none")	Stochastic method: "none" (none), "mca" (Monte Carlo – asymptotic), "mcbs" (Monte Carlo – bootstrap), "bs" (bootstrap).
reps = integer (default = 999)	Number of stochastic replications

lhr = arg (default = 0.1)	Lower historical range (number between 0 and upper historical range).
uhr = arg (default = 0.9)	Upper historical range (number between lower historical range and 1).
bsdep	Bootstrap only the dependent variable (not the functional coefficient variable).

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=l) 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. 139). See `Equation::makemodel` (p. 196) and `Model::solve` (p. 640) for forecasting from systems of equations or ordered equations.

See Chapter 25. "Forecasting from an Equation," on page 169 of *User's Guide II* for a discussion of forecasting in EViews and Chapter 31. "Discrete and Limited Dependent Variable Models," on page 425 of *User's Guide II* for a discussion of forecasting from binary, censored, truncated, and count models.

fitoutliers	Equation Views

Detect outliers using the results of an estimated equation.

Use Tukey fences, mean/standard deviation fences, wavelet outliers, ARMA outliers or influence statistics to identify observations that may contain outliers.

Syntax

equation_name.**resoutliers***(options)*

Options

sens = *arg*	Set the sensitivity level. Valid arguments are "low", "medium" (default), and "high".
nofence	Do not perform Tukey and mean/standard deviation fences.
nowave	Do not perform Wavelet Outlier detection.
noarma	Do not perform ARMA based outlier detection. ARMA outlier detection is only available for least squares equations containing ARMA terms, and is turned on by default.
noinf	Do not perform influence statistic (not including DFBETAS) based outlier detection. Influence statistic outlier detection is only available for linear least squares equations, and is turned on by default.
dfbeta	Perform DFBETA influence statistic based outlier detection. DFBETA based outlier detection is only available for linear least squares equations, and is turned off by default.
tukeyk = *arg*	Set the value k in the Tukey fence detection routine. This will override the value of k set by the sens = option.

meanstdevk = *arg*	Set the value *k* in the mean/standard deviation fence detection routine. This will override the value of *k* set by the sens = option.
wavesig = *arg*	Set the value false discovery rate significance value used in the Wavelet Outlier detection routine. This will override the value set by the sens = option.
armac = *arg*	Set the value *c* in the ARMA outlier detection routine. This will override the value of *c* set by the sens = option.
rsbound = *arg*	Set the value *c* in RSTUDENT outlier detection. This will override the value of *c* set by the sens = option.
hbound = *arg*	Set the value *c* in HatMatrix outlier detection. This will override the value of *c* set by the sens = option.
dfsbound = *arg*	Set the value *c* in DFFITS outlier detection. This will override the value of *c* set by the sens = option.
covbound = *arg*	Set the value *c* in CovRatio outlier detection. This will override the value of *c* set by the sens = option.
betabound = *arg*	Set the value *c* in DFBETA outlier detection. This will override the value of *c* set by the sens = option.
series = *name*	Create a new series in the workfile, named *name*, containing a value of 1 for any observations identified as an outlier, and a value of 0 for any observation identified as not an outlier.
datestring = *name*	Create a new string object in the workfile containing the dates (or observation identifiers) for any observations identified as an outlier.
grlabels	Turn on observation labels on the outlier graph.
prompt	Force the dialog to appear from within a program.
p	Print results.

Examples

```
equation eq01.ls gdpc1 c unemp
eq01.resoutliers(nofence, dfbeta, sens=low)
```

Estimates an equation with GDPC1 as the dependent variable, and a constant and UNEMP as regressors. Then, outlier detection on the residuals is performed, opting to not use either fence detection, but to include the dfbeta influence statistics (along with the other influence statistics included by default), and setting the sensitivity of the detection to "low".

Cross-references

See "Outlier Detection" on page 198 of *User's Guide II* for discussion. See also `Series::outliers` (p. 818).

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 "Fixed Effects Testing" on page 1354 of *User's Guide II* for discussion.

See also `Equation::rcomptest` (p. 227) for testing random for random components.

forecast	Equation Procs

Computes 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.
ga	Graph the forecasts along with the actuals (if available).
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.
n	Ignore coef uncertainty in standard error calculations that use them.
b = *arg*	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.
forcsmpl = *smpl*	Forecast sample (optional). If forecast sample is not provided, the workfile sample will be employed
f = *arg* (*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).
stochastic	Perform stochastic simulation for dynamic equations estimated using least squares.
streps = *integer* (*default* = 1000)	Number of stochastic repetitions (for threshold regression or stochastic simulation).

stfrac = *number* *(default* = .02*)*	Fraction of failed repetitions before stopping (for threshold regression or stochastic simulation).
prompt	Force the dialog to appear from within a program.
p	Print view.

Examples

The following lines:

```
smpl 1970q1 1990q4
equation eq1.ls con c con(-1) inc
smpl 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. 134). For multiple equation forecasting, see `Equation::makemodel` (p. 196), and `Model::solve` (p. 640).

For more information on equation forecasting in EViews, see Chapter 25. "Forecasting from an Equation," on page 169 of *User's Guide II*.

funbias

Equation Views

Display functional coefficients bias results in graphical form.

For equations estimated using the functional coefficients method.

Syntax

eq_name.**funbias***(options)*

Options

Basic Options

nolocalbw	Do not use the existing local pilot bandwidth (if it exists). Note that this option is only available if a local pilot bandwidth has been set previously, either through the estimation step, using the setpilotbw proc, or through another view or proc which set the local bandwidth.
p	Print output.

Pilot Bandwidth Options

If a local pilot bandwidth exists, all of the pilot bandwidth computation options below will be ignored unless the "nolocalbw" option is specified.

noupdate	Do not update the local pilot bandwidth using the current pilot bandwidth computation.
plth = *arg* *(default = "cv")*	Pilot bandwidth method: simple rule-of-thumb ("rot"), robust rule-of-thumb ("rotr"), residual squares criterion ("rsc"), modified multi cross-validation ("cv"), user-defined ("user").
pltbw = *arg* *(default = 1)*	User-defined bandwidth (if "plth = user").
plthmin = *arg* *(default = 0.1)*	Bandwidth grid search minimum value (if not "plth = user").
plthmax = *arg* *(default = 1)*	Bandwidth grid search maximum value (if not "plth = user").
plthlen = *integer* *(default = 100)*	Bandwidth grid search length (if not "plth = user").
plthinc = *integer* *(default = 10)*	Bandwidth grid search increment step percentage increase (if not "plth = user").
plthcup = *integer* *(default = 10)*	Stop rule: consecutive increases of objective function before stop (not available when "plth = user").
pltm = *arg* *(default = 10)*	Modified multifold CV m-value: percentage of sample size used in bandwidth determination (when "plth = cv").
pltq = *integer* *(default = 4)*	Modified multifold CV Q-value: percentage of sample size used in bandwidth determination (when "plth = cv").
auxk = *integer* *(default = 2)*	Estimation polynomial degree for pilot stage in excess of final stage degree. This number should always be an even positive integer.

Examples

```
eq1.funbias(p)
```

displays and prints a graph of the functional bias results for each functional coefficient computed using the existing local pilot bandwidth if present, or estimating the pilot bandwidth, if not. If estimated, the local pilot bandwidth is updated with the result.

```
eq1.funbias(nolocalbw, noupdate, plth=rsc)
```

ignores an existing local pilot bandwidth and instead computes a pilot bandwidth using the residual squares criterion. The existing saved local pilot bandwidth is retained.

Cross-references

See Chapter 38. "Functional Coefficient Regression," on page 645 of *User's Guide II* for discussion of functional coefficients estimation See "Functional Bias" on page 654 of *User's Guide II* for discussion of bias.

See also `Equation::makefunobj` (p. 191) and `Equation::setpilotbw` (p. 237).

funbw	Equation Views

Display functional coefficients final bandwidth selection results.

For equations estimated using the functional coefficients method.

Syntax

eq_name.**funbw***(options)*

Options

Basic Options

pilot	Display pilot bandwidths.
graph	Display results in graphical form.
p	Print output.

Examples

```
eq1.funbw
```

displays the bandwidth results in table form.

```
eq1.funbw(graph, p)
```

displays and prints a graph of the estimation final bandwidth selection results.

```
eq1.funbw(pilot)
```

displays a table for results for the estimation and local pilot bandwidths.

Cross-references

See Chapter 38. "Functional Coefficient Regression," on page 645 of *User's Guide II* for discussion of functional coefficients estimation. See "Bandwidth Selection" on page 646 and "Bandwidth Views" on page 658 of *User's Guide II* for discussion of bandwidths.

See also `Equation::funcoef` (p. 146) and `Equation::setpilotbw` (p. 237).

funci	Equation Views

Display graphs showing estimated lower and upper functional confidence intervals for the functional coefficients.

For equations estimated using the functional coefficients method.

Syntax

eq_name.**funci***(options) ci_level*

where *ci_level* is a value between 0 and 1 specifying the confidence level you wish to display. If *ci_level* is omitted, EViews will use 0.95.

Options

Basic Options

nolocalbw	Do not use the existing local pilot bandwidth (if it exists). Note that this option is only available if a local pilot bandwidth has been set previously, either through the estimation step, using `Equation::setpilotbw` (p. 237), or through another view or proc which sets the local bandwidth.
p	Print output.

Confidence Interval Options

seband	Produce standard error bands instead of confidence intervals.
sewidth = *integer* (*default* = 1)	Number of standard errors to use as half-width of confidence band (when "seband" is specified).
nobias	Ignore bias in confidence interval determination.

Pilot Bandwidth Options

If a local pilot bandwidth exists, all of the pilot bandwidth computation options below will be ignored unless the "nolocalbw" option is specified.

noupdate	Do not update the local pilot bandwidth using the current pilot bandwidth computation.
plth = *arg* *(default = "cv")*	Pilot bandwidth method: simple rule-of-thumb ("rot"), robust rule-of-thumb ("rotr"), residual squares criterion ("rsc"), modified multi cross-validation ("cv"), user-defined ("user").
pltbw = *arg* *(default = 1)*	User-defined bandwidth (if "plth = user").
plthmin = *arg* *(default = 0.1)*	Bandwidth grid search minimum value (if not "plth = user").
plthmax = *arg* *(default = 1)*	Bandwidth grid search maximum value (if not "plth = user").
plthlen = *integer* *(default = 100)*	Bandwidth grid search length (if not "plth = user").
plthinc = *integer* *(default = 10)*	Bandwidth grid search increment step percentage increase (if not "plth = user").
plthcup = *integer* *(default = 10)*	Stop rule: consecutive increases of objective function before stop (not available when "plth = user").
pltm = *arg* *(default = 10)*	Modified multifold CV m-value: percentage of sample size used in bandwidth determination (when "plth = cv").
pltq = *integer* *(default = 4)*	Modified multifold CV Q-value: percentage of sample size used in bandwidth determination (when "plth = cv").
auxk = *integer* *(default = 2)*	Estimation polynomial degree for pilot stage in excess of final stage degree. This number should always be an even positive integer.

Examples

```
eq1.funci 0.95 0.99
```

displays a graph of the 95% and 99% confidence intervals computed using the existing local pilot bandwidth if present, or estimating the pilot bandwidth, if not. If estimated, the local pilot bandwidth is updated with the result.

```
eq1.funci(seband, sewidth=3)
```

displays and prints a graph of the +/−3 confidence intervals.

The saved local pilot bandwidth is updated with the resulting bandwidth.

```
eq1.funci(nolocalbw, noupdate, plth=rsc) 0.9
```

ignores an existing local pilot bandwidth and instead computes a pilot bandwidth using the residual squares criterion and uses it in a 90% confidence interval calculation. The existing saved local pilot bandwidth is retained.

Cross-references

See Chapter 38. "Functional Coefficient Regression," on page 645 of *User's Guide II* for discussion of functional coefficients estimation. See "Confidence Intervals" on page 660 of *User's Guide II* for discussion of functional confidence intervals.

See also `Equation::makefunobj` (p. 191) and `Equation::setpilotbw` (p. 237).

funcoef	Equation Methods

Estimate a functional coefficient regression equation.

Syntax

eq_name.**funcoef**(*options*) *y x1 [x2 x3 ...] @ funcoef_series*

List the `funcoef` keyword, the dependent variable and a list of the regressor variables, followed by the "@" symbol and the name of the functional coefficient series.

Options

Basic Options

kern = *arg* *(default* = "epan"*)*	Kernel type: "epan" (Epanechnikov, default), "trngl" (Triangular), "unif" (Uniform), "gauss" (Normal–Gaussian), "bi" (Biweight–Quartic), "tri" (Triweight).
eval = *arg* *(default* = "data"*)*	Evalution points: observed data ("data"), grid of values ("grid"). if "eval = grid" you must specify the grid values using "gmin = ", "gmax = " and "glen = ", or using "gvec = ".
gmin = *arg*	Estimation grid minimum (if "eval = grid"). Must be specified along with "gmax = " and "glen = ".
gmax = *arg*	Estimation grid maximum (if "eval = grid"). Must be specified along with "gmin = " and "glen = ".
glen = *arg*	Estimation grid length (if "eval = grid"). Must be specified along with "gmin = " and "gmax = ".
gvec = *arg*	Estimation grid points in a vector object (if "eval = grid").

plyk = *arg* *(default = 1)*	Estimation polynomial degree for final stage.
auxk = *arg* *(default = 2)*	Estimation polynomial degree for pilot stage in excess of final stage.
p	Print results.

Pilot Bandwidth Options

plth = *arg* *(default = "cv")*	Pilot bandwidth method: simple rule-of-thumb ("rot"), robust rule-of-thumb ("rotr"), residual squares criterion ("rsc"), modified multi cross-validation ("cv"), user-defined ("user").
pltbw = *arg* *(default = 1)*	User-defined bandwidth (if "plth = user").
plthmin = *arg* *(default = 0.1)*	Bandwidth grid search minimum value (if not "plth = user").
plthmax = *arg* *(default = 1)*	Bandwidth grid search maximum value (if not "plth = user").
plthlen = *integer* *(default = 100)*	Bandwidth grid search length (if not "plth = user").
plthinc = *integer* *(default = 10)*	Bandwidth grid search increment step percentage increase (if not "plth = user").
plthcup = *integer* *(default = 10)*	Stop rule: consecutive increases of objective function before stop (not available when "plth = user").
pltm = *arg* *(default = 10)*	Modified multifold CV m-value: percentage of sample size used in bandwidth determination (when "plth = cv").
pltq = *integer* *(default = 4)*	Modified multifold CV Q-value: percentage of sample size used in bandwidth determination (when "plth = cv").

Final Bandwidth Options

fnlh = *arg* *(default = "cv")*	Final bandwidth method: simple rule-of-thumb ("rot"), robust rule-of-thumb ("rotr"), residual squares criterion ("rsc"), modified multi cross-validation ("cv"), integrated asymptotic mean square error ("mse"), leave-one-out cross-validation ("loo"), nonparametric AIC ("aic"), user-defined ("user").
fnlbw = *arg*	User-defined bandwidth (if "fnlh = user").
fnlhmin = *arg* *(default = 0.1)*	Bandwidth grid search minimum value (if not "fnlh = user").

fnlhmax = *arg* *(default = 1)*	Bandwidth grid search maximum value (if not "fnlh = user").
fnlhlen = *integer* *(default = 100)*	Bandwidth grid search length (if not "fnlh = user").
fnlhinc = *integer* *(default = 10)*	Bandwidth grid search increment step percentage increase (if not "fnlh = user").
fnlhcup = *integer* *(default = 10)*	Stop rule: consecutive increases of objective function before stop (if not "fnlh = user").
fnlm = *arg* *(default = 10)*	Modified multifold CV m-value: percentage of sample size used in bandwidth determination (when "fnlh = cv").
fnlfq = *integer* *(default = 4)*	Modified multifold CV Q-value: percentage of sample size used in bandwidth determination (when "fnlh = cv").

Examples

We consider examples for three equations that estimate FCOEF using UNRATE as the dependent variable, UNRATE(-1 to -2) as independent variables, and LWAGE(-4) as the functional coefficient variable.

```
eq.funcoef(eval=grid, gmin=0, gmax=10, glen=100) unrate unrate(-1)
    unrate(-2) @ lwage(-4)
```

evaluates over a custom uniform grid from 0 to 10 with length 100.

```
eq.funcoef(eval=grid, gvec=vecgrid) unrate unrate(-1) unrate(-2)
    @  lwage(-4)
```

evaluates over a custom grid provided by the values of a workfile vector called VECGRID.

```
eq.funcoef(plyk=3, auxk=5) unrate unrate(-1) unrate(-2) @ lwage(-
    4)
```

estimates using local polynomial fitting with main polynomial degree 3 and auxiliary polynomial degree 5. The latter is employed deriving bias, variance, and bandwidths.

Cross-references

See Chapter 38. "Functional Coefficient Regression," on page 645 of *User's Guide II* for additional discussion of functional coefficients estimation.

funcov	Equation Views

Display functional coefficient covariance functions in graphical form.

For equations estimated using the functional coefficients method.

Syntax

eq_name.**funcov***(options)*

Options

Basic Options

corr	Produce correlations instead of covariances.
nolocalbw	Do not use the existing local pilot bandwidth (if it exists) Note that this option is only available if a local pilot bandwidth has been set previously, either through the estimation step, using `Equation::setpilotbw` (p. 237), or through another view or proc which sets the local bandwidth.
p	Print output.

Pilot Bandwidth Options

If a local pilot bandwidth has exists, all of the pilot bandwidth computation options below will be ignored unless the "nolocalbw" option is specified.

noupdate	Do not update the local pilot bandwidth using the current pilot bandwidth computation.
plth = *arg* *(default = "cv")*	Pilot bandwidth method: simple rule-of-thumb ("rot"), robust rule-of-thumb ("rotr"), residual squares criterion ("rsc"), modified multi cross-validation ("cv"), user-defined ("user").
pltbw = *arg* *(default = 1)*	User-defined bandwidth (if "plth = user").
plthmin = *arg* *(default = 0.1)*	Bandwidth grid search minimum value (if not "plth = user").
plthmax = *arg* *(default = 1)*	Bandwidth grid search maximum value (if not "plth = user").
plthlen = *integer* *(default = 100)*	Bandwidth grid search length (if not "plth = user").
plthinc = *integer* *(default = 10)*	Bandwidth grid search increment step percentage increase (if not "plth = user").
plthcup = *integer* *(default = 10)*	Stop rule: consecutive increases of objective function before stop (not available when "plth = user").

pltm = *arg* *(default = 10)*	Modified multifold CV m-value: percentage of sample size used in bandwidth determination (when "plth = cv").
pltq = *integer* *(default = 4)*	Modified multifold CV Q-value: percentage of sample size used in bandwidth determination (when "plth = cv").
auxk = *integer* *(default = 2)*	Estimation polynomial degree for pilot stage in excess of final stage degree. This number should always be an even positive integer.

Examples

```
eq1.funcov
```

displays the functional covariances computed using the existing local pilot bandwidth if present, or estimating the pilot bandwidth, if not. If estimated, the local pilot bandwidth is updated with the result.

```
eq1.funcov(corr, p)
```

displays and prints a graph of the correlation results.

```
eq1.funcov(nolocalbw, noupdate, auxk=4, plth=rsc)
```

ignores an existing local pilot bandwidth and instead computes a pilot bandwidth using the residual squares criterion and auxiliary degree of 4, and uses it in the covariance calculation. The existing saved local pilot bandwidth is retained.

Cross-references

See Chapter 38. "Functional Coefficient Regression," on page 645 of *User's Guide II* for discussion of functional coefficients estimation. See "Functional Covariance/Correlation Matrix" on page 656 of *User's Guide II* for discussion of functional covariances.

See also `Equation::makefunobj` (p. 191) and `Equation::setpilotbw` (p. 237).

funtest	Equation Views

Perform functional coefficients hypothesis and stability tests.

Test one or more functional coefficients for stability or equality with values in a specified series.

For equations estimated using the functional coefficients method.

Syntax

eq_name.**funtest***(options) arg*

where *arg* is a comma separated list of functional coefficient restrictions for which you wish separately to test.

The coefficient restrictions

coefspec = restriction

where *coefspec* is a coefficient spec consisting of,

- **c**(*k*)
- **@all**

for individual coefficient *k*, and all coefficients, respectively, and *restriction* takes the form:

- *series_name* or *vector*
- *scalar*
- **@constant**

Options

Basic Options

nolocalbw	Do not use the existing local pilot bandwidth (if it exists) Note that this option is only available if a local pilot bandwidth has been set previously, either through the estimation step, using `Equation::setpilotbw` (p. 237), or through another view or proc which sets the local bandwidth.
p	Print output.

Pilot Bandwidth Options

If a local pilot bandwidth has exists, all of the pilot bandwidth computation options below will be ignored unless the "nolocalbw" option is specified.

noupdate	Do not update the local pilot bandwidth using the current pilot bandwidth computation.
plth = *arg* *(default = "cv")*	Pilot bandwidth method: simple rule-of-thumb ("rot"), robust rule-of-thumb ("rotr"), residual squares criterion ("rsc"), modified multi cross-validation ("cv"), user-defined ("user").
pltbw = *arg* *(default = 1)*	User-defined bandwidth (if "plth = user").
plthmin = *arg* *(default = 0.1)*	Bandwidth grid search minimum value (if not "plth = user").

plthmax = *arg* *(default = 1)*	Bandwidth grid search maximum value (if not "plth = user").
plthlen = *integer* *(default = 100)*	Bandwidth grid search length (if not "plth = user").
plthinc = *integer* *(default = 10)*	Bandwidth grid search increment step percentage increase (if not "plth = user").
plthcup = *integer* *(default = 10)*	Stop rule: consecutive increases of objective function before stop (not available when "plth = user").
pltm = *arg* *(default = 10)*	Modified multifold CV m-value: percentage of sample size used in bandwidth determination (when "plth = cv").
pltq = *integer* *(default = 4)*	Modified multifold CV Q-value: percentage of sample size used in bandwidth determination (when "plth = cv").
auxk = *integer* *(default = 2)*	Estimation polynomial degree for pilot stage in excess of final stage degree. This number should always be an even positive integer.

Examples

```
eq1.funtest @all=10, c(2)=2, c(3)=y
```

performs three separate tests for the restrictions: all of the coefficients are constant and equal to 10, c(2) is a constant equal to 2, and c(3) is equal to the series Y using an existing local pilot bandwidth if present, or estimating the pilot bandwidth, if not. If estimated, the local pilot bandwidth is updated with the result.

```
eq1.funtest(nolocalbw, noupdate, auxk=4, plth=rsc) @all=3, c(2)=2,
    c(3)=y
```

ignores an existing local pilot bandwidth and instead computes a pilot bandwidth using the residual squares criterion and auxiliary degree of 4, and uses it in the test calculation. The existing saved local pilot bandwidth is retained.

Cross-references

See Chapter 38. "Functional Coefficient Regression," on page 645 of *User's Guide II* for discussion of functional coefficients estimation. See "Stability and Significance Tests" on page 663 of *User's Guide II* for discussion of functional covariances.

See also `Equation::makefunobj` (p. 191) and `Equation::setpilotbw` (p. 237).

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 27. “ARCH and GARCH Estimation,” on page 273 of *User’s Guide II*.

glm	Equation Methods

Estimate a Generalized Linear Model (GLM).

Syntax

eq_name.**glm**(*options*) *spec*

List the `glm` keyword, followed by the dependent variable and a list of the explanatory variables, or an explicit linear expression.

If you enter an explicit *linear* specification such as "Y = C(1) + C(2)*X", the response variable will be taken to be the variable on the left-hand side of the equality ("Y") and the linear predictor will be taken from the right-hand side of the expression ("C(1) + C(2)*X").

Offsets may be entered directly in an explicit linear expression or they may be entered as using the "offset = " option.

Specification Options

family = *arg* *(default* = "normal"*)*	Distribution family: Normal ("normal"), Poisson ("poisson"), Binomial Count ("binomial"), Binomial Proportion ("binprop"), Negative Binomial ("negbin"), Gamma ("gamma"), Inverse Gaussian ("igauss"), Exponential Mean ("emean)", Power Mean ("pmean"), Binomial Squared ("binsq"). The Binomial Count, Binomial Proportion, Negative Binomial, and Power Mean families all require specification of a distribution parameter:
n = *arg (default* = 1*)*	Number of trials for Binomial Count ("family = binomial") or Binomial Proportions ("family = binprop") families.
fparam = *arg*	Family parameter value for Negative Binomial ("family = negbin") and Power Mean ("family = pmean") families.
link = *arg* *(default* = "identity"*)*	Link function: Identity ("identity"), Log ("log"), Log Compliment ("logc"), Logit ("logit"), Probit ("probit"), Log-log ("loglog"), Complementary Log-log ("cloglog"), Reciprocal ("recip"), Power ("power"), Box-Cox ("boxcox"), Power Odds Ratio ("opow"), Box-Cox Odds Ratio ("obox"). The Power, Box-Cox, Power Odds Ratio, and Box-Cox Odds Ratio links all require specification of a link parameter specified using "lparam = ".
lparam = *arg*	Link parameter for Power ("link = power"), Box-Cox ("link = boxcox"), Power Odds Ratio ("link = opow") and Box-Cox Odds Ratio ("link = obox") link functions.
offset = *arg*	Offset terms.

disp = *arg*	Dispersion estimator: Pearson χ^2 statistic ("pearson"), deviance statistic ("deviance"), unit ("unit"), user-specified ("user"). The default is family specific: "unit" for Binomial Count, Binomial Proportion, Negative Binomial, and Poison, and "pearson" for all others. The "deviance" option is only offered for families in the exponential family of distributions (Normal, Poisson, Binomial Count, Binomial Proportion, Negative Binomial, Gamma, Inverse Gaussian).
dispval = *arg*	User-dispersion value (if "disp = user").
fwgts = *arg*	Frequency weights.
w = *arg*	Weight series or expression.
wtype = *arg* *(default* = "istdev*")*	Weight specification type: inverse standard deviation ("istdev"), inverse variance ("ivar"), standard deviation ("stdev"), variance ("var").
wscale = *arg*	Weight scaling: EViews default ("eviews"), average ("avg"), none ("none"). The default setting depends upon the weight type: "eviews" if "wtype = istdev", "avg" for all others.

In addition to the specification options, there are options for estimation and covariance calculation.

Additional Options

optmethod = *arg*	Optimization method: "bfgs" (BFGS); "newton" (Newton-Raphson), "opg" or "bhhh" (OPG or BHHH), "fisher" (IRLS – Fisher Scoring), "legacy" (EViews legacy). Newton-Raphson is the default method.
optstep = *arg*	Step method: "marquardt" (Marquardt); "dogleg" (Dogleg); "linesearch" (Line search). Marquardt is the default method.
estmeth = *arg* *(default* = "marquardt")	Legacy estimation algorithm: Quadratic Hill Climbing ("marquardt"), Newton-Raphson ("newton"), IRLS - Fisher Scoring ("irls"), BHHH ("bhhh"). (Applicable when "optmethod = legacy".)
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 estimator coefficient vector as starting values (see also `param` (p. 564) in the *Command and Programming Reference*).
s = *number*	Specify a number between zero and one to determine starting values as a fraction of EViews default values (out of range values are set to "*s = 1*").
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
preiter = *arg* *(default = 0)*	Number of IRLS pre-iterations to refine starting values (only available for non-IRLS estimation).
cov = *arg*	Covariance method: "ordinary" (default method based on inverse of the estimated information matrix), "huber" or "white" (Huber-White sandwich method), "glm" (GLM method), "cr" (cluster robust).
covinfo = *arg*	Information matrix method: "opg" (OPG); "hessian" (observed Hessian), "fisher" (expected Hessian). (Applicable when "optmethod = " not equal to "legacy".
nodf	Do not degree-of-freedom correct the coefficient covariance estimate.(For non-cluster robust methods).
covlag = *arg* *(default = 1)*	Whitening lag specification: *integer* (user-specified lag value), "a" (automatic selection). Applicable where "cov = hac".
covinfosel = *arg* *(default = "aic")*	Information criterion for automatic selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn) (if "lag = a"). For settings where "cov = hac, covlag = a".
covmaxlag = *integer*	Maximum lag-length for automatic selection *(optional)* (if "lag = a"). The default is an observation-based maximum of $T^{1/3}$. For settings where "cov = hac, covlag = a".

covkern = *arg* *(default* = "bart"*)*	Kernel shape: "none" (no kernel), "bart" (Bartlett, *default*), "bohman" (Bohman), "daniell" (Daniel), "parzen" (Parzen), "parzriesz" (Parzen-Riesz), "parzgeo" (Parzen-Geometric), "parzcauchy" (Parzen-Cauchy), "quadspec" (Quadratic Spectral), "trunc" (Truncated), "thamm" (Tukey-Hamming), "thann" (Tukey-Hanning), "tparz" (Tukey-Parzen). For settings where "cov = hac".
covbw = *arg* *(default* = "fixednw"*)*	Kernel Bandwidth: "fixednw" (Newey-West fixed), "andrews" (Andrews automatic), "neweywest" (Newey-West automatic), *number* (User-specified bandwidth). For settings where "cov = hac" and "covkern = " is specified.
covnwlag = *integer*	Newey-West lag-selection parameter for use in nonparametric kernel bandwidth selection (if "covbw = newey-west"). For settings where "cov = hac" and "covkern = " is specified.
covbwoffset = *number*	Apply offset to automatically selected bandwidth. For settings where "cov = hac", "covkern = " is specified, and "covbw = " is not user-specified.
covbwint	Use integer portion of kernel bandwidth. For settings where "cov = hac" and "covkern = " is specified.
crtype = *arg (default "cr1")*	Cluster robust weighting method: "cr0" (no finite sample correction), "cr1" (finite sample correction), when "cov = cr".
crname = *arg*	Cluster robust series name, when "cov = cr".
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
prompt	Force the dialog to appear from within a program.
p	Print results.

Examples

```
equation eqstrike.glm(link=log) numb c ip feb
```

estimates a normal regression model with exponential mean.

```
equation eqbinom.glm(family=binomial, n=total) disease c snore
```

estimates a binomial count model with default logit link where TOTAL contains the number of binomial trials and DISEASE is the number of binomial successes. The specification

```
equation eqbinom.glm(family=binprop, n=total, cov=huber, nodf)
    disease/total c snore
```

estimates the same specification in proportion form, and computes the coefficient covariance using the Huber-White sandwich with no d.f. correction.

```
equation eqprate.glm(family=binprop, disp=pearson) prate mprate
    log(totemp) log(totemp)^2 age age^2 sole
```

estimates a binomial proportions model with default logit link, but computes the coefficient covariance using the GLM scaled covariance with dispersion computed using the Pearson Chi-square statistic.

```
equation eqprate.glm(family=binprop, link=probit, cov=huber) prate
    mprate log(totemp) log(totemp)^2 age age^2 sole
```

estimates the same basic specification, but with a probit link and Huber-White standard errors.

```
equation testeq.glm(family=poisson, offset=log(pyears)) los hmo
    white type2 type3 c
```

estimates a Poisson specification with an offset term LOG(PYEARS).

Cross-references

See Chapter 32. “Generalized Linear Models,” beginning on page 485 of *User’s Guide II* for discussion.

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

Non-Panel GMM Options

Basic GMM Options

nocinst	Do not include automatically a constant as an instrument.
method = *keyword*	Set the weight updating method. *keyword* should be one of the following: "nstep" (N-Step Iterative, or Sequential N-Step Iterative, default), "converge" (Iterate to Convergence or Sequential Iterate to Convergence), "simul" (Simultaneous Iterate to Convergence), "oneplusone" (One-Step Weights Plus One Iteration), or "cue" (Continuously Updating".
gmmiter = *integer*	Number of weight iterations. Only applicable if the "method = nstep" option is set.
w = *arg*	Weight series or expression.
wtype = *arg* *(default* = "istdev"*)*	Weight specification type: inverse standard deviation ("istdev"), inverse variance ("ivar"), standard deviation ("stdev"), variance ("var").
wscale = *arg*	Weight scaling: EViews default ("eviews"), average ("avg"), none ("none"). The default setting depends upon the weight type: "eviews" if "wtype = istdev", "avg" for all others.
m = *integer*	Maximum number of iterations.
s	Use the current coefficient values in estimator coefficient vector as starting values for equations specified by list (see also `param` (p. 564) of the *Command and Programming Reference*).

s = *number*	Determine starting values for equations specified by list. Specify a number between zero and one representing the fraction of preliminary TSLS 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". Does not apply to coefficients for AR and MA terms which are instead set to EViews determined default values.
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.
l = *number*	Set maximum number of iterations on the first-stage iteration to get the one-step weighting matrix.
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
numericderiv / -numericderiv	[Do / do not] use numeric derivatives only. If omitted, EViews will follow the global default.
fastderiv / -fastderiv	[Do / do not] use fast derivative computation. If omitted, EViews will follow the global default.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
prompt	Force the dialog to appear from within a program.
p	Print results.

Estimation Weighting Matrix Options

instwgt = *keyword*	Set the estimation weighting matrix type. *Keyword* should be one of the following: "tsls" (two-stage least squares), "white" (White diagonal matrix), "hac" (Newey-West HAC, default) or "user" (user defined).
instwgtmat = *name*	Set the name of the user-defined estimation weighting matrix. Only applicable if the "instwgt = user" option is set.
instlag = *arg* *(default* = 1*)*	Whitening Lag specification: *integer* (user-specified lag value), "a" (automatic selection).
instinfosel = *arg* *(default* = "aic"*)*	Information criterion for automatic whitening lag selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn) (if "instlag = a").

instmaxlag = *integer*	Maximum lag-length for automatic selection *(optional)* (if "instlag = a"). The default is an observation-based maximum of $T^{1/3}$.
instkern = *arg* *(default* = "bart"*)*	Kernel shape: "none" (no kernel), "bart" (Bartlett, *default*), "bohman" (Bohman), "daniell" (Daniell), "parzen" (Parzen), "parzriesz" (Parzen-Riesz), "parzgeo" (Parzen-Geometric), "parzcauchy" (Parzen-Cauchy), "quadspec" (Quadratic Spectral), "trunc" (Truncated), "thamm" (Tukey-Hamming), "thann" (Tukey-Hanning), "tparz" (Tukey-Parzen).
instbw = *arg* *(default* = "fixednw"*)*	Kernel Bandwidth: "fixednw" (Newey-West fixed), "andrews" (Andrews automatic), "neweywest" (Newey-West automatic), *number* (User-specified bandwidth).
instnwlag = *integer*	Newey-West lag-selection parameter for use in nonparametric bandwidth selection (if "instbw = neweywest").
instbwoffset = *integer* *(default* = 0*)*	Apply integer offset to bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").
instbwint	Use integer portion of bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").

Covariance Options

cov = *keyword*	Covariance weighting matrix type *(optional)*: "updated" (estimation updated), "tsls" (two-stage least squares), "white" (White diagonal matrix), "hac" (Newey-West HAC), "wind" (Windmeijer) "cr" (cluster robust). or "user" (user defined). The default is to use the estimation weighting matrix.
nodf	Do not perform degree of freedom corrections in computing coefficient covariance matrix. The default is to use degree of freedom corrections. (For non-cluster robust methods).
covlag = *arg* *(default* = 1*)*	Whitening lag specification: *integer* (user-specified lag value), "a" (automatic selection).
covinfosel = *arg* *(default* = "aic"*)*	Information criterion for automatic selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn) (if "lag = a").
covmaxlag = *integer*	Maximum lag-length for automatic selection *(optional)* (if "lag = a"). The default is an observation-based maximum of $T^{1/3}$.

covkern = *arg* *(default* = "bart"*)*	Kernel shape: "none" (no kernel), "bart" (Bartlett, *default*), "bohman" (Bohman), "daniell" (Daniel), "parzen" (Parzen), "parzriesz" (Parzen-Riesz), "parzgeo" (Parzen-Geometric), "parzcauchy" (Parzen-Cauchy), "quadspec" (Quadratic Spectral), "trunc" (Truncated), "thamm" (Tukey-Hamming), "thann" (Tukey-Hanning), "tparz" (Tukey-Parzen).
covbw = *arg* *(default* = "fixednw"*)*	Kernel Bandwidth: "fixednw" (Newey-West fixed), "andrews" (Andrews automatic), "neweywest" (Newey-West automatic), *number* (User-specified bandwidth).
covnwlag = *integer*	Newey-West lag-selection parameter for use in nonparametric kernel bandwidth selection (if "covbw = neweywest").
covbwoffset = *integer* *(default* = 0*)*	Apply integer offset to bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").
covbwint	Use integer portion of bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").
crtype = *arg (default "cr1")*	Cluster robust weighting method: "cr0" (no finite sample correction), "cr1" (finite sample correction), when "cov = cr".
crname = *arg*	Cluster robust series name, when "cov = cr".

Panel GMM Options

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 weighs ("wgt = cxdiag"), period diagonal weights ("wgt = perdiag").

gmm = *arg*	GMM weighting: 2SLS ("gmm = 2sls"), White period system covariances (Arellano-Bond 2-step/n-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.
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").
keepwgts	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).
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
iter = *arg* *(default* = "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").
s	Use the current coefficient values in estimator coefficient vector as starting values for equations specified by list (see also `param` (p. 564) of the *Command and Programming Reference*).

s = *number*	Determine starting values for equations specified by list. Specify a number between zero and one representing the fraction of preliminary TSLS estimates computed without AR 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". Does not apply to coefficients for AR terms which are instead set to EViews determined default values.
m = *integer*	Maximum number of iterations.
c = *number*	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
l = *number*	Set maximum number of iterations on the first-stage iteration to get the one-step weighting matrix.
unbalsur	Compute SUR factorization in unbalanced data using the subset of available observations for a cluster.
numericderiv / -numericderiv	[Do / do not] use numeric derivatives only. If omitted, EViews will follow the global default.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
prompt	Force the dialog to appear from within a program.
p	Print results.

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:

```
gmmc.gmm(instwgt=white,gmmiter=2,nodf) cons c y y(-1) w @ c p(-1)
    k(-1) x(-1) tm wg g t
```

estimates the Klein equation for consumption using GMM with a White diagonal weighting matrix (two steps and no degree of freedom correction). The command:

```
gmmi.gmm(method=cue,instwgt=hac,instlag=1,instkern=thann,instbw=an
    drews,nodf) i c y y(-1) k(-1) @  c p(-1) k(-1) x(-1) tm wg g t
```

estimates the Klein equation for investment using a Newey-West HAC weighting matrix, with pre-whitening with 1 lag, a Tukey-Hanning kernel and the Andrews automatic bandwidth routine. The estimation is performed using continuously updating weight iterations.

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 "Generalized Method of Moments" on page 103 and Chapter 55. "Panel Estimation," on page 1321 of *User's Guide II* for discussion of the various GMM estimation techniques.

See also `Equation::tsls` (p. 254).

grads	Equation Views

Gradients of the objective function.

Displays the gradients of the objective function. Evaluating the gradients at current coefficient values allows you to examine the behavior of the objective function at starting values.

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.

You may optionally choose to display the results in tabular or graphical form. 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

t	Display spreadsheet view of the values of the gradients of the objective function with respect to the coefficients evaluated at each observation.
g	Display multiple graph showing 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. 117), `Equation::makederivs` (p. 190), and `Equation::makegrads` (p. 194).

heckit	Equation Methods

Estimate a selection equation using the Heckman ML or 2-step method.

Syntax

equation_name.**heckit**(*options*) *response_eqn @ selection_eqn*

The response equation should be the dependent variable followed by a list of regressors. The selection equation should be a binary dependent variable followed by a list of regressors.

Options

General Options

2step	Use the Heckman 2-step estimation method. Note that this option is incompatible with the maximum likelihood options below.
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
prompt	Force the dialog to appear from within a program.
p	Print the estimation results.

ML Options

Note these options are not available if the "2step" option, above, is used.

optmethod = *arg*	Optimization method: "bfgs" (BFGS); "newton" (Newton-Raphson), "opg" or "bhhh" (OPG or BHHH), "legacy" (EViews legacy). Newton-Raphson is the default method.
optstep = *arg*	Step method: "marquardt" (Marquardt); "dogleg" (Dog-leg); "linesearch" (Line search). Marquardt is the default method.
cov = *arg*	Covariance method: "ordinary" (default method based on inverse of the estimated information matrix), "huber" or "white" (Huber-White sandwich methods).,
covinfo = *arg*	Information matrix method: "opg" (OPG); "hessian" (observed Hessian). (Applicable when non-legacy "optmethod = ".)
m = *integer*	Set maximum number of iterations.
c = *number*	Set convergence criteria.
numericderiv / -numericderiv	[Do / do not] use numeric derivatives only. If omitted, EViews will follow the global default.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
s = *number*	Scale EViews' starting values by *number*.
r	Use Newton-Raphson optimizer.
b	Use BHHH optimizer.

Examples

```
wfopen http://www.stern.nyu.edu/~wgreene/Text/Edition7/TableF5-
    1.txt
equation eq01.heckit ww c ax ax^2 we cit @ lfp c wa wa^2 faminc we
    (k618+kl6)>0
equation eq02.heckit(2step) ww c ax ax^2 we cit @ lfp c wa wa^2
    faminc we (k618+kl6)>0
```

This example replicates the Heckman Selection example given in Greene (2008, page 888), which uses data from the Mroz (1987) study to estimate a selection model. The first line of this example downloads the data set, the second line creates an equation object and estimates it using the default maximum likelihood estimation method of Heckman Selection, which replicates the first pane of Table 24.3 in Greene. The third line estimates the same model, using the two-step approach, which replicates the second pane of Table 24.3.

Cross-references

See "Heckman Selection Model" on page 465 of *User's Guide II* for discussion.

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

type = *keyword*	where *keyword* is either "BPG" (Breusch-Pagan-Godfrey - *default*), "Harvey", "Glejser", "ARCH", or "White".
c	include cross terms for White test.
lags = *int*	set number of lags to use for ARCH test. (Only applies when type = "ARCH".
prompt	Force the dialog to appear from within a program.

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:

@regs	include every regressor from the original equation.
@grads	include every gradient in the original equation (non-linear equations only).
@grad(*int*)	include the *int*-th gradient.
@white(*key*)	include white-style regressors (the cross-product of the regressor list, or the gradient list if non-linear). *key* may be on of the following keywords: "@regs" (include every regressor from the original equation), "@drop(*variables*)" (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.e.* levels, squares, and cross-products).
@arch(*lag_structure*)	include an ARCH specification with the number of lags specified by *lag_structure*. If *lag_structure* 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.
@uw(*variables*)	include unweighted variables (only applicable in a weighted original equation).

Examples

```
eq1.hettest(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 225 of *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

```
eq1.hist
```

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

Cross-references

See "Histogram and Stats" on page 450 of *User's Guide I* for a discussion of the descriptive statistics reported in the histogram view.

icgraph	Equation Views

Display a graph of the selection criteria for the top 20 models observed as part of model selection during estimation.

This view is only available for equations estimated using the ARDL or TAR methods.

Syntax

equation_name.**icgraph**

Examples

```
wfopen http://www.stern.nyu.edu/~wgreene/Text/Edition7/TableF5-
    2.txt
equation eq01.ardl(deplags=8, reglags=8) log(realcons)
    log(realgdp) @ @expand(@quarter, @droplast)
show eq01.icgraph
```

This example uses data from Greene (2008, page 685), containing quarterly US macroeconomic variables between 1950 and 2000. The first line of this example downloads the data set, the second line creates an equation object and estimates an ARDL model with the log of real consumption as the dependent variable, and the log of real GDP as a dynamic regressor. Quarterly dummy variables are included as static regressors. Automatic model selection is used.

The final line of code displays a graph showing the Akaike information criteria (the default selection method) for each of the models estimates.

Cross-references

See Chapter 29. "ARDL and Quantile ARDL," on page 341 and "Discrete Threshold Regression," beginning on page 555 of *User's Guide II* for further discussion.

ictable	Equation Views

Display a table of the log-likelihood and selection criteria for the top 20 models observed as part of model selection during estimation.

This view is only available for equations estimated using the ARDL or TAR methods.

Syntax

equation_name.ictable

Examples

```
wfopen http://www.stern.nyu.edu/~wgreene/Text/Edition7/TableF5-
    2.txt
equation eq01.ardl(deplags=8, reglags=8) log(realcons)
    log(realgdp) @ @expand(@quarter, @droplast)
show eq01.ictable
```

This example uses data from Greene (2008, page 685), containing quarterly US macroeconomic variables between 1950 and 2000. The first line of this example downloads the data set, the second line creates an equation object and estimates an ARDL model with the log of real consumption as the dependent variable, and the log of real GDP as a dynamic regressor. Quarterly dummy variables are included as static regressors. Automatic model selection is used.

The final line of code displays a table showing the log-likelihood value, Akaike information criteria, Schwarz information criteria, the Hannan-Quinn criteria and the adjusted R-squared of the top 20 models.

Cross-references

See Chapter 29. "ARDL and Quantile ARDL," on page 341 and "Discrete Threshold Regression" on page 555 of *User's Guide II* for further discussion.

infbetas	Equation Views

Scaled difference in the estimated betas for influence statistics.

DFBETAS are the scaled difference in the estimated betas between the original equation and an equation estimated without that observation.

Syntax

equation_name.**infbetas***(options) [base_name]*

where *base_name* is an optional naming suffix used to store the DFBETAS into the workfile.

Options

t	Show a table of the statistics (the *default* is to display a graph view of the specified statistics).
rows = *key*	The number of observations/rows to display in the table, where *key* can be either "50", "100" (default), "150", or "200".
g = *arg*	*arg* is the name of an object in which the graph output will be saved.
prompt	Force the dialog to appear from within a program.

Examples

```
equation eq1.ls y c x z
eq1.infbetas
```

displays a graph of the DFBETAS corresponding to the coefficients for C, X, and Z.

```
eq1.infbetas(t) out
```

will display a table showing the first 150 rows of DFBETAs in table form and saves the results in the series COUT, XOUT and ZOUT.

Cross-references

See also "Influence Statistics" on page 259 of *User's Guide II*.

See also `Equation::infstats` (p. 172).

infstats	Equation Views

Influence statistics.

Displays influence statistics to discover influential observations, or outliers.

Syntax

equation_name.**infstats***(options)*

equation_name.**infstats***(options) stats_list [@ save_names]*

If no *stats_list* is provided all of the statistics will be displayed. *save_names* is an optional list of names for storing the statistics into series in the workfile. The *save_names* should match the order in which the keywords in *stats_list* are entered.

Options

t	Show a table of the statistics (the *default* is to display a graph view of the specified statistics).
rows = *key*	The number of observations/rows to display in the table, where *key* can be either "50", "100" (default), "150", or "200".
sort = *key*	Sort order for the table, where *key* can be "r" (Residual - default), "rs" (RStudent), "df" (DFFITS), "dr" (Dropped Residual), "cov" (COVRATIO), "h" (diagonal elements of the hat matrix).
sortdisp	Display the table by the sort order rather than by the observation order.
prompt	Force the dialog to appear from within a program.

The *stats_list* parameter is a list of keywords indicating which statistics to display. It may take on the values:

rstudent	The studentized residual: the *t*-statistic on a dummy variable that is equal to 1 on that observation only.
dffits	The scaled difference in fitted values.
drresid	Dropped residual: the estimated residual for that observation had the equation been run without that observation.
covratio	The ratio of the covariance matrix of the coefficients with and without that observation.
hatmatrix	Diagonal elements of the hat matrix: $x_i'(X'X)^{-1}x_i$

Examples

```
eq1.infstats(t, rows=150, sort=rs) rstudent covratio dffits @
    rstuds covs
```

will display a table showing the 150 largest RSTUDENT statistics, along with the corresponding COVRATIO and DFFITS statistics. It will save the RSTUDENT and COVRATIO statistics into the series in the workfile named RSTUDS and COVS, respectively.

Cross-references

See also "Influence Statistics" on page 259 of *User's Guide II*.

See also `Equation::infbetas` (p. 171).

instsum	Equation Views

Shows a summary of the equation instruments.

Changes the view of the equation to the Instrument Summary view. Note this is only available for equations estimated by TSLS, GMM, or LIML.

Syntax

eq_name.**instsum**

Examples

```
equation eq1.tsls sales c adver lsales @ gdp unemp int
e1.instsum
```

creates an equation E1 and estimates it via two-stage least squares, then shows a summary of the instruments used in estimation.

Cross-references

See "Instrument Summary" on page 114 of *User's Guide II* for discussion.

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 *text*.
s	Sets the source field to *text*.

u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User's Guide I* for a discussion of labels.

See also `Equation::displayname` (p. 122).

lambdacoefs	Equation Views

Display the spreadsheet of the matrix of coefficient values along the lambda path.

This view is only available for equations estimated with elastic net, ridge regression, Lasso, and variable selection using Lasso.

Rows will contain coefficients for a given lambda; columns contain coefficients for specific variables.

The row displaying the model selected optimal lambda will be highlighted.

Only coefficients that have non-zero values for at least one lambda in the path will be displayed.

Syntax

eq_name.**lambdacoefs**(*options*)

Options

p	Print output.

Examples

Consider the estimated elastic net equation

```
equation my_eq.enet(xtrans=none, lambdaratio=.0001,
    cvseed=513255899) lpsa c lcavol_s lweight_s age_s lbph_s svi_s
    lcp_s gleason_s pgg45_s
```

Then the command

```
my_eq.lambdacoefs
```

displays a table of the anywhere non-zero coefficients, with rows corresponding to values of lambda in the path, and columns corresponding to different variables. The row with the cross-validation selected optimal value of lambda will be highlighted.

Cross-references

For further discussion, see Chapter 37. "Elastic Net and Lasso" of *User's Guide II*.

See also `Equation::coefpath` (p. 95) and `Equation::lambdapath` (p. 178).

The data underlying this table are available via the data member `@lambdacoefs`.

lambdaest	Equation Views

Display the table showing various values associated with estimation along the lambda path.

This view is only available for equations estimated with elastic net, ridge regression, Lasso, and variable selection using Lasso.

Display a table showing the number of non-zero coefficients, estimation objective, sums-of-squared residuals penalty, and where applicable, the L1-norm and the L2-norm penalty portions of the objective associated with each lambda in the path.

The row displaying the model selected optimal lambda will be highlighted.

Syntax

eq_name.**lambdaest**(*options*)

Options

p	Print output.

Examples

Consider the estimated elastic net equation

```
equation my_eq.enet(xtrans=none, lambdaratio=.0001,
    cvseed=513255899) lpsa c lcavol_s lweight_s age_s lbph_s svi_s
    lcp_s gleason_s pgg45_s
```

Then the command

```
my_eq.lambdaest
```

displays the table showing various estimation statistics associated with each lambda in the path.

Cross-references

For further discussion, see Chapter 37. "Elastic Net and Lasso" of *User's Guide II*.

See also `Equation::coefpath` (p. 95), `Equation::lambdafit` (p. 177), and `Equation::lambdapath` (p. 178).

The data underlying this table are available via the data member `@lambdaest`.

lambdafit	Equation Views

Display the table showing various fit statistics associated with estimates along the lambda path.

This view is only available for equations estimated with elastic net, ridge regression, Lasso, and variable selection using Lasso.

Display a table showing the number of non-zero coefficients, R-squared, adjusted R-squared, standard error of the regression, and sums-of-squared residuals associated with each of the lambda in the path.

The row displaying the model selected optimal lambda will be highlighted.

Syntax

eq_name.**lambdafit**(*options*)

Options

p	Print output.

Examples

Consider the estimated elastic net equation

```
equation my_eq.enet(xtrans=none, lambdaratio=.0001,
    cvseed=513255899) lpsa c lcavol_s lweight_s age_s lbph_s svi_s
    lcp_s gleason_s pgg45_s
```

Then the command

```
my_eq.lambdafit
```

displays the table showing various fit statistics associated with each lambda in the path.

Cross-references

For further discussion, see Chapter 37. "Elastic Net and Lasso" of *User's Guide II*.

See also `Equation::coefpath` (p. 95), `Equation::lambdaest` (p. 176), and `Equation::lambdapath` (p. 178).

The data underlying this table are available via the data member `@lambdafit`.

lambdapath	Equation Views

Display graphs of lambda against various fit and estimation measures.

This view is only available for equations estimated with elastic net, ridge regression, Lasso, and variable selection using Lasso.

You may plot lambda against the paths of: the number of non-zero coefficients, model selection objective, R-squared and adjusted R-square fit statistics, standard error of the regression, sum-of-squared residuals, L1-norm coefficient penalty, L2-norm squared coefficient penalty, and the estimation objective.

A vertical line will be included to identify the selected optimal lambda.

By default, EViews will display a spool object containing all of the plots. You may use the "type = " option to produce a specific graph.

Syntax

eq_name.**lambdapaths**(*options*)

Options

type = *arg*	Graph of the log lambda against the path of: "non-zero" (number of non-zero coefficients), "model" (model selection objective), "fit" (R-squared and adjusted R-squared fit statistics), "se" (standard error of regression), "ssr" (sum-of-squared residuals), "l1" (L1 coefficient penalty, if applicable), "l2" (L2-squared coefficient penalty, if applicable), "estobj" (estimation objective). If "type = " is not provided, EViews will display the spool object all of the graphs.
p	Print output.

Examples

Consider the estimated elastic net equation

```
equation my_eq.enet(xtrans=none, lambdaratio=.0001,
    cvseed=513255899) lpsa c lcavol_s lweight_s age_s lbph_s svi_s
    lcp_s gleason_s pgg45_s
```

Then the command

```
my_eq.lambdapath
```

displays estimates a spool containing graphs of log lambda plotted against paths of the number of non-zero coefficients, fit measures, and estimation measures.

```
my_eq.lambapath(type=nonzero, p)
```

displays and prints a single graph of log lambda plotted against the number of non-zero coefficients, while

```
my_eq.lambapath(type=fit)
```

plots log lambda against the paths of the R-squared and adjusted R-squared.

Cross-references

For further discussion, see Chapter 37. "Elastic Net and Lasso" in *User's Guide II*.

See also `Equation::coefpath` (p. 95), `Equation::lambdaest` (p. 176), and `Equation::lambdafit` (p. 177).

The data underlying these graphs are available via the data members `@lambdafit`, and `@lambdaest`.

liml	Equation Methods

Limited Information Maximum Likelihood and K-class Estimation.

Syntax

eq_name.**liml***(options) y c x1 [x2 x3 ...] @ z1 [z2 z3 ...]*

eq_name.**liml***(options) specification @ z1 [z2 z3 ...]*

To use the `liml` 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, unless the `noconst` option is specified.

Options

noconst	Do not include a constant in the instrumental list. Without this option, a constant will always be included as an instrument, even if not specified explicitly.
w = *arg*	Weight series or expression.
wtype = *arg* *(default* = "istdev*")*	Weight specification type: inverse standard deviation ("istdev"), inverse variance ("ivar"), standard deviation ("stdev"), variance ("var").
wscale = *arg*	Weight scaling: EViews default ("eviews"), average ("avg"), none ("none"). The default setting depends upon the weight type: "eviews" if "wtype = istdev", "avg" for all others.
kclass = *number*	Set the value of k in the K-class estimator. If omitted, LIML is performed, and k is calculated as part of the estimation procedure.
se = *arg* *(default* = "iv*")*	Set the standard-error calculation type: IV based ("se = iv"), K-Class based ("se = kclass"), Bekker ("se = bekk"), or Hansen, Hausman, and Newey ("se = hhn").
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.
numericderiv / -numericderiv	[Do / do not] use numeric derivatives only. If omitted, EViews will follow the global default.
fastderiv / -fastderiv	[Do / do not] use fast derivative computation. If omitted, EViews will follow the global default. Available only for legacy estimation ("optmeth = legacy").
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
prompt	Force the dialog to appear from within a program.
p	Print estimation results.

Examples

```
equation eq1.liml gdp c cpi inc @ lw lw(-1)
```

creates equation EQ1 and calculates a LIML estimation of GDP on a constant, CPI, and INC, using a constant, LW, and LW(-1) as instruments.

```
e1.liml(kclass=2)
```

estimates the same equation, but this time via K-Class estimation, with K = 2.

Cross-references

See also "Limited Information Maximum Likelihood and K-Class Estimation" on page 99 of *User's Guide II* for discussion.

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. 79).

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*

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, a D fractional differencing term, 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

Non-Panel LS Options

indicator	Include indicator saturation detection as part of estimation routine.
w = *arg*	Weight series or expression. Note: *we recommend that, absent a good reason, you employ the default settings* **Inverse std. dev. weights** *("wtype = istdev") with* **EViews default** *scaling ("wscale = eviews") for backward compatibility with versions prior to EViews 7.*
wtype = *arg* (*default* = "istdev")	Weight specification type: inverse standard deviation ("istdev"), inverse variance ("ivar"), standard deviation ("stdev"), variance ("var").
wscale = *arg*	Weight scaling: EViews default ("eviews"), average ("avg"), none ("none"). The default setting depends upon the weight type: "eviews" if "wtype = istdev", "avg" for all others.
z	Turn off backcasting in ARMA models where "arma = cls".
optmethod = *arg*	Optimization method for nonlinear least squares and ARMA: "bfgs" (BFGS); "newton" (Newton-Raphson), "opg" or "bhhh" (OPG or BHHH), "kohn" (Kohn-Ansley for ARMA estimated by ML or GLS), or "legacy" (EViews legacy for nonlinear least squares and ARMA estimated by CLS). Gauss-Newton is the default method.
optstep = *arg*	Step method for nonlinear least squares and ARMA: "marquardt" (Marquardt); "dogleg" (Dogleg); "linesearch" (Line search). Marquardt is the default method.
cov = *arg*	Covariance method: "ordinary" (default method based on inverse of the estimated information matrix), "huber" or "white" (Huber-White sandwich method available for nonlinear least squares or ARMA estimated by CLS), "hac" (Newey-West HAC, available for nonlinear least squares or ARMA estimated by CLS)..

covinfo = *arg*	Information matrix method: “opg” (OPG); “hessian” (observed Hessian). (Applicable when non-legacy “optmethod = ”.)
nodf	Do not perform degree of freedom corrections in computing coefficient covariance matrix. The default is to use degree of freedom corrections.
covlag = *arg* *(default* = 1*)*	Whitening lag specification: *integer* (user-specified lag value), “a” (automatic selection).
covinfosel = *arg* *(default* = “aic”*)*	Information criterion for automatic selection: “aic” (Akaike), “sic” (Schwarz), “hqc” (Hannan-Quinn) (if “lag = a”).
covmaxlag = *integer*	Maximum lag-length for automatic selection *(optional)* (if “lag = a”). The default is an observation-based maximum of $T^{1/3}$.
covkern = *arg* *(default* = “bart”*)*	Kernel shape: “none” (no kernel), “bart” (Bartlett, *default*), “bohman” (Bohman), “daniell” (Daniel), “parzen” (Parzen), “parzriesz” (Parzen-Riesz), “parzgeo” (Parzen-Geometric), “parzcauchy” (Parzen-Cauchy), “quadspec” (Quadratic Spectral), “trunc” (Truncated), “thamm” (Tukey-Hamming), “thann” (Tukey-Hanning), “tparz” (Tukey-Parzen).
covbw = *arg* *(default* = “fixednw” *)*	Kernel Bandwidth: “fixednw” (Newey-West fixed), “andrews” (Andrews automatic), “neweywest” (Newey-West automatic), *number* (User-specified bandwidth).
covnwlag = *integer*	Newey-West lag-selection parameter for use in nonparametric kernel bandwidth selection (if “covbw = newey-west”).
covbwint	Use integer portion of bandwidth.
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.
arma = *arg*	ARMA estimation method: “ml” (maximum likelihood); “gls” (generalized least squares), “cls” (conditional least squares). Not applicable to ARFIMA models which always estimate using maximum likelihood.

armastart = *arg*	ARMA coefficient starting values: "auto" (automatic) "fixed" (legacy EViews fixed); "random" (random draw); "user" (user-specified). Applicable when "arma = ml" or "arma = gls".
s	Use the current coefficient values in estimator coefficient vector as starting values for equations specified by list with AR or MA terms when "arma = cls" (see also `param` (p. 564) of the *Command and Programming Reference*).
s = *number*	Determine starting values for equations specified by list with AR or MA terms when "arma = cls". 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". Does not apply to coefficients for AR and MA terms which are set to EViews determined default values.
numericderiv / -numericderiv	[Do / do not] use numeric derivatives only. If omitted, EViews will follow the global default.
fastderiv / -fastderiv	[Do / do not] use fast derivative computation. If omitted, EViews will follow the global default. Available only for legacy estimation ("optmeth = legacy").
cov = *arg*	Covariance method: "ordinary" (default method based on inverse of the estimated information matrix), "huber" or "white" (Huber-White sandwich method available for nonlinear least squares or ARMA estimated by CLS), "hac" (Newey-West HAC, available for nonlinear least squares or ARMA estimated by CLS)., "hc" (extended heteroskedasticity consistent), "hcuser" (user-specified heteroskedasticity), "cr" (cluster robust). The extended "hc" methods are only available for linear specifications.
hctype = *arg (default "hc2")*	Extended heteroskedasticity consistent method: "hc0" (no d.f. adjustment), "hc1" (d.f. adjusted), "hc2", "hc3", "hc4", "hc4m", "hc5", when "cov = hc".
userwt = *arg*	Name of series containing user-diagonal weights (if "cov = hcuser")
crtype = *arg (default "cr1")*	Cluster robust weighting method: "cr0" (no finite sample correction), "cr1" (finite sample correction), "hc2", "hc3", "hc4", "hc4m", "hc5", when "cov = cr".
crname = *arg*	Cluster robust series name, when "cov = cr".

k = *arg* *(default = 0.7)*	Parameter for "cov = hc, hctype = hc5" or "cov = cr, crtype = cr5".
k1 = *arg* *(default = 1.0)*	Parameter for "cov = hc, hctype = hc4m" or "cov = cr, crtype = cr4m".
k2 = *arg* *(default = 1.5)*	Parameter for "cov = hc, hctype = hc4m" or "cov = cr, crtype = cr4m".
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
prompt	Force the dialog to appear from within a program.
p	Print estimation results.

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

Non-Panel Indicator Saturation Options

For use if "indicator" option is specified.

noiis	Do not search for impulse terms.
sis	Search for step-shift terms.
trend	Search for trend terms.
pval = *number* *(default = 0.05)*	Set the terminal condition p-value used to determine the stopping point of each search path
nolm	Do not perform AR LM diagnostic test.
arpval = *number* *(default = 0.025)*	Set p-value used in AR LM diagnostic test.
arlags = *int (default = 1)*	Set number of lags used in AR LM diagnostic test.
noarch	Do not perform ARCH LM diagnostic test.
archpval = *number* *(default = 0.025)*	Set p-value used in ARCH LM diagnostic test.
archlags = *int* *(default = 1)*	Set number of lags used in ARCH LM diagnostic test.
nojb	Do not perform Jarque-Bera normality diagnostic test.
jbpval = *number* *(default = 0.025)*	Set p-value used in Jarque-Bera normality diagnostic test.

nopet	Do not perform Parsimonious Encompassing diagnostic test.
petpval = *number (default = 0.025)*	Set *p*-value used in Parsimonious Encompassing diagnostic test.
nogum	Do not include the general model as a candidate for model selection.
noempty	Do not include the empty model as a candidate for model selection.
ic = *arg*	Set the information criterion used in model selection: "AIC" (Akaike information criteria, default), "BIC" (Schwarz information criteria), "HQ" (Hannan-Quin criteria).
blocks = *int*	Override the EViews' determination of the number of blocks in which to split the estimation sample.

Panel LS Options

cx = *arg*	Cross-section effects: (default) none, fixed effects ("cx = f"), random effects ("cx = r").
per = *arg*	Period effects: (default) none, fixed effects ("per = f"), random effects ("per = r").
wgt = *arg*	GLS weighting: (default) none, cross-section system weights ("wgt = cxsur"), period system weights ("wgt = persur"), cross-section diagonal weighs ("wgt = cxdiag"), period diagonal weights ("wgt = perdiag").
cov = *arg*	Coefficient covariance method: (default) ordinary, White cross-section system (period clustering) robust ("cov = cxwhite" or "cov = percluster"), White period system (cross-section clustering) robust ("cov = perwhite" or "cov = cxcluster"), White heteroskedasticity robust ("cov = stackedwhite"), White two-way cluster robust (cov = bothcluster"), 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").
keepwgts	Keep full set of GLS weights used in estimation with object, if applicable (by default, only small memory weights are saved).
rancalc = *arg (default = "sa")*	Random component method: Swamy-Arora ("rancalc = sa"), Wansbeek-Kapteyn ("rancalc = wk"), Wallace-Hussain ("rancalc = wh").

nodf	Do not perform degree of freedom corrections in computing coefficient covariance matrix. The default is to use degree of freedom corrections.
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
iter = *arg (default* = "onec")	Iteration control for GLS specifications: perform one weight iteration, then iterate coefficients to convergence ("iter = onec"), iterate weights and coefficients simultaneously to convergence ("iter = sim"), iterate weights and coefficients sequentially to convergence ("iter = seq"), perform one weight iteration, then one coefficient step ("iter = oneb"). Note that random effects models currently do not permit weight iteration to convergence.
unbalsur	Compute SUR factorization in unbalanced data using the subset of available observations for a cluster.
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 estimator coefficient vector as starting values for equations specified by list with AR terms (see also `param` (p. 564) of the *Command and Programming Reference*).
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 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". Does not apply to coefficients for AR terms which are instead set to EViews determined default values.
numericderiv / -numericderiv	[Do / do not] use numeric derivatives only. If omitted, EViews will follow the global default.
fastderiv / -fastderiv	[Do / do not] use fast derivative computation. If omitted, EViews will follow the global default.

showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
prompt	Force the dialog to appear from within a program.
p	Print basic estimation results.

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(cov=white) q = beta(1)+beta(2)*(l^c(1) + k^(1-
    c(1)))
```

estimates the nonlinear regression starting from the specified initial values. The "*cov=white*" 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.

```
equation eq5.ls(cx=f, per=f) n w k ys c
```

estimates the equation EQ5 in the panel workfile using both cross-section and period fixed effects.

```
equation eq6.ls(cx=f, wgt=cxdiag) n w k ys c
```

estimates the equation EQ6 in a panel workfile with cross-section weights and fixed effects.

Cross-references

Chapter 20. "Basic Regression Analysis," on page 5 and Chapter 21. "Additional Regression Tools," on page 23 of the *User's Guide II* discuss the various regression methods in greater depth.

Chapter 13. "Special Expression Reference," on page 323 of the *Command and Programming Reference* describes special terms that may be used in `ls` specifications.

See Chapter 55. "Panel Estimation," on page 1321 of the *User's Guide II* for a discussion of panel equation estimation.

lvageplot	Equation Views

Leverage plots.

Displays leverage plots to discover influential observations, or outliers.

Syntax

equation_name.**lvageplot***(options) variables @ name_suffix*

where *name_suffix* is an optional naming suffix for storing the statistics into series in the workfile.

Options

raw	Do not use partial residuals.
nofit	Do not include a line of fit on the graphs
prompt	Force the dialog to appear from within a program.

Examples

```
eq1.lvageplot x1 x2 @ lplot_
```

will display two graphs, one for the leverage plot of X1 and one for the leverage plot of X2, and will create two new series in the workfile, LPLOT_X1 and LPLOT_X2.

Cross-references

See also "Leverage Plots" on page 258 of the *User's Guide II*.

makecoint	Equation Procs

Create a series containing the estimated cointegrating relationship from an ARDL estimated equation.

This view is only available for non-panel equations estimated using the ARDL method.

Syntax

equation_name.**makecoint [series_name]**

Examples

```
wfopen http://www.stern.nyu.edu/~wgreene/Text/Edition7/TableF5-
    2.txt
```

```
equation eq02.ardl(deplags=3, reglags=3, fixed) log(realcons)
    log(realgdp) @ @expand(@quarter, @droplast)
show eq02.makecoint cointser
```

This example uses data from Greene (2008, page 685), containing quarterly US macroeconomic variables between 1950 and 2000. The first line of this example downloads the data set, the second line creates an equation object and estimates an ARDL model with the log of real consumption as the dependent variable. Three lags of the dependent variable, and three lags of the log of real GDP are used as dynamic regressors. Quarterly dummy variables are included as static regressors.

The final line creates a new series, COINTSER, containing the estimated cointegrating relationship.

Cross-references

See Chapter 29. "ARDL and Quantile ARDL," on page 341 of *User's Guide II* for further discussion.

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 = *arg*	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 51. "State Space Models and the Kalman Filter," on page 1151 of the *User's Guide II* for details on state space estimation.

See also `Equation::derivs` (p. 117), `Equation::grads` (p. 165), `Equation::makegrads` (p. 194).

makefunobj	Equation Procs

Export functional objects to matrices in the workfile.

For equations estimated using the functional coefficients method.

Syntax

eq_name.**makefunobj***(options) matrix_name*

where *matrix_name* is the name of the output matrix.

Options

Basic Options

type = *arg* (*default* = "coef")	Type of result to save: coefficients ("coef"), residuals ("res"), bias ("bias"), covariances involving a single coefficient ("cov"), correlations involving a single coefficient ("cor"), confidence intervals ("ci"). If saving covariances or correlations, you may identify the coefficient of interest using the "coefid = " option.
wf	Make output of workfile length. Note: this option is only relevant if estimation was evaluated over the functional coefficient variable values.
derivs	Include derivative coefficients as part of the output (for all but "type = resid"). Use derivative coefficients are part of the calculation (when type = resid").
nodups	Duplicate observations for in the set of functional coefficient evaluation points are removed.
sort	Rows of the output matrix are sorted in increasing order of the functional coefficient evaluation points.

Pilot Bandwidth Options

If a local pilot bandwidth has exists, all of the pilot bandwidth computation options below will be ignored unless the "nolocalbw" option is specified.

plth = *arg* *(default = "cv")*	Pilot bandwidth method: simple rule-of-thumb ("rot"), robust rule-of-thumb ("rotr"), residual squares criterion ("rsc"), modified multi cross-validation ("cv"), user-defined ("user").
pltbw = *arg* *(default = 1)*	User-defined bandwidth (if "plth = user").
plthmin = *arg* *(default = 0.1)*	Bandwidth grid search minimum value (if not "plth = user").
plthmax = *arg* *(default = 1)*	Bandwidth grid search maximum value (if not "plth = user").
plthlen = *integer* *(default = 100)*	Bandwidth grid search length (if not "plth = user").
plthinc = *integer* *(default = 10)*	Bandwidth grid search increment step percentage increase (if not "plth = user").
plthcup = *integer* *(default = 10)*	Stop rule: consecutive increases of objective function before stop (not available when "plth = user").
pltm = *arg* *(default = 10)*	Modified multifold CV m-value: percentage of sample size used in bandwidth determination (when "plth = cv").
pltq = *integer* *(default = 4)*	Modified multifold CV Q-value: percentage of sample size used in bandwidth determination (when "plth = cv").
auxk = *integer* *(default = 2)*	Estimation polynomial degree for pilot stage in excess of final stage degree. This number should always be an even positive integer.

Covariance Options

coefid = *integer* (*default* = 1)	Coefficient ID for which covariances or correlations are produced when "type = cov" or "type = cov".

Confidence Interval Options

The following options are only available when "type = ci".

seband	Produce standard error bands instead of confidence intervals.
sewidth = *integer* *(default = 1)*	Number of standard errors to use as half-width of confidence band (when "seband" is specified).
cilevel = *arg* *(default = 0.95)*	Confidence interval coverage as a number between 0 and 1 (not applicable when "seband" is specified).
nobias	Ignore bias in confidence interval determination.

Examples

```
eq.makefunobj funcoef
```

produces a matrix of functional coefficients (one column per coefficient) called "FUNCOEF" using the existing local pilot bandwidth if present, or estimating the pilot bandwidth, if not. If estimated, the local pilot bandwidth is updated with the result.

```
eq.makefunobj(type=cor, coefid=3) myfuncorr
```

produces a matrix of functional correlations with respect to the third functional coefficient. The matrix is stored in the workfile under the name "MYFUNCORR".

```
eq.makefunobj(nolocalbw, type=ci, cilevel=0.9) myfunci
```

produces a matrix with the 90% functional confidence intervals. The lower and upper bounds of the intervals are paired in adjacent columns.

Cross-references

See Chapter 38. "Functional Coefficient Regression," on page 645 of the *User's Guide II* for discussion of functional coefficients estimation. See "Post-Estimation Views and Procs" on page 654 of the *User's Guide II* for detail on the various functional calculations.

See also `Equation::funbias` (p. 141), `Equation::funci` (p. 144), `Equation::funcoef` (p. 146), `Equation::funcov` (p. 148), `Equation::funtest` (p. 150), and `Equation::setpilotbw` (p. 237).

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.

Options

prompt	Force the dialog to appear from within a program.

Examples

```
equation eq1.arch sp c
eq1.makegarch cvar
plot cvar^.5
```

estimates a GARCH(1,1) model, saves the conditional variance as a series named CVAR, and plots the conditional standard deviation. If you merely wish to view a plot of the conditional standard deviation without saving the series, use the `Equation::garch` (p. 153) view.

The commands

```
equation eq1.arch(cgarch) sp c
eq1.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 27. "ARCH and GARCH Estimation," on page 273 of the *User's Guide II* for a discussion of GARCH models.

See also `Equation::arch` (p. 64), `Equation::archtest` (p. 70), and `Equation::garch` (p. 153).

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

n = *arg*	Name of group object to contain the series.

Examples

```
eq1.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 "Gradients" on page 1547 of the *User's Guide II* for discussion.

See also `Equation::derivs` (p. 117), `Equation::makederivs` (p. 190), `Equation::grads` (p. 165).

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. 210).

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 444 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 52. "Models," on page 1177 of the *User's Guide II* for a discussion of specifying and solving models in EViews. See also `solve` (p. 594).

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. 476).

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 *(default)*	Ordinary residuals.
s	Standardized residuals (available only after weighted estimation and GARCH, binary, ordered, censored, and count models).
g *(default* for ordered models*)*	Generalized residuals (available only for binary, ordered, censored, and count models).
prompt	Force the dialog to appear from within a program.

Examples

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

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

Cross-references

See "Weighted Least Squares" on page 47 of the *User's Guide II* for a discussion of standardized residuals after weighted least squares and Chapter 31. "Discrete and Limited Dependent Variable Models," on page 425 of the *User's Guide II* for a discussion of standardized and generalized residuals in binary, ordered, censored, and count models.

makergmprobs	Equation Procs

Save the regime probabilities for switching regression equation into series in the workfile.

Syntax

equation_name.**makergmprobs***(options) series_names*

where *equation_name* is the name of an equation estimated using switching regression. The series to be saved should be listed following the command name and options, with one name per regime for one up to the number of estimated regimes.

Options

type = *arg* *(default* = "pred"*)*	Type of regime probability to compute: one-step ahead predicted ("pred"), filtered ("filt"), smoothed ("smooth").
n = *arg*	(optional) Name of group to contain the saved regime probabilities.
prompt	Force the dialog to appear from within a program.

Examples

```
equation eq1.switchreg(type=markov) y c @nv ar(1) ar(2) ar(3)
eq1.makergmprobs r1 r2
```

saves the one-step ahead regime probabilities for the Markov switching regression estimated in EQ1 in series R1 and R2 in the workfile

```
eq1.makergmprobs(type=filt) f1
```

saves the filtered probabilities for regime 1 in F1.

```
eq1.makergmprobs(type=smooth, n=smoothed) s1 s2
```

saved the smoothed probabilities for both regimes in the series S1 and S2, and creates the group SMOOTHED containing S1 and S2.

Cross-references

See "Switching Regression" on page 671 of the *User's Guide II* for discussion.

See also `Equation::rgmprobs` (p. 232).

makestrwgts	Equation Procs

Save a series containing the smooth transition weights for each observation in the estimation sample in a smooth threshold regression.

Syntax

eq_name.**makestrwgts**(*options*) *name*

The command makes weight series in the workfile using *name* or the next available name using *name* as a basename.

Options

prompt	Force the dialog to appear from within a program.

Examples

```
eq1.makestrwgts w
```

saves the weights in the series W.

Cross-references

See Chapter 36. "Smooth Transition Regression," on page 569 of *User's Guide II* for discussion.

See also `Equation::strconstant` (p. 240).

maketransprobs	Equation Procs

Save the regime transition probabilities and expected durations for a switching regression equation into the workfile.

Syntax

equation_name.**maketransprobs***(options) [base_name]*

equation_name.**maketransprobs**(**out = mat**, *options) [matrix_name]*

where *equation_name* is the name of an equation estimated using switching regression.

- In the first form of the command, *base_name* will be used to generate series names for the series that will hold the transition probabilities or durations. The series names for regime transition probabilities will be of the form *base_name*##, where ## are the indices representing elements of the transition matrix (i, j). The series names for expected durations will be of the form *base_name*# where # corresponds to the regime index. Thus, in a two-regime model, the base name "TEMP" corresponds to the tran-

sition probability series TEMP11, TEMP12, TEMP21, TEMP22, and the expected duration series TEMP1, TEMP2.

If *base_name* is not provided, EViews will use the default of "TPROB"

- When the option "output = mat" is provided, the *matrix_name* is the name of the output matrix that will hold the transition probabilities or durations.

  If *matrix_name* are not provided, EViews will default to "TPROB" or the next available name of the form "TPROB##".

  EViews will evaluate the transition probabilities or durations at the date specified by the "obs = " option. If no observation is specified, EViews will use the first date of the estimation sample to evaluate the transition probabilities. Note that if the transition probabilities are time-invariant, setting the observation will have no effect on the contents of the saved results.

Options

type = *arg* *(default* = "trans"*)*	Transition probability results to save: transition probabilities ("trans"), expected durations ("expect").
out = *arg* *(default* = "series"*)*	Output format: series ("series") or matrix ("mat"). If saved as a matrix, only a single transition matrix will be saved using the date specified by "obs = ".
obs = *arg*	Date/observation used to evaluate the transition probabilities if saving results as a matrix ("out = mat"). If no observation is specified, EViews will use the first date of the estimation sample to evaluate the transition probabilities. Note that if the transition probabilities are time-invariant, setting the observation will have no effect on the content of the saved results.
n = *arg*	(optional) Name of group to contain the saved transition probabilities.
prompt	Force the dialog to appear from within a program.

Examples

```
equation eq1.switchreg(type=markov) y c @nv ar(1) ar(2) ar(3)
eq1.maketransprobs(n=transgrp) trans
```

saves the transition probabilities in the workfile in the series TRANS11, TRANS12, TRANS21, TRANS22 and creates the group TRANSGRP containing the series.

The command

```
eq1.maketransprobs(type=expect) AA
```

saves the expected durations in the series AA1 and AA2.

```
eq1.maketransprobs(out=mat) BB
```

saves the transition probabilities in the matrix BB.

Cross-references

See "Switching Regression" on page 671 of *User's Guide II* for discussion.

See also `Equation::transprobs` (p. 252).

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 for binary and censored/truncated models

Syntax

eq_name.**means***(options)*

Options

p	Print the descriptive statistics table.

Examples

```
equation eq1.binary(d=l) 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 31. "Discrete and Limited Dependent Variable Models," on page 425 of *User's Guide II* for a discussion of binary and censored/truncated dependent variable models.

midas	Equation Methods

Estimates an equation using Mixed Data Sampling (MIDAS) regression.

MIDAS regression is an estimation technique which allows for data sampled at different frequencies to be used in the same regression.

Syntax

eq_name.**midas***(options) y x1 [x2 x3 ...] @ z1page\z1 [z2page\z2 ...]*

where *y*, *x1*, etc., are the dependent and explanatory variables in the current page frequency, and *z1page\z1* and *z2page\z2* are the high frequency variable *page\series* specification.

You may not include ARMA terms in a MIDAS regression.

Options

General options

midwgt = *arg*	MIDAS weight method: step function("step"), normalized exponential Almon ("expalmon"), normalized beta function ("beta"), U-MIDAS ("umidas"), Auto-search/GETS ("autogets") or the default Almon/PDL weighting ("almon").
lag = *arg*	Method for specifying the number of lags of the high frequency regressor to use: lag selection ("auto"), fixed ("fixed"). The default is "lag = fixed".
maxlag = *arg*	Maximum number of lags of the high frequency regressor to use when using lag selection. For use when "lag = auto". The default value is 4.
fixedlag = *arg*	Fixed number of lags of the high frequency regressor to use. For use when "lags = fixed". The default value is 4.
steps = *integer*	Stepsize (number of high frequency periods to group). For use when "midwgt = step".
polynomial = *integer*	Polynomial degree. For use when Almon/PDL weighting is employed.
beta = *arg*	Beta function restriction: none ("none"), trend coefficient equals 1 ("trend"), endpoints coefficient equals 0 ("endpoint"), both trend and endpoints restriction ("both"). For use when "midwgt = beta". The default is "beta = none".
optmethod = *arg*	Optimization method for nonlinear estimation: "bfgs" (BFGS); "newton" (Newton-Raphson), "opg" or "bhhh" (OPG or BHHH), "hybrid" (initial BHHH followed by BFGS). Hybrid is the default method.
optstep = *arg*	Step method for nonlinear estimation: "marquardt" (Marquardt); "dogleg" (Dogleg); "linesearch" (Line search). Marquardt is the default method.
cov = *arg*	Covariance method for nonlinear models: "ordinary" (default method based on inverse of the estimated information matrix), "huber" or "white" (Huber-White sandwich).

covinfo = *arg*	Information matrix method for nonlinear models: "opg" (OPG); "hessian" (observed Hessian).
freq = *key*	Set the frequency conversion method. Key can be "first" (the higher frequency data are used from the first observation in the lower frequency period), "last" (default, the higher frequency data are used from the last observation in the lower frequency), or "match" (a specific date matching series from each page is used).
freqsrc = *arg*	Set the source date matching series. Only applies if freq = match is used.
freqdest = *arg*	Set the destination date matching series. Only applies if freq = match is used.
nodf	Do not perform degree of freedom corrections in computing coefficient covariance matrix. The default is to use degree of freedom corrections.
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 estimator coefficient vector as starting values in nonlinear estimation (see also `param` (p. 564) of the *Command and Programming Reference*).
s = *number*	Determine starting values for nonlinear estimation.. Specify a number between zero and one representing the fraction of preliminary EViews chosen values. Note that out of range values are set to "s = 1". Specifying "s = 0" initializes coefficients to zero. By default EViews uses "s = 1".
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
prompt	Force the dialog to appear from within a program.
p	Print estimation results.

Auto-search/GETS options

pval = *number (default = 0.05)*	Set the terminal condition p-value used to determine the stopping point of each search path
nolm	Do not perform AR LM diagnostic test.
arpval = *number (default = 0.025)*	Set p-value used in AR LM diagnostic test.
arlags = *int (default = 1)*	Set number of lags used in AR LM diagnostic test.
noarch	Do not perform ARCH LM diagnostic test.
archpval = *number (default = 0.025)*	Set p-value used in ARCH LM diagnostic test.
archlags = *int (default = 1)*	Set number of lags used in ARCH LM diagnostic test.
nojb	Do not perform Jarque-Bera normality diagnostic test.
jbpval = *number (default = 0.025)*	Set p-value used in Jarque-Bera normality diagnostic test.
nopet	Do not perform Parsimonious Encompassing diagnostic test.
petpval = *number (default = 0.025)*	Set p-value used in Parsimonious Encompassing diagnostic test.
nogum	Do not include the general model as a candidate for model selection.
noempty	Do not include the empty model as a candidate for model selection.
ic = *arg*	Set the information criterion used in model selection: "AIC" (Akaike information criteria, default), "BIC" (Schwarz information criteria), "HQ" (Hannan-Quin criteria).
blocks = *int*	Override the EViews' determination of the number of blocks in which to split the estimation sample.

Examples

```
equation eq1.midas(fixedlag=9, midwgt=beta, beta=endpoint) realgdp
    c realgdp(-1) @ monthlypage\emp(-5)
```

estimates a MIDAS beta weight specification using the low frequency dependent variable REALGDP and regressors C and REALGDP(-1), and 9 beta weighted lags of EMP(-5) from the "monthlypage" workfile page. The beta weight function places zero restrictions on the endpoint coefficient.

```
equation eq2.midas(maxlag=12, lag=auto) realgdp c realgdp(-1) @
    monthlypage\emp(-5)
```

estimates the same equation using PDL/Almon weights. The number of lags is chosen automatically with a maximum of 12 lags.

Cross-references

Chapter 30. "Midas Regression," on page 405 of *User's Guide II* discusses the specification and estimation of MIDAS regression models in EViews.

modselgraph	Equation Views

Display a graph of the selection criteria for the top 20 models as determined by model selection during estimation.

This view is currently only available for equations estimated with elastic net, ridge regression, Lasso, and variable selection using Lasso.

The graph elements will be ordered by the value of the selection criterion.

Syntax

equation_name.**modselgraph**

Examples

Consider the estimated elastic net equation

```
equation my_eq.enet(xtrans=none, lambdaratio=.0001,
    cvseed=513255899) lpsa c lcavol_s lweight_s age_s lbph_s svi_s
    lcp_s gleason_s pgg45_s
```

Then the command

```
my_eq.modselgraph
```

shows a plot of the top 20 model selection objective values.

Cross-references

For further discussion, see Chapter 37. "Elastic Net and Lasso" in *User's Guide II*.

See also `Equation::lambdapath` (p. 178) and `Equation::modseltable` (p. 206).

The data underlying this graph are available via the data member `@modselresults`.

modseltable	Equation Views

Display the selection objective and fit measures associated with the estimation and model selection procedures.

This view is only available for equations estimated with elastic net, ridge regression, Lasso, and variable selection using Lasso.

Displays a table showing the model selection objective, number of non-zero coefficients, and the fit statistics (sum-of-squared residuals, mean-square error, R-squared, and adjusted R-squared) associated with the estimated model.

The row displaying the model selected optimal lambda will be highlighted.

Syntax

eq_name.**modseltable**(*options*)

Options

p	Print output.

Examples

Consider the estimated elastic net equation

```
equation my_eq.enet(xtrans=none, lambdaratio=.0001,
    cvseed=513255899) lpsa c lcavol_s lweight_s age_s lbph_s svi_s
    lcp_s gleason_s pgg45_s
```

Then the command

```
my_eq.modseltable
```

displays the table showing the model selection objective and various fit statistics associated with each lambda in the path.

Cross-references

For further discussion, see Chapter 37. "Elastic Net and Lasso" of *User's Guide II*.

See also `Equation::lambdacoefs` (p. 175) and `Equation::coefpath` (p. 95).

The data underlying this table are available via the data member `@modselresults`.

multibreak	Equation Views

Multiple breakpoint testing.

The `multibreak` view of an equation displays the results of multiple breakpoint testing using sequential and global optimization methods.

This view is only available for (non-panel) equations specified by list without ARMA terms and estimated by ordinary least squares.

Syntax

equation_name.**multibreak***(options) [list_of_breaking_regressors]*

where *equation_name* is the name on an equation specified by list and estimated using least squares. The `multibreak` may be followed by options, and an optional list of breaking regressor names. If the latter list is omitted, the coefficients for all of the regressors in the original equation will be allowed to vary across regimes.

Options

method = *arg* (*default* = "seqplus1")	Breakpoint testing method: "seqplus1" (sequential tests of single $l+1$ versus l breaks), "seqall" (sequential test of all possible $l+1$ versus l breaks), "glob" (tests of global l vs. no breaks), "globplus1" (tests of $l+1$ versus l globally determined breaks), "globinfo" (information criteria evaluation).
trim = *arg* (*default* = 5)	Trimming percentage for determining minimum segment size (5, 10, 15, 20, 25).
maxbreaks = *integer* (*default* = 5)	Maximum number of breakpoints to allow (not applicable if "method = seqall").
maxlevels = *integer* (*default* = 5)	Maximum number of break levels to consider in sequential testing (applicable when "method = sequall").
size = *arg* (*default* = 5)	Test sizes for use in sequential determination and final test evaluation (10, 5, 2.5, 1) corresponding to 0.10, 0.05, 0.025, 0.01, respectively
heterr	Assume regimes specific error distributions in variance computation.
commondata	Assume a common distribution for the data across segments (only applicable if original equation is estimated with a robust covariance method, "heterr" is not specified).

prompt	Force the dialog to appear from within a program.
p	Print the view.

Examples

```
equation eq01.ls m1 c tb3 gdp
eq01.multibreak(maxbreaks=3)
eq01.multibreak(method=glob, size=10, trim=15) tb3
```

The first test line tests for up to 3 structural breaks in all of the coefficients using sequential tests of single $l+1$ versus l breaks. The second line tests uses the global l breaks versus none tests with trimming value 0.15, and a size of 0.10 to test for differences in the coefficient on TB3 across regimes.

The multiple breakpoint tests will use the covariance matrix settings from the original equation when constructing tests. The command

```
equation eq01.ls(cov=hac, covkern=quadspec, covlag=1,
   covbw=andrews) rates c
eq01.multibreak(heterr)
eq01.multibreak(method=glob, heterr)
eq01.multibreak(method=globinfo)
```

estimate an equation using HAC covariances. The second line tests for up to 5 structural breaks using sequential tests of single $l+1$ versus l breaks. The third line uses the global l breaks versus none tests. Both of these tests allow for error distributions to vary across the different segments. The final line evaluates the breakpoints using information criteria associated with the global optimizers.

Cross-references

See "Multiple Breakpoint Tests" on page 238 of *User's Guide II* for discussion. See also Chapter 34. "Least Squares with Breakpoints," beginning on page 535 of *User's Guide II* for tools which estimate equations with structural breaks.

See `Equation::breakls` (p. 82) for estimation of regression equations with breaks.

newsimpact	Equation Views

Display a news-impact graph of equations estimated using GARCH.

Syntax

eq_name.**newsimpact**(*options*)

Options

p	Print graph.

Examples

```
equation eq1.arch(2, 1) y c
```

estimates a GARCH(2, 1) model.

```
eq1.newsimpact
```

displays the news-impact graph of the estimated GARCH model.

Cross-references

See Chapter 27. "ARCH and GARCH Estimation," on page 273 of *User's Guide II* for a discussion of ARCH models. See also "Views of ARCH Models" on page 283 of *User's Guide II*.

nyblom — Equation Views

Perform the Nyblom test of parameter stability or structural change in equations estimated using GARCH.

Syntax

eq_name.**nyblom***(options)*

Options

p	Print results.

Examples

```
equation eq1.arch(2, 1) y c
```

estimates a GARCH(2, 1) model.

```
eq1.nyblom
```

displays the results of a Nyblom stability test.

Cross-references

See Chapter 27. "ARCH and GARCH Estimation," on page 273 of *User's Guide II* for a discussion of ARCH models. See also "Views of ARCH Models" on page 283 of *User's Guide II*.

olepush	Equation Procs

Push updates to OLE linked objects in open applications.

Syntax

equation_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

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

d = *arg* *(default* = "n"*)*	Specify likelihood: normal likelihood function, ordered probit ("n"), logistic likelihood function, ordered logit ("l"), Type I extreme value likelihood function, ordered Gompit ("x").
optmethod = *arg*	Optimization method: "bfgs" (BFGS); "newton" (Newton-Raphson), "opg" or "bhhh" (OPG or BHHH), "legacy" (EViews legacy). Newton-Raphson is the default method.
optstep = *arg*	Step method: "marquardt" (Marquardt); "dogleg" (Dogleg); "linesearch" (Line search). Marquardt is the default method.
cov = *arg*	Covariance method: "ordinary" (default method based on inverse of the estimated information matrix), "huber" or "white" (Huber-White sandwich method)., "glm" (GLM method), "cr" (cluster robust).

covinfo = *arg*	Information matrix method: "opg" (OPG); "hessian" (observed Hessian). (Applicable when non-legacy "optmethod = ".)
df	Degree-of-freedom correct the coefficient covariance estimate.(For non-cluster robust methods estimated using non-legacy estimation).
h	Huber-White quasi-maximum likelihood (QML) standard errors and covariances. (Legacy option Applicable when "optmethod = legacy").
crtype = *arg* *(default "cr1")*	Cluster robust weighting method: "cr0" (no finite sample correction), "cr1" (finite sample correction), when "cov = cr".
crname = *arg*	Cluster robust series name, when "cov = cr".
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 (see also `param` (p. 564) of the *Command and Programming Reference*).
s = *number*	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.
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
prompt	Force the dialog to appear from within a program.
p	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=l,cov=huber) 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 444 of the *User's Guide II* for additional discussion.

See `Equation::binary` (p. 79) for the estimation of binary dependent variable models. See also `Equation::makelimits` (p. 195).

orthogtest	Equation Views

Performs the Instrument Orthogonality Test

The Orthogtest view of an equation carries out the Instrument Orthogonality / C-test Test for equations estimated via TSLS or GMM.

Syntax

eq_name.**orthogtest***(options) instruments*

Options

prompt	Force the dialog to appear from within a program.
p	Print results.

Instruments

A list of instruments to be tested for orthogonality. Note the instruments must have been included in the original equation.

Examples

```
equation eq1.gmm y c x1 x2 @ z1 z2 z3 z4
e1.orthogtest z1 z4
```

estimates an equation, called EQ1, and estimates it via GMM with four instruments Z1, Z2, Z3, Z4, and then performs the Orthogonality Test where Z1 and Z4 are tested for orthogonality.

Cross-references

See "Instrument Orthogonality Test" on page 115 of *User's Guide II* for discussion.

outliers	Equation Views

Display the outliers summary view for an equation estimated via least squares with automatic outlier indicator saturation.

Syntax

equation_name.**outliers**

Examples

```
equation eq1.ls(indicator) y c x
show eq1.outliers
```

Estimates a least squares regression with Y as the dependent variable, and X as a regressor, with an intercept, and with automatic detection of outliers via indicator saturation, and then displays the summary of the outlier detection routine.

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. 231)).

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. 231).

paths	Equation Views

Display the auto-gets paths view.

Syntax

equation_name.**paths**

Examples

```
eq1.varsel(method=gets) y c @ x1 x2 x3 x4 x5 x6 x7 x8
show eq1.paths
```

performs a variable selection routine using the auto-gets method, and then displays the paths view of that procedure.

Cross-references

See also Chapter 22. "Regression Variable Selection," on page 65 of *User's Guide II* for extensive discussion.

pmghausmantest	Equation Views

Displays a spool object with the results of the Hausman test for similarity against mean-group and dynamic fixed effects estimators in PMG estimation.

Syntax

equation_name.**pmghausmantest**(*options*)

Options

p	Print results.

Examples

```
equation eq.ardl log(cons) log(inf)
eq.pmghausmantest
```

Displays a spool object with the results of the Hausman test for similarity against mean-group and dynamic fixed effects estimators in PMG estimation.

Cross-references

See "Pooled Mean Group ARDL Estimation," on page 1329 and Chapter 29. "ARDL and Quantile ARDL," on page 341 of *User's Guide II* for discussion.

See also `Equation::ardl` (p. 71).

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**(*n, 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 the first option in parentheses after the keyword `predict`.

Options

n (default = .5)	Cutoff probability for success prediction in binary models (between 0 and 1).
prompt	Force the dialog to appear from within a program.
p	Print the prediction table.

Examples

```
equation eq1.binary(d=l) 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 425 of *User's Guide II* for a discussion of binary models, and "Expectation-Prediction (Classification) Table" on page 434 of *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. 79).

qrcrprocess	Equation Views

Displays a spool object producing a quantile process of the cointegrating relation.

Syntax

eq_name.**qrcrprocess**(*options*) *[arg]*

where *arg* is an optional list containing the quantile values (specified using numbers, scalar objects, or vectors) for which you wish to compute estimates.

- If *arg* is not specified, EViews will display results for the original equation along with coefficients for equations estimated at a set of equally spaced number of quantiles as specified by the "n = " option. If "n = " is not specified, the default is to display results for the deciles.
- If *arg* is specified, EViews will display results for the original equation along with coefficients for equations estimated at the specified quantiles.

Options

n = *arg* *(default* = 10*)*	Number of quantiles for process estimates.
quantout = *name*	Save vector containing test quantile values.
coefout = *name*	Save matrix containing coefficient estimates of the cointegrating relation form. 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.
prompt	Force the dialog to appear from within a program.
p	Print results.

Examples

```
ardl_eq.qrcrprocess
```

This generates a quantile process of the cointegrating relation for each quantile running from 0.1 to 0.9.

```
ardl_eq.qrcrprocess(n=4, quantout=_vec_q, coefout=_mat_cointrel)
ardl_eq.qrcrprocess(quantout=_vec_q, coefout=_mat_cointrel) 0.25
    0.5 0.75
```

Both commands generate the cointegrating relation process for each of the quantile values 0.25, 0.5, and 0.75. Additionally, this produces 2 workfile objects, "_vec_q" (a vector),

"_mat_cointrel¨(a matrix), and "_mat_cointrel" (a matrix), corresponding to the three quantile values and the cointegrating relation process.

Cross-references

See also `Equation::qrecprocess` (p. 217) and `Equation::qrprocess` (p. 221).

qrecprocess	Equation Views

Displays a spool object producing a quantile process for each of the conditional error correction and error correction coefficients.

Syntax

eq_name.**qrecprocess**(*options*) *[arg]*

where *arg* is a optional list containing the quantile values (specified using numbers, scalar objects, or vectors) for which you wish to compute estimates.

- If *arg* is not specified, EViews will display results for the original equation along with coefficients for equations estimated at a set of equally spaced number of quantiles as specified by the "n = " option. If "n = " is not specified, the default is to display results for the deciles.
- If *arg* is specified, EViews will display results for the original equation along with coefficients for equations estimated at the specified quantiles.

Options

n = *arg* *(default* = 10*)*	Number of quantiles for process estimates.
quantout = *name*	Save vector containing test quantile values.
coefout = *name*	Saves two matrices differentiated by suffixes "_cec" and "_ec", corresponding to the coefficients associated with the CEC and EC forms, respectively. 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.
covout = *name*	Saves two symmetric matrices, differentiated by suffixes "_cec" and "_ec", corresponding to the covariance matrices of coefficients associated with the CEC and EC forms, respectively.
prompt	Force the dialog to appear from within a program.
p	Print results.

Examples

```
ardl_eq.qrecprocess
```

This generates a quantile process of the CEC and EC coefficients for each quantile running from 0.1 to 0.9.

```
ardl_eq.qrecprocess(n=4, quantout=_vec_q, coefout=_mat_coefs,
    covout=_mat_covs)
ardl_eq.qrecprocess(quantout=_vec_q, coefout=_mat_coefs,
    covout=_mat_covs) 0.25 0.5 0.75
```

Both commands generate the process for CEC and EC coefficients for each of the quantile values 0.25, 0.5, and 0.75. Additionally, this produces 5 workfile objects, "_vec_q" (a vector), "_mat_coefs_cec" (a matrix), "_mat_coefs_ec" (a matrix), "_mat_covs_cec" (a matrix), and "_mat_covs_ec" (a matrix), corresponding to the three quantile values, coefficient process for each of the CEC and EC forms, covariances process for each of the CEC and EC forms, respectively.

Cross-references

See also `Equation::qrcrprocess` (p. 216) and `Equation::qrprocess` (p. 221).

qreg	Equation Methods

Estimate a quantile regression specification.

Syntax

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

eq_name.**qreg***(options) linear_specification*

Options

quant = *number* *(default = 0.5)*	Quantile to be fit (where *number* is a value between 0 and 1).
w = *arg*	Weight series or expression. Note: *we recommend that, absent a good reason, you employ the default settings* **Inverse std. dev. weights** *("wtype = istdev") with* **EViews default** *scaling ("wscale = eviews") for backward compatibility with versions prior to EViews 7.*
wtype = *arg* *(default = "istdev")*	Weight specification type: inverse standard deviation ("istdev"), inverse variance ("ivar"), standard deviation ("stdev"), variance ("var").

wscale = *arg*	Weight scaling: EViews default ("eviews"), average ("avg"), none ("none"). The default setting depends upon the weight type: "eviews" if "wtype = istdev", "avg" for all others.
cov = *arg* (*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 = *arg* (*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 = *number*	Use user-specified bandwidth value in place of automatic method specified in "bwmethod = ".
bwsize = *number* (*default* = 0.05)	Size parameter for use in computation of bandwidth (used when "bw = hs" and "bw = bf").
spmethod = *arg* (*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 = *arg* (*default* = "pair")	Bootstrap method: "resid" (residual bootstrap), "pair" (xy-pair bootstrap), "mcmb" (MCMB bootstrap), "mcmba" (MCMB-A bootstrap).
btreps = *integer* (*default* = 100)	Number of bootstrap repetitions
btseed = *positive integer*	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.
btrnd = *arg* (*default* = "kn" or method previously set using `rndseed` (p. 577) in the *Command and Programming Reference*).	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").

btobs = *integer*	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.
btout = *name*	*(optional)* Matrix to hold results of bootstrap simulations.
k = *arg* *(default* = "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).
m = *integer*	Maximum number of iterations.
s	Use the current coefficient values in estimator coefficient vector as starting values (see also `param` (p. 564) in the *Command and Programming Reference*).
s = *number (default* = 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.
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
prompt	Force the dialog to appear from within a program.
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, btmethod=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 40. "Quantile Regression," on page 707 of *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]* *[***@coefs** *coeflist]*

where *arg* is a optional list containing the quantile values (specified using numbers, scalar objects, or vectors) for which you wish to compute estimates, and optionally the **@coefs** keyword followed by a *coeflist* of the subset of coefficients to display.

- If *arg* is not specified, EViews will display results for the original equation along with coefficients for equations estimated at a set of equally spaced number of quantiles as specified by the "n = " option. If "n = " is not specified, the default is to display results for the deciles.
- If *arg* is specified, EViews will display results for the original equation along with coefficients for equations estimated at the specified quantiles.
- If a *coeflist* is not provided, results for all coefficients will be displayed. For models that contain an intercept, the *coeflist* may consist of the **@incptonly** keyword, indicating that only results for the intercept will be displayed.

You may specify a maximum of 1000 total coefficients (number of display coefficients times the number of quantiles) and a maximum of 500 quantiles.

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

Options

n = *arg* *(default* = 10*)*	Number of quantiles for process estimates.
graph	Display process estimate results as graph.
size = *arg* *(default* = 0.95*)*	Confidence interval size for graph display
quantout = *name*	Save vector containing test quantile values.

coefout = *name*	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.
covout = *name*	Save symmetric matrix containing covariance matrix for the vector set of coefficient estimates.
prompt	Force the dialog to appear from within a program.
p	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 714 of *User's Guide II* for a discussion of the quantile process.

See also `Equation::qrcrprocess` (p. 216) and `Equation::qrecprocess` (p. 217).

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] [@***coefs** *coeflist]*

where *arg* is a optional list containing the quantile values (specified using numbers, scalar objects, or vectors) for which you wish to compute estimates, and optionally the **@coefs** keyword followed by a *coeflist* of the subset of coefficients to display.

- If *arg* is not specified, EViews will perform tests for the existing equation and coefficients for equations estimated at a set of equally spaced quantiles as specified by the "n = " option. If "n = " is not specified, the default is to display results for the quartiles (.25, .75).
- If *arg* is specified, EViews will perform results for the original equation along with tests including coefficients for equations estimated at the specified quantiles.
- If a *coeflist* is not provided, all of the slope coefficients will be employed in the test.

You may specify a maximum of 1000 total coefficients (number of coefficients in the equation specification times the number of quantiles) and a maximum of 500 quantiles in the test.

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

Options

n = *arg* *(default = 4)*	Number of quantiles for process estimates.
quantout = *name*	Save vector containing test quantile values.
coefout = *name*	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.
covout = *name*	Save symmetric matrix containing covariance matrix for the vector set of coefficient estimates.
prompt	Force the dialog to appear from within a program.
p	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 716 of *User's Guide II* for a discussion of the slope equality test.

See also `Equation::qrsymm` (p. 224).

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] [***@coefs** *coeflist]*

where *arg* is a optional list containing the quantile values (specified using numbers, scalar objects, or vectors) for which you wish to compute estimates, and optionally the **@coefs** keyword followed by a *coeflist* of the subset of coefficients to display.

- 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 as specified by the "n = " option. If "n = " is not specified, the default is to display results for the outer quartiles {0.25, 0.75}.

For specifications estimated with $\tau \neq 0.5$, EViews will include the original quantile in the set of quantiles to test. You may specify "n = e" to perform a test using only estimates obtained at the symmetric pair $\{\tau, 1 - \tau\}$.

- If *arg* is specified, EViews will perform the test using only the specified quantiles and their complements. The original equation quantile will not be tested unless it is entered explicitly.
- If a *coeflist* is not provided, results for all coefficients will be displayed. For models that contain an intercept, the *coeflist* may consist of the **@incptonly** keyword, indicating that only results for the intercept will be displayed.

You may specify a maximum of 1000 total coefficients (number of coefficients in the equation specification times the number of quantiles) and a maximum of 500 quantiles in the test.

All estimation will be performed using the settings from the original equation. Note that the original equation must include an intercept for you to perform this test

Options

n = *arg* *(default = 4)*	Number of quantiles for testing.
quantout = *name*	Save vector containing test quantile values.
coefout = *name*	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.
covout = *name*	Save symmetric matrix containing covariance matrix for the vector set of coefficient estimates.
prompt	Force the dialog to appear from within a program.
p	Print output from the test.

Examples

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

```
eq1.qrsymm @coefs @incptonly
```

and we may specify alternative quantiles to test

```
eq1.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 eq1.qreg(quant=.4) log(y) c log(x)
eq1.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(4) v1
v1.fill .1,.2,.3,.4
eq1.qrsymm v1
```

Cross-references

See "Symmetric Quantiles Test" on page 717 of *User's Guide II* for a discussion of the symmetric quantiles test.

See also Equation::qrslope (p. 223).

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. 246), `Equation::testdrop` (p. 247), `Equation::fixedtest` (p. 139), and `Equation::wald` (p. 267).

rcomptest

Equation Views

Tests for the presence of cross-sectional or time random components in a panel equation. estimated using pooled least squares.

Computes the conventional LM (Breusch-Pagan, 1980, uniformly most powerful LM (Honda, 1985), standardized Honda (Moulton and Randolph, 1989; Baltagi, Chang, and Li, 1998), locally mean most powerful (LMMP) (King and Wu, 1997), Standardized King-Wu, and Gourieroux, Holly, and Monfort (1982) test statistics.

Note that the equation must be estimated with pooled least squares for this test to be applied.

Syntax

equation_name.**rcomptest**

Options

p	Print test results

Examples

```
equation eq1.ls @log(gsp) c @log(p_cap) @log(pc) @log(emp) unemp
eq1.rcomptest
```

will estimate a panel model using pooled least squares and will compute and display the panel random effects test results.

Cross-references

See "LM Tests for Random Effects" on page 1358 of *User's Guide II* for discussion.

See also `Equation::fixedtest` (p. 139).

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.

Cross-references

See also `Equation::results` (p. 231).

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

prompt	Force the dialog to appear from within a program.
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 252 of *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

g *(default)*	Display graph of actual/fittted/residuals (with one standard error bands)
n	Display graph of residuals only (with one standard error bands)
t	Display table of actual/fitted/residuals.
s	Display graph of standardized residuals.
p	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::makeresids` (p. 197).

resoutliers	Equation Views

Detect outliers in the residuals or regressors of the equation.

Use Tukey fences, mean/standard deviation fences, wavelet outliers, ARMA outliers or influence statistic detection methods to identify observations that may contain outliers.

Syntax

equation_name.**resoutliers***(options)*

Options

sens = *arg*	Set the sensitivity level. Valid arguments are “low”, “medium” (default), and “high”.
nofence	Do not perform Tukey and mean/standard deviation fences.
nowave	Do not perform Wavelet Outlier detection.
noarma	Do not perform ARMA based outlier detection. ARMA outlier detection is only available for least squares equations containing ARMA terms, and is turned on by default.
noinf	Do not perform influence statistic (not including DFBETAS) based outlier detection. Influence statistic outlier detection is only available for linear least squares equations, and is turned on by default.
dfbeta	Perform DFBETA influence statistic based outlier detection. DFBETA based outlier detection is only available for linear least squares equations, and is turned off by default.
tukeyk = *arg*	Set the value *k* in the Tukey fence detection routine. This will override the value of *k* set by the sens = option.
meanstdevk = *arg*	Set the value *k* in the mean/standard deviation fence detection routine. This will override the value of *k* set by the sens = option.
wavesig = *arg*	Set the value false discovery rate significance value used in the Wavelet Outlier detection routine. This will override the value set by the sens = option.
armac = *arg*	Set the value *c* in the ARMA outlier detection routine. This will override the value of *c* set by the sens = option.
rsbound = *arg*	Set the value *c* in RSTUDENT outlier detection. This will override the value of *c* set by the sens = option.

hbound = *arg*	Set the value *c* in HatMatrix outlier detection. This will override the value of *c* set by the sens = option.
dfsbound = *arg*	Set the value *c* in DFFITS outlier detection. This will override the value of *c* set by the sens = option.
covbound = *arg*	Set the value *c* in CovRatio outlier detection. This will override the value of *c* set by the sens = option.
betabound = *arg*	Set the value *c* in DFBETA outlier detection. This will override the value of *c* set by the sens = option.
series = *name*	Create a new series in the workfile, named *name*, containing a value of 1 for any observations identified as an outlier, and a value of 0 for any observation identified as not an outlier.
datestring = *name*	Create a new string object in the workfile containing the dates (or observation identifiers) for any observations identified as an outlier.
grlabels	Turn on observation labels on the outlier graph.

Examples

```
equation eq01.ls gdpc1 c unemp
eq01.resoutliers(nofence, dfbeta, sens=low)
```

Estimates an equation with GDPC1 as the dependent variable, and a constant and UNEMP as regressors. Then, outlier detection on the residuals is performed, opting to not use either fence detection, but to include the dfbeta influence statistics (along with the other influence statistics included by default), and setting the sensitivity of the detection to "low".

results	Equation Views

Displays the results view of an estimated equation.

Syntax

equation_name.**results***(options)*

Options

p	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.

Cross-references

See also Equation::representations (p. 228).

rgmprobs	Equation Views

Display regime probabilities for a switching regression equation.

Syntax

eq_name.**rgmprobs***(options) [indices]*

where *eq_name* is the name of an equation estimated using switching regression. The elements to display are given by the optional *indices* corresponding to the regimes (*e.g.*, "1 2 3" or "2 3"). If *indices* is not provided, results for all of the regimes will be displayed.

Options

type = *arg* (*default* = "pred")	Type of regime probability to compute: one-step ahead predicted ("pred"), filtered ("filt"), smoothed ("smooth").
view = *arg* (*default* = "graph")	Display format: multiple graphs ("graph"), single graph "graph1", sheet ("sheet"), summary ("summary").
prompt	Force the dialog to appear from within a program.
p	Print results.

Examples

```
equation eq1.switchreg(type=markov) y c @nv ar(1) ar(2) ar(3)
eq1.rgmprobs
```

displays two graphs containing the one-step ahead regime probabilities for the Markov switching regression estimated in EQ1.

```
eq1.rgmprobs(type=filt) 2
```

displays the filtered probabilities for regime 2.

```
eq1.rgmprobs(type=smooth, view=graph1)
```

displays the smoothed probabilities for both regimes in a single graph.

Cross-references

See "Switching Regression" on page 671 of *User's Guide II* for discussion.

See also Equation::makergmprobs (p. 198).

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

r	Plot the recursive residuals about the zero line with plus and minus two standard errors.
r,s	Plot the recursive residuals and save the residual series and their standard errors as series named R_RES and R_RESSE, respectively.
c	Plot the recursive coefficient estimates with two standard error bands.
c,s	Plot the listed recursive coefficients and save all coefficients and their standard errors as series named R_C1, R_C1SE, R_C2, R_C2SE, and so on.
o	Plot the p-values of recursive one-step Chow forecast tests.
n	Plot the p-values of recursive n-step Chow forecast tests.
q	Plot the CUSUM (standardized cumulative recursive residual) and 5 percent critical lines.
v	Plot the CUSUMSQ (CUSUM of squares) statistic and 5 percent critical lines.
prompt	Force the dialog to appear from within a program.
p	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 253 of *User's Guide II*.

See also `Equation::facbreak` (p. 133) and `Equation::breaktest` (p. 86).

robustls	Equation Methods

Estimates an equation using robust least squares.

You may perform three different types of robust estimation: M-estimation, S-estimation and MM-estimation.

Syntax:

eq_name.**robustls***(options) y x1 [x2 x3…]*

Enter the `robustls` keyword, followed by the dependent variable and a list of the regressors.

Options

method = *arg* (*default* = "m")	Robust estimation method: "m" (M-estimation), "s" (S-estimation) or "mm" (MM-estimation).
cov = *arg* (*default* = "type1")	Covariance method type: "type1", "type2", or "type3".
tuning = *number*	Specify a value for the tuning parameter. If a value is not specified, EViews will use the default tuning parameter for the type of estimation and weighting function (if applicable).
c = *s*	Convergence criterion. The criterion will be set to the nearest value between 1e-24 and 0.2.
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.

m = *integer*	Maximum number the number of iterations.
prompt	Force the dialog to appear from within a program.
p	Print results.

M-estimation Options

fn = *arg* (*default* = "bisquare")	Weighting function used during M-estimation: "andrews" (Andrews), "bisquare" (Bisquare), "cauchy" (Cauchy), "fair", "huber", "huberbi" (Huber-bisquare), "logistic" (Logistic), "median", "tal" (Talworth), "Welsch" (Welsch).
scale = *arg* (*default* = "madzero")	Scaling method used for calculating the scalar parameter during M estimation: "madzero" (median absolute deviation, zero centered), "madmed" (median absolute deviation, median centered), "huber" (Huber scaling).
hmat	Use the hat-matrix to down-weight observations with high leverage.

S and MM estimation options

compare = *integer* (*default* = 4)	Number of comparison sets.
refine = *integer* (*default* = 2)	Number of refinements.
trials = *integer* (*default* = 200)	Number of trials.
subsmpl = *integer*	Specifies the size of the subsamples. Note, the default is number of coefficients in the regression.
seed = *number*	Specifies the random number generator seed
rng = *arg*	Specifies the type of random number generator. The key can be; 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").

MM estimation options

mtuning = *arg*	M-estimator tuning parameter. Note the S-estimator tuning parameter is set with the "tuning = " option outlined above.
hmat	Use the hat-matrix to down-weight observations with high leverage during m-estimation.

Examples

The following examples use the "Rousseeuw and Leroy.wf1" file located in the EViews application data directory.

```
equation eq1.robustls salinity c lagsal trend discharge
```

This line estimates a simple M-type robust estimation, with SALINITY as the dependent variable, and a constant, LAGSAL, TREND and DISCHARGE as independent variables.

The line:

```
equation eq2.robustls(method=mm, tuning=2.937, mtuning=3.44,
   cov=type2) salinity c lagsal trend discharge
```

estimates the same model, but using MM-estimation, with an S tuning constant of 2.937, an M tuning constant of 3.44, and using Huber Type II standard errors.

Cross-references

See Chapter 33. "Robust Least Squares," beginning on page 515 of *User's Guide II* for discussion.

setattr — Equation Procs

Set the object attribute.

Syntax

equation_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the "revised" attribute in the object A to the string "never", and extracts the attribute into the string object S.

Cross-references

See "Adding Custom Attributes in the Label View" on page 123 and "Adding Your Own Label Attributes" on page 70 of *User's Guide I*.

setpilotbw	Equation Procs

Set or clear the local pilot bandwidth. Once set, the cached value may be used in all post-estimation routines that require a pilot bandwidth.

For equations estimated using the functional coefficients method.

Syntax

eq_name.**setpilotbw***(options)*

Options

Basic Options

clear	Clear any previously set local pilot bandwidth.s

Pilot Bandwidth Options

Unless clearing the local pilot bandwidth using the option "clear", the following options specify the pilot bandwidth computation.

plth = *arg* *(default = "cv")*	Pilot bandwidth method: simple rule-of-thumb ("rot"), robust rule-of-thumb ("rotr"), residual squares criterion ("rsc"), modified multi cross-validation ("cv"), user-defined ("user").
pltbw = *arg* *(default = 1)*	User-defined bandwidth (if "plth = user").
plthmin = *arg* *(default = 0.1)*	Bandwidth grid search minimum value (if not "plth = user").
plthmax = *arg* *(default = 1)*	Bandwidth grid search maximum value (if not "plth = user").
plthlen = *integer* *(default = 100)*	Bandwidth grid search length (if not "plth = user").
plthinc = *integer* *(default = 10)*	Bandwidth grid search increment step percentage increase (if not "plth = user").
plthcup = *integer* *(default = 10)*	Stop rule: consecutive increases of objective function before stop (not available when "plth = user").

pltm = *arg* *(default = 10)*	Modified multifold CV m-value: percentage of sample size used in bandwidth determination (when "plth = cv").
pltq = *integer* *(default = 4)*	Modified multifold CV Q-value: percentage of sample size used in bandwidth determination (when "plth = cv").
auxk = *integer* *(default = 2)*	Estimation polynomial degree for pilot stage in excess of final stage degree. This number should always be an even positive integer.

Examples

```
eq1.setpilotbw.
```

sets the local pilot bandwith using computation default options.

```
eq1.setpilotbw(plth=rsc)
```

computes a pilot bandwidth using the residual squares criterion and saves the value as the local pilot bandwidth.

```
eq1.setpilotbw(plth=user, pltbw=0.5)
```

sets the local pilot bandwidth to 0.5.

```
eq1.setpilotbw(clear)
```

clears (uninitializes) the local pilot bandwidth.

Cross-references

See Chapter 38. "Functional Coefficient Regression," on page 645 of *User's Guide II* for discussion of functional coefficients estimation.

See "Bandwidth Selection" on page 646 and "Bandwidth Views" on page 658 of *User's Guide II* for a discussion of bandwidths.

signbias	Equation Views

Perform the sign-bias test (Engle and Ng, 1993) of misspecification in equations estimated using GARCH.

Syntax

eq_name.**signbias**(*options*)

Options

p	Print output from the test.

Examples

```
equation eq1.arch(2, 1) y c
```

estimates a GARCH(2, 1) model.

```
eq1.signbias
```

displays the results of a sign-bias misspecification test.

Cross-references

See Chapter 27. "ARCH and GARCH Estimation," on page 273 of *User's Guide II* for a discussion of ARCH models.

See also "Views of ARCH Models" on page 283 of *User's Guide II*.

similarity	Equation Views

Compute the PMG Hausman test for similarity against mean-group and dynamic fixed effects estimators (in panel equations estimated by ARDL/PMG)

Syntax

eq_name.**similarity**(*options*)

Options

p	Print output.

Example

```
pmg_eq.similarity
```

displays a spool object with several tables containing the results of the Hausman test, comparisons of results, and auxiliary estimation results employed in computing the test statistic.

Cross-references

See "Pooled Mean Group ARDL Estimation," on page 1329 of *User's Guide II* for discussion.

See also `Equation::ardl` (p. 71).

srcoefs	Equation Views

Displays a spool object with the results of error-correction regressions for each cross-section in PMG estimation.

Syntax

eq_name.**srcoefs**

Options

p	Print results.

Example

```
equation eq.ardl log(cons) log(inf)
eq.srcoefs
```

Displays a spool object with the results of error-correction regressions for each cross-section in PMG estimation.

Cross-references

See "Pooled Mean Group ARDL Estimation," on page 1329 and Chapter 29. "ARDL and Quantile ARDL," on page 341 of *User's Guide II* for discussion.

See also `Equation::ardl` (p. 71).

strconstant	Equation Views

Compute tests of parameter constancy of the base specification against a smooth transition alternative in a smooth threshold regression.

Syntax

eq_name.**strconstant**(*options*)

Options

p	Print output from the test.

Cross-references

See Chapter 36. "Smooth Transition Regression," on page 569 of *User's Guide II* for discussion.

See also `Equation::strlinear` (p. 241) and `Equation::strnonlin` (p. 241).

strlinear	Equation Views

Compute tests for linearity of the base specification against the smooth threshold alternative in a smooth threshold regression.

Syntax

eq_name.**strlinear**(*options*)

Options

p	Print output from the test.

Cross-references

See Chapter 36. "Smooth Transition Regression," on page 569 and "Linearity Testing" on page 581 of *User's Guide II* for discussion.

See also `Equation::strlinear` (p. 241) and `Equation::strnonlin` (p. 241).

strnonlin	Equation Views

Compute tests for additional nonlinearity against additive or encapsulated alternatives (for equations in a smooth threshold regression).

Syntax

eq_name.**strnonlin**(*options*)

Options

encap	Compute tests for additional nonlinearity against encapsulated alternatives.
p	Print output from the test.

Cross-references

See Chapter 36. "Smooth Transition Regression," on page 569 and "Remaining Nonlinearity Tests" on page 584 of *User's Guide II* for discussion.

See also `Equation::strconstant` (p. 240) and `Equation::strlinear` (p. 241).

strwgts	Equation Views

Compute and display the transition weights in a smooth threshold regression.

The default display shows a graph of the transition function. You may also display the weights for each observation in the estimation sample.

Syntax

eq_name.**strwgts**(*options*)

Options

view = *arg*	Weight display: “graph” (graph of the weight for each individual in the estimation sample), “sheet” (spreadsheet containing weights for each individual), “summary” (summary statistics). The default view displays a graph of the function with optional borders.
ab = *arg*	Additional graph borders to display when showing the default view of the weights: “none” (do not display borders)”, “boxplot” (display boxplot borders), “histogram” (display a histogram). The default view shows a boxplot on each border.
output = *arg*	Optional name of matrix to save the data used in the function plot.
prompt	Force the dialog to appear from within a program.
p	Print output from the test.

Cross-references

See Chapter 36. “Smooth Transition Regression,” on page 569 and “Remaining Nonlinearity Tests” on page 584 of *User’s Guide II* for discussion.

See also `Equation::makestrwgts` (p. 199).

switchreg	Equation Methods

Estimate a switching regression model (simple exogenous or Markov).

Syntax

eq_name.**switchreg**(*options*) *dependent_var list_of_varying_regressors [* **@nv** *list_of_nonvarying_regressors] [* **@prv** *list_of_probability_regressors]*

List the `switchreg` keyword, followed by options, then the dependent variable and a list of the regressors with regime-varying coefficients, following optionally by the keyword **@nv** and a list of regressors with regime-invariant coefficients, and by the keyword **@prv** and a list of regressors that enter into the transition probability specification.

The dependent variable in `switchreg` may not be an expression. Dynamics may be specified by including lags of the dependent variable as regressors, or by specifying AR errors using the `AR` keyword. The latter incorporate mean adjusted lags of the form specified by the "Hamilton-model."

Options

type = *arg*	Type of switching: simple exogenous ("simple"), Markov ("markov").
nstates = *integer* *(default* = 2*)*	Number of regimes.
heterr	Allow for heterogeneous error variances across regimes
fprobmat = *arg*	Name of fixed transition probability matrix allows for fixing specific elements of the time-invariant transition matrix. Leave NAs in elements of the matrix to estimate. The (i, j) element of the matrix corresponds to $P(s_t = j \mid s_{t-1} = i)$.
initprob = *arg* *(default* = "ergodic"*)*	Method for determining initial Markov regime probabilities: ergodic solution ("ergodic"), estimated parameter ("est"), equal probabilities ("uniform"), user-specified probabilities ("user"). If "initprob = user" is specified, you will need to specify the "userinit = " option.
userinit = *arg*	Name of vector containing user-specified initial Markov probabilities. The vector should have rows equal to the number of states; we expand this to the size of the initial lag state vector where necessary for AR specifications. For use in specifications containing both the "type = markov" and "initprob = user" options.
startnum = *arg* *(default* = 0 or 25*)*	Number of random starting values tried. The default is 0 for user-supplied coefficients (option "s") and 25 in all other cases.
startiter = *arg* *(default* = 10*)*	Number of iterations taken after each random start before comparing objective to determine final starting value.
searchnum = *arg* *(default* = 0*)*	Number of post-estimation perturbed starting values tried.

searchsds = *arg* *(default* = 1*)*	Number of standard deviations to use in perturbed starts (if "searchnum = ") is specified.
seed = *positive_integer* from 0 to 2,147,483,647	Seed the random number generator. If not specified, EViews will seed random number generator with a single integer draw from the default global random number generator.
rnd = *arg* *(default* = "kn" or method previously set using `rndseed` (p. 577) in the *Command and Programming Reference)*.	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").

In addition to the specification options, there are options for estimation and covariance calculation.

Additional Options

optmethod = *arg*	Optimization method: "bfgs" (BFGS); "newton" (Newton-Raphson), "opg" or "bhhh" (OPG or BHHH), "legacy" (EViews legacy). BFGS is the default method.
optstep = *arg*	Step method: "marquardt" (Marquardt); "dogleg" (Dogleg); "linesearch" (Line search). Marquardt is the default method.
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.
cov = *arg*	Covariance method: "ordinary" (default method based on inverse of the estimated information matrix), "huber" or "white" (Huber-White sandwich method).
covinfo = *arg*	Information matrix method: "opg" (OPG); "hessian" (observed Hessian). (Applicable when non-legacy "optmethod = ".)
nodf	Do not degree-of-freedom correct the coefficient covariance estimate.

coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
s	Use the current coefficient values in "C" as starting values (see also `param` (p. 564) of the *Command and Programming Reference*).
s = *number*	Specify a number between zero and one to determine starting values as a fraction of EViews default values (out of range values are set to "*s = 1*").
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
prompt	Force the dialog to appear from within a program.
p	Print results.

Examples

```
equation eq_41a.switchreg(type=markov) y c @nv ar(1) ar(2) ar(3)
    ar(4)
```

estimates a Hamilton-type Markov switching regression model with four non-regime varying autoregressive terms implying mean adjustment for the lagged endogenous.

```
equation eq_lagdep.switchreg(type=markov) y c @nv y(-1) y(-2) y(-3)
    y(-4)
```

specifies an alternate dynamic model in which the lags enter directly into the contemporaneous equation without mean adjustment.

```
equation eq_filardo.switchreg(type=markov) yy_dalt c @nv ar(1)
    ar(2) ar(3) ar(4) @prv c yy_ldalt
```

estimates a 2 state model with non-varying AR(4) and transition matrix probability regressor YY_LDALT.

Cross-references

See Chapter 39. "Switching Regression," beginning on page 671 of *User's Guide II* for a description of the switching regression methodology.

See also `Equation::rgmprobs` (p. 232), `Equation::transprobs` (p. 252), `Equation::makergmprobs` (p. 198) and `Equation::maketransprobs` (p. 199) for routines that allow you to work with the regime probabilities and transition probabilities.

symmtest	Equation Views

Compute symmetry test for distributed lag variables in an equation estimated with a non-linear ARDL (NARDL) specification.

This view displays a table object with the NARDL symmetry test. The top part of the table is a is a summary of the test. This is followed by three additional sections with test statistics and corresponding p-values for relevant regressors tests for: 1) long-run asymmetry, 2) short-run asymmetry, 3) both long and short-run asymmetry.

Syntax

eq_name.**symmtest**(*options*)

Options

p	Print output.

Example

```
ardl_eq.symmtest
```

computes the NARDL symmetry tests for relevant regressors.

Cross-references

See "Symmetry Test View" on page 360 of the *User's Guide II* for further discussion.

See also `Equation::ardl` (p. 71).

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 ...]*

eq_name.**testadd***(options) arg1 [arg2 arg3 ...] [@***nv** *x1 x2 x3 ...]*

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

For equations estimated using `breakls`, there are two types of added series, those with coefficients that break, and those with coefficients that are non-breaking. The former should be listed before, and the latter should be listed after the optional **@nv** keyword.

Options

prompt	Force the dialog to appear from within a program.
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 Diagnostics" on page 203 of the *User's Guide II* for further discussion.

See also `Equation::testdrop` (p. 247) and `Equation::wald` (p. 267).

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

prompt	Force the dialog to appear from within a program.
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 Diagnostics" on page 203 of the *User's Guide II* for further discussion of testing coefficients.

See also `Equation::testadd` (p. 246) and `Equation::wald` (p. 267).

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 = *series_name*	Group by the specified series.
integer (*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 = *integer* (*default* = 100)	Limit the number of values to use for grouping. Should be used with the "*v*" option.
prompt	Force the dialog to appear from within a program.
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 436 of the *User's Guide II* for a discussion of the Andrews and Hosmer-Lemeshow tests.

threshold	Equation Methods

Estimation by discrete or smooth threshold least squares, including threshold autoregression.

Syntax

eq_name.**threshold***(options) y z1 [z2 z3 ...] [*@**nv** *x1 x2 x3 ...] *@**thresh** *t1 [t2 t3 ...]*

List the dependent variable first, followed by a list of the independent variables that have coefficients that are allowed to vary across threshold, followed optionally by the keyword **@nv** and a list of non-varying coefficient variables.

List a threshold variable or variables (for model selection) or a single integer or range pairs after the keyword **@thresh**. The integer or range pairs indicate a self-exciting model with the lagged dependent variable as the threshold variable.

For smooth threshold equations you may specify variables that are to be included only in the base specification or only in the alternative specification. Base-only variables should be specified in parentheses using the **@base** key, as in "@base(x1) @base(x2) @base(x3 x4)". Alternative-only variables may be specified analogously using the **@alt** key.

Options

Specification Options

type = *arg* (*default* = "discrete")	Type of threshold estimation: "discrete" (discrete), "smooth" (smooth).

Discrete Threshold Options

method = *arg* (*default* = "seqplus1")	Threshold selection method: "seqplus1" (sequential tests of single $l+1$ versus l thresholds), "seqall" (sequential test of all possible $l+1$ versus l thresholds), "glob" (tests of global l vs. no thresholds), "globplus1" (tests of $l+1$ versus l globally determined thresholds), "globinfo" (information criteria evaluation)., "fixedseq" (fixed number of sequentially determined thresholds), "fixedglob" (fixed number of globally determined thresholds), "user" (user-specified thresholds)
nthresh = *arg* (default = 1)	Number of thresholds for fixed number threshold selection methods.

select = *arg*	Sub-method setting (options depend on "method = "). (1) if "method = glob": Sequential ("seq") (default), Highest significant ("high"), $UDmax$ ("udmax"), $WDmax$ ("wdmax"). (2) if "method = globinfo": Schwarz criterion ("bic" or "sic") (default), Liu-Wu-Zidek criterion ("lwz").
trim = *arg (default* = 5)	Trimming percentage for determining minimum segment size (5, 10, 15, 20, 25).
maxthresh = *integer (default* = 5)	Maximum number of thresholds to allow (not applicable if "method = seqall").
maxlevels = *integer (default* = 5)	Maximum number of threshold levels to consider in sequential testing (applicable when "method = sequall").
size = *arg (default* = 5)	Test sizes for use in sequential determination and final test evaluation (10, 5, 2.5, 1) corresponding to 0.10, 0.05, 0.025, 0.01, respectively
heterr	Assume regimes specific error distributions in variance computation.
commondata	Assume a common distribution for the data across segments (only applicable if original equation is estimated with a robust covariance method, "heterr" is not specified).

Smooth Threshold Options

smoothtrans = *arg (default* = "logistic")	Smooth threshold transition function: "logistic" (logistic), "logistic2" (second-order logistic), "exponential" (exponential), "normal" (normal).
smoothstart = *arg (default* = "grid_conc")	Smoth threshold starting value method: or fixed number threshold selection methods: "grid_conc" (grid search with concentrated regression coefficients", "grid_zeros" (grid search with zero regression coefficients), "data" (data-based), "user" (user-specified using the contents of the coefficient vector in the workfile).
smoothst = *arg*	Sub-method setting (options depend on "method = "). (1) if "method = glob": Sequential ("seq") (default), Highest significant ("high"), $UDmax$ ("udmax"), $WDmax$ ("wdmax"). (2) if "method = globinfo": Schwarz criterion ("bic" or "sic") (default), Liu-Wu-Zidek criterion ("lwz").

General Options

w = *arg*	Weight series or expression.
wtype = *arg* (*default* = "istdev")	Weight specification type: inverse standard deviation ("istdev"), inverse variance ("ivar"), standard deviation ("stdev"), variance ("var").
wscale = *arg*	Weight scaling: EViews default ("eviews"), average ("avg"), none ("none"). The default setting depends upon the weight type: "eviews" if "wtype = istdev", "avg" for all others.
cov = *keyword*	Covariance type *(optional)*: "white" (White diagonal matrix), "hac" (Newey-West HAC).
nodf	Do not perform degree of freedom corrections in computing coefficient covariance matrix. The default is to use degree of freedom corrections.
covlag = *arg* (*default* = 1)	Whitening lag specification: *integer* (user-specified lag value), "a" (automatic selection).
covinfosel = *arg* (*default* = "aic")	Information criterion for automatic selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn) (if "lag = a").
covmaxlag = *integer*	Maximum lag-length for automatic selection *(optional)* (if "lag = a"). The default is an observation-based maximum of $T^{1/3}$.
covkern = *arg* (*default* = "bart")	Kernel shape: "none" (no kernel), "bart" (Bartlett, *default*), "bohman" (Bohman), "daniell" (Daniel), "parzen" (Parzen), "parzriesz" (Parzen-Riesz), "parzgeo" (Parzen-Geometric), "parzcauchy" (Parzen-Cauchy), "quadspec" (Quadratic Spectral), "trunc" (Truncated), "thamm" (Tukey-Hamming), "thann" (Tukey-Hanning), "tparz" (Tukey-Parzen).
covbw = *arg* (*default* = "fixednw")	Kernel Bandwidth: "fixednw" (Newey-West fixed), "andrews" (Andrews automatic), "neweywest" (Newey-West automatic), *number* (User-specified bandwidth).
covnwlag = *integer*	Newey-West lag-selection parameter for use in nonparametric kernel bandwidth selection (if "covbw = neweywest").
covbwint	Use integer portion of bandwidth.
prompt	Force the dialog to appear from within a program.
p	Print basic estimation results.

Examples

```
equation eq1.threshold(method=fixedseq, type=discrete) ss_transf c
   ss_transf(-1 to -11) @thresh 2
```

uses the fixed number of thresholds test to determine the optimal threshold in a model regressing SS_TRANSF on the threshold variables C and SS_TRANSF(-1 to -11).

```
equation eq2.threshold(method=fixedseq, type=discrete) ss_transf c
   ss_transf(-1 to -11) @thresh 1 5
```

uses the fixed number of thresholds test to determine the optimal threshold and does model selection over lags of SS_TRANSF from SS_TRANSF(-1) to SS_TRANSF(-5).

```
equation eq3.threshold(method=user, threshold=7.44) ss_transf c
   @nv ss_transf(-1 to -11) @thresh 2
```

estimates the model with one user-specified threshold value. In addition, the variables SS_TRANSF(-1 to -11) are restricted to have common coefficients across the regimes.

Cross-references

See Chapter 35. “Discrete Threshold Regression,” on page 555 and Chapter 36. “Smooth Transition Regression,” on page 569 for a discussion of the various forms of threshold models.

transprobs	Equation Views

Display regime transition probabilities and expected durations for a switching regression equation.

Syntax

equation_name.**transprobs***(options)*

where *equation_name* is the name of an equation estimated using switching regression.

Options

type = *arg* *(default* = “summary”*)*	Transition probability results to display: summary (“default”), transition probabilities (“trans”), expected durations (“expect”). The default summary displays the transition matrix and expected regime durations for constant transition probability models, and descriptive statistics for the transition and expected durations for varying probability models.

view = *arg* *(default* = "graph"*)*	Display method: graph ("graph"), spreadsheet ("sheet"), table ("table"). Applicable when displaying the transition probabilities or expected durations ("type = trans" or "type = expect"). The spreadsheet form represents shows the transition probabilities or regime expected durations in columns and observations in rows. The table form displays the transition probabilities or expected durations in a table (in a single matrix for a time-constant model, and individual matrices for a time-varying model).
prompt	Force the dialog to appear from within a program.
p	Print results.

Examples

```
equation eq1.switchreg(type=markov) y c @nv ar(1) ar(2) ar(3)
eq1.transprobs
```

displays the default summary of the transition probabilities estimated in EQ1.

The command

```
eq1.transprobs(type=trans)
```

displays the transition probabilities in a graph, while

```
eq1.transprobs(type=trans, view=sheet)
```

displays the transition probabilities in a spreadsheet, with each row column representing one of the probabilities and each row representing an observation.

```
eq1.transprobs(type=trans, view=table)
```

displays the transition probabilities in a table.

```
eq1.transprobs(type=expect, view=sheet)
```

displays the expected durations in spreadsheet form.

Cross-references

See "Switching Regression" on page 671 of the *User's Guide II* for discussion.

See also `Equation::maketransprobs` (p. 199).

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

Non-Panel TSLS Options

nocinst	Do not automatically include a constant as an instrument.
w = *arg*	Weight series or expression. Note: *we recommend that, absent a good reason, you employ the default settings* **Inverse std. dev. weights** *("wtype = istdev") with* **EViews default** *scaling ("wscale = eviews") for backward compatibility with versions prior to EViews 7.*
wtype = *arg* *(default* = "istdev"*)*	Weight specification type: inverse standard deviation ("istdev"), inverse variance ("ivar"), standard deviation ("stdev"), variance ("var").
wscale = *arg*	Weight scaling: EViews default ("eviews"), average ("avg"), none ("none"). The default setting depends upon the weight type: "eviews" if "wtype = istdev", "avg" for all others.
cov = *keyword*	Covariance type *(optional)*: "white" (White diagonal matrix), "hac" (Newey-West HAC), "cr" (cluster robust).

nodf	Do not perform degree of freedom corrections in computing coefficient covariance matrix. The default is to use degree of freedom corrections. (For non-cluster robust methods).
covlag = *arg (default* = 1*)*	Whitening lag specification: *integer* (user-specified lag value), "a" (automatic selection).
covinfosel = *arg (default* = "aic"*)*	Information criterion for automatic selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn) (if "lag = a").
covmaxlag = *integer*	Maximum lag-length for automatic selection *(optional)* (if "lag = a"). The default is an observation-based maximum of $T^{1/3}$.
covkern = *arg (default* = "bart"*)*	Kernel shape: "none" (no kernel), "bart" (Bartlett, *default*), "bohman" (Bohman), "daniell" (Daniel), "parzen" (Parzen), "parzriesz" (Parzen-Riesz), "parzgeo" (Parzen-Geometric), "parzcauchy" (Parzen-Cauchy), "quadspec" (Quadratic Spectral), "trunc" (Truncated), "thamm" (Tukey-Hamming), "thann" (Tukey-Hanning), "tparz" (Tukey-Parzen).
covbw = *arg (default* = "fixednw"*)*	Kernel Bandwidth: "fixednw" (Newey-West fixed), "andrews" (Andrews automatic), "neweywest" (Newey-West automatic), *number* (User-specified bandwidth).
covnwlag = *integer*	Newey-West lag-selection parameter for use in nonparametric kernel bandwidth selection (if "covbw = neweywest").
covbwint	Use integer portion of bandwidth.
crtype = *arg (default "cr1")*	Cluster robust weighting method: "cr0" (no finite sample correction), "cr1" (finite sample correction), when "cov = cr".
crname = *arg*	Cluster robust series name, when "cov = cr".
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 estimator coefficient vector as starting values for equations specified by list with AR or MA terms (see also `param` (p. 564) of the *Command and Programming Reference*).

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 TSLS 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". Does not apply to coefficients for AR and MA terms which are instead set to EViews determined default values.
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
numericderiv / -numericderiv	[Do / do not] use numeric derivatives only. If omitted, EViews will follow the global default.
fastderiv / -fastderiv	[Do / do not] use fast derivative computation. If omitted, EViews will follow the global default.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
z	Turn off backcasting in ARMA models.
prompt	Force the dialog to appear from within a program.
p	Print basic estimation results.

Panel TSLS Options

cx = *arg*	Cross-section effects. For fixed effects estimation, use "cx = f"; for random effects estimation, use "cx = r".
per = *arg*	Period effects. For fixed effects estimation, use "cx = f"; for random effects estimation, use "cx = r".
wgt = *arg*	GLS weighting: (default) none, cross-section system weights ("wgt = cxsur"), period system weights ("wgt = persur"), cross-section diagonal weighs ("wgt = cxdiag"), period diagonal weights ("wgt = perdiag").
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").

keepwgts	Keep full set of GLS weights used in estimation with object, if applicable (by default, only small memory weights are saved).
rancalc = *arg* (*default* = "sa")	Random component method: Swamy-Arora ("rancalc = sa"), Wansbeek-Kapteyn ("rancalc = wk"), Wallace-Hussain ("rancalc = wh").
nodf	Do not perform degree of freedom corrections in computing coefficient covariance matrix. The default is to use degree of freedom corrections.
iter = *arg* *(default* = "onec"*)*	Iteration control for GLS specifications: perform one weight iteration, then iterate coefficients to convergence ("iter = onec"), iterate weights and coefficients simultaneously to convergence ("iter = sim"), iterate weights and coefficients sequentially to convergence ("iter = seq"), perform one weight iteration, then one coefficient step ("iter = oneb"). Note that random effects models currently do not permit weight iteration to convergence.
unbalsur	Compute SUR factorization in unbalanced data using the subset of available observations for a cluster.
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default is to use the "C" coefficient vector.
s	Use the current coefficient values in estimator coefficient vector as starting values for equations specified by list with AR terms (see also `param` (p. 564) of the *Command and Programming Reference*).
s = *number*	Determine starting values for equations specified by list with AR terms. Specify a number between zero and one representing the fraction of TSLS estimates computed without AR 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". Does not apply to coefficients for AR terms which are instead set to EViews determined default values.
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.
numericderiv / -numericderiv	[Do / do not] use numeric derivatives only. If omitted, EViews will follow the global default.

fastderiv / -fastderiv	[Do / do not] use fast derivative computation. If omitted, EViews will follow the global default.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
prompt	Force the dialog to appear from within a program.
p	Print estimation results.

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 Chapter 23. "Instrumental Variables and GMM," on page 91 and "Two-Stage Least Squares" on page 899 of the *User's Guide II* for details on two-stage least squares estimation in single equations and systems, respectively.

"Instrumental Variables" on page 1291 of the *User's Guide II* discusses estimation using pool objects, while "Instrumental Variables Estimation" on page 1324 of the *User's Guide II* discusses estimation in panel structured workfiles.

See also `Equation::ls` (p. 181) and `Equation::gmm` (p. 158).

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 *series_name*.
lfname = *series_name*	Store the individual likelihood ratio F-statistics into the series *series_name*.
prompt	Force the dialog to appear from within a program.
p	Print the result of the test.

Examples

```
equation ppp.ls 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 236 of the *User's Guide II* for further discussion.

See also `Equation::chow` (p. 91) and `Equation::rls` (p. 233).

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. 26).

varinf	Equation Views

Variance Inflation Factor (VIF).

Display the Variance Inflation Factors (VIFs). VIFs are a method of measuring the level of collinearity between the regressors in an equation.

Syntax

eq_name.**varinf**

Options

p	Print the results.

Examples

The set of commands:

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

displays the variance inflation factor view of EQ1.

Cross-references

See also "Variance Inflation Factors" on page 207 of *User's Guide II*.

varsel	Equation Methods

Estimation using variable selection.

Syntax

eq_name.**varsel***(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), "gets" (auto-search/GETS), "lasso" (Lasso).
nvars = *int*	Set the number of search regressors. Required for swapwise and combinatorial methods, optional for uni-directional and stepwise methods.
w = *arg*	Weight series or expression. Note: *we recommend that, absent a good reason, you employ the default settings* **Inverse std. dev. weights** *("wtype = istdev") with* **EViews default** *scaling ("wscale = eviews") for backward compatibility with versions prior to EViews 7.*
wtype = *arg* (*default* = "istdev")	Weight specification type: inverse standard deviation ("istdev"), inverse variance ("ivar"), standard deviation ("stdev"), variance ("var").
wscale = *arg*	Weight scaling: EViews default ("eviews"), average ("avg"), none ("none"). The default setting depends upon the weight type: "eviews" if "wtype = istdev", "avg" for all others.
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
prompt	Force the dialog to appear from within a program.
p	Print estimation results.

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.

fmaxstep = *int* *(default = 1000)*	Set the maximum number of steps forward.
bmaxstep = *int* *(default = 1000)*	Set the maximum number of steps backward.
tmaxstep = *int* *(default = 2000)*	Set the maximum total number of steps.

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.

Auto-search/GETS method options

pval = *number* *(default = 0.05)*	Set the terminal condition p-value used to determine the stopping point of each search path
nolm	Do not perform AR LM diagnostic test.
arpval = *number* *(default = 0.025)*	Set p-value used in AR LM diagnostic test.
arlags = *int (default = 1)*	Set number of lags used in AR LM diagnostic test.
noarch	Do not perform ARCH LM diagnostic test.
archpval = *number* *(default = 0.025)*	Set p-value used in ARCH LM diagnostic test.
archlags = *int (default = 1)*	Set number of lags used in ARCH LM diagnostic test.
nojb	Do not perform Jarque-Bera normality diagnostic test.
jbpval = *number* *(default = 0.025)*	Set p-value used in Jarque-Bera normality diagnostic test.
nopet	Do not perform Parsimonious Encompassing diagnostic test.
petpval = *number* *(default = 0.025)*	Set p-value used in Parsimonious Encompassing diagnostic test.
nogum	Do not include the general model as a candidate for model selection.

noempty	Do not include the empty model as a candidate for model selection.
ic = *arg*	Set the information criterion used in model selection: "AIC" (Akaike information criteria, default), "BIC" (Schwarz information criteria), "HQ" (Hannan-Quin criteria).
blocks = *int*	Override the EViews' determination of the number of blocks in which to split the estimation sample.

Lasso method options

Penalty Options

ytrans = *arg* *(default* = "none")	Scaling of the dependent variable: "none" (none), "L1" (L1), "L2" (L2), "stdsmpl" (sample standard deviation), "stdpop" (population standard deviation), "minmax" (min-max).
xtrans = *arg* *(default* = "stdpop")	Scaling of the regressor variables: "none" (none), "L1" (L1 norm), "L2" (L2 norm), "stdsmpl" (sample standard deviation), "stdpop" (population standard deviation), "minmax" (min-max).
lambda = *arg*	Value of the penalty parameter. Can be a single number, list of space-delimited numbers, a workfile series object, or left blank for a EViews determined path (default). Values must be zero or greater.
nlambdas = *integer* *(default = 100)*	Number of penalty values for EViews-supplied list.
nlambdamin = *integer* *(default = 5)*	Minimum number of lambda values in the path before applying stopping rules.
minddev = *arg* *(default* = 1e-05*)*	Minimum change in deviance fraction to continue estimation. Truncate path estimation if relative change in deviance is smaller than this value.

maxedev = *arg* *(default* = 0.99*)*	Maximum of deviance explained fraction attained to terminate estimation. Truncate path estimation if fraction of null deviance explained is larger than this value.
maxvars = *arg*	Maximum number of regressors in the model. Truncate path estimation if the number of coefficients (including those for non-penalized variables like the intercept) reaches this value.
maxvarsratio = *arg*	Maximum number of regressors in the model as a fraction of the number of observations. Truncate path estimation if the number of coefficients (including those for non-penalized variables like the intercept) divided by the number of observations reaches this value.

Cross Validation Options

cvmethod = *arg* *(default* = "kfold_cv")	Cross-validation method: "kfold" (k-fold), "simple" (simple split), "mcarlo" (Monte Carlo), "leavepout" (leave-P-out), "leave1out" (leave-1-out), "rolling" (rolling window), "expanding" (expanding window).
cvmeasure = *arg* *(default* = "mse")	Cross-validation fit measure: "mse" (mean-squared error), "r2" (R-squared), "mae" (mean absolute error), "mape" (mean absolute percentage error), "smape" (symmetric mean absolute percentage error).
cvnfolds = *arg* *(default* = 5)	Number of folds for K-fold cross-validation. For "cvmethod = kfold".
cvftrain = *arg* *(default* = 0.8)	Proportion of data for split and Monte Carlo methods. For "cvmethod = simple" and "cvmethod = mcarlo".
cvnreps = *arg* *(default* = 1)	Number of Monte Carlo method repetitions. For "cvmethod = mcarlo".
cvleaveout = *arg* *(default* = 2)	Number of data points left out for leave-p-out method. For "cvmethod = leavepout".
cvnwindows = *arg* *(default* = 4)	Number of windows for rolling window cross-validation method. For "cvmethod = rolling".
cvinitial = *arg* *(default* = 12)	Number of initial data points in the training set for expanding cross-validation. For "cvmethod = expanding".

cvpregap = *arg* (*default* = 0)	Number of observations between end of training set and beginning of test set. For "cvmethod = simple", "cvmethod = rolling" and "cvmethod = expanding".
cvhorizon = *arg* (*default* = 1)	Number of observation in the test set. For "cvmethod = rolling" and "cvmethod = expanding".
cvpostgap = *arg* (*default* = 0)	Number of observations between end of test set and beginning of next training set for rolling window or between end of test set and end of next training set for expanding window. For "cvmethod = rolling" and "cvmethod = expanding"

Random Number Options

seed = *positive_integer* from 0 to 2,147,483,647	Seed the random number generator. If not specified, EViews will seed random number generator with a single integer draw from the default global random number generator.
rnd = *arg* (*default* = "kn" or method previously set using `rndseed` (p. 577) in the *Command and Programming Reference)*.	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").

Other Options

coefmin = *vector_name, number*	Vector of individual coefficient minimum values, containing negative or missing values sized to and matching the order of the variables in the specification, or a negative value for the minimum for all coefficients. Missing values in the vector should be used to indicate that the coefficient is unrestricted. If a vector of values is provided and individual minimums are specified using one or more `@vw` regressors, the vector values will be applied first, then overwritten by the individual values.

coefmax = *vector_name, number*	Vector of individual coefficient maximum values, containing positive or missing values sized to and matching the order of the variables in the specification, or a positive value for the maximum for all coefficients. Missing values in the vector should be used to indicate that the coefficient is unrestricted. If a vector of values is provided and individual maximums are specified using one or more `@vw` regressors, the vector values will be applied first, then overwritten by the individual values.
maxit = *integer*	Maximum number of iterations.
conv = *scalar*	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.
w = *arg*	Weight series or expression.
wtype = *arg* *(default* = "istdev"*)*	Weight specification type: inverse standard deviation ("istdev"), inverse variance ("ivar"), standard deviation ("stdev"), variance ("var").
wscale = *arg*	Weight scaling: EViews default ("eviews"), average ("avg"), none ("none"). The default setting depends upon the weight type: "eviews" if "wtype = istdev", "avg" for all others.
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the rotation output.
prompt	Force the dialog to appear from within a program.
p	Print estimation results.

Examples

```
eq1.varsel(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 Chapter 22. "Regression Variable Selection," on page 65 of *User's Guide II* for extensive discussion.

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 210 of the *User's Guide II* for a discussion of Wald tests.

See also `Equation::cellipse` (p. 87), `Equation::testdrop` (p. 247), `Equation::testadd` (p. 246).

weakinst	Equation Views

Displays the Weak Instruments Summary

The `weakinst` view of an equation displays the Weak Instrument Summary for equations estimated by TSLS, GMM or LIML. The summary includes both the Cragg-Donald test and Moment Selection Criteria (for TSLS and GMM only).

Syntax

eq_name.**weakinst**

Examples

```
equation eq1.gmm y c x1 x2 @ z1 z2 z3 z4
e1.weakinst
```

estimates and equation via GMM and then displays the weak instrument summary.

Cross-references

See "Weak Instrument Diagnostics" on page 116 of the *User's Guide II* for discussion.

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. 168). 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.
prompt	Force the dialog to appear from within a program.
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 227 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 958 of the *User's Guide II*.

See also `Equation::hettest` (p. 168) for a more full-featured version of this test.

Factor

Factor analysis object.

Factor Declaration

factor .....factor object declaration (p. 280).

To declare a factor object, use the `factor` keyword, followed by a name to be given to the object. See also `factest` (p. 447).

Factor Methods

gls .....generalized least squares estimation (p. 283).
ipf.....iterated principal factors estimation (p. 288).
ml.....maximum likelihood estimation (p. 297).
pace.....non-iterative partitioned covariance estimation (PACE) (p. 304).
pf.....principal factors estimation (p. 309).
uls .....unweighted least squares estimation (p. 324).

Factor Views

anticov.....display the anti-image covariance matrix of the observed matrix (p. 275).
display.....display table, graph, or spool in object window (p. 277).
eigen.....display table or graph of eigenvalues of observed, scaled observed, or reduced covariance matrix (p. 278).
fitstats.....show table of Goodness-of-Fit statistics (p. 281).
fitted.....show fitted and reproduced covariances (p. 281).
fsel .....display results of Bai and Ng or Ahn and Horenstein factor selection techniques (p. 282).
loadings .....display loadings tables or graphs (p. 293).
maxcor .....display maximum absolute correlations for the observed covariance matrix (p. 297).
msa.....compute and display Kaiser's Measure of Sampling Adequacy (MSA) (p. 302).
observed .....display observed covariance matrix, scaled covariance matrix, or number of observations used in analysis (p. 302).
output.....display main factor analysis estimation output (p. 303).
partcor.....show observed partial correlation matrix (p. 308).
reduced.....display reduced covariance matrix using initial or final uniquenesses (p. 313).
resids.....display residual covariance estimates (p. 314).
rotateout .....show rotated factors and rotation estimation results (p. 319).

scores ..... compute factor score coefficients and scores and display results (p. 320).
smc ..... display table of squared multiple correlations for the observed covariance matrix (p. 323).
structure ..... display factor structure matrix (p. 323).

Factor Procs

clearhist ..... clear the contents of the history attribute (p. 276).
clearremarks ..... clear the contents of the remarks attribute (p. 276).
copy ..... creates a copy of the factor (p. 277).
displayname ..... set display name for factor object (p. 278).
factnames ..... specify names for factors (p. 280).
label ..... label view of factor object (p. 292).
makescores ..... compute and save factor score scores series (p. 294).
olepush ..... push updates to OLE linked objects in open applications (p. 303).
rotate ..... perform an orthogonal or oblique factor rotation (p. 314).
rotateclear ..... clear existing rotation results (p. 318).
setattr ..... set the value of an object attribute (p. 319).

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

@attr("*arg*") ..... string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@command..... full command line form of the Factor estimation command. Note this is a combination of `@method`, `@options`, and `@spec`.
@description..... string containing the Factor object's description (if available).
@detailedtype ..... returns a string with the object type: "FACTOR".
@displayname..... returns the Factor object's display name. If the Factor object has no display name set, the name is returned.
@factnames ..... factor names.
@method ..... command line form of the Factor estimation method type.
@name ..... returns the Factor object's name.
@options..... command line form of estimation options.
@smpl ..... sample used for estimation.
@spec..... original factor specification.
@type ..... returns a string with the object type: "FACTOR".
@updatetime..... returns a string representation of the time and date at which the Factor was last updated.
@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 1498 of *User's Guide II*. See also `Factor::observed` (p. 302), `Factor::partcor` (p. 308), `Factor::smc` (p. 323).

clearhist	Factor Procs

Clear the contents of the history attribute.

Removes the factor's history attribute, as shown in the label view of the factor.

Syntax

factor_name.**clearhist**

Examples

```
f1.clearhist
f1.label
```

The first line removes the history from the factor F1, and the second line displays the label view of F1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names.

See also `Factor::label` (p. 292).

clearremarks	Factor Procs

Clear the contents of the remarks attribute.

Removes the factor's remarks attribute, as shown in the label view of the factor.

Syntax

factor_name.**clearremarks**

Examples

```
f1.clearremarks
f1.label
```

The first line removes the remarks from the factor F1, and the second line displays the label view of F1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names.

See also `Factor::label` (p. 292).

copy	Factor Procs

Creates a copy of the factor.

Creates either a named or unnamed copy of the factor.

Syntax

factor_name.**copy**

factor_name.**copy** *dest_name*

Examples

```
f1.copy
```

creates an unnamed copy of the factor F1.

```
f1.copy f2
```

creates F2, a copy of the factor F1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

display	Factor Views

Display table, graph, or spool output in the factor object window.

Display the contents of a table, graph, or spool in the window of the factor object.

Syntax

factor_name.**display** *object_name*

Examples

```
factor1.display tab1
```

Display the contents of the table TAB1 in the window of the object FACTOR1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

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 123 of *User's Guide I* for a discussion of labels and display names. See also `Factor::label` (p. 292).

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.
prompt	Force the dialog to appear from within a program.
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 1500 of *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. 283), `ipf` (p. 288), `ml` (p. 297), `pace` (p. 304), `pf` (p. 309), and `uls` (p. 324). 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 60. "Factor Analysis," on page 1485 of *User's Guide II* provides basic information on factor analysis.

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 1524 of *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 1498 of *User's Guide II*. See also `Factor::reduced` (p. 313).

fsel — Factor Views

Display results of Bai and Ng or Ahn and Horenstein factor selection techniques.

Syntax

factor_name.**fsel**

Only relevant for factor models estimated using the "n = bn" or "n = ah" methods for determining the number of factors to retain.

Options

p	Print the results.

Examples

```
factor f1.ml(n=bn) group01
f1.fsel(p)
```

estimates a factor model using the Bai and Ng method for determining the number of factors, and then displays and prints a table of selection results.

Cross-references

See "Number of Factors" on page 1519 of *User's Guide II* for discussion of methods for selecting the number of factors retained in factor analysis.

See "Bai and Ng" on page 703 and "Ahn and Horenstein" on page 704 of *User's Guide I* for a discussion of these specific methods.

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 = *integer*	Maximum number of iterations.
conv = *scalar*	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.
prompt	Force the dialog to appear from within a program.
p	Print basic estimation results.

Number of Factors Options

n = *arg* or fsmethod = *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), "bn" (Bai and Ng (2002)), "ah" (Ahn and Horenstein (2013)), *integer* (user-specified integer value).
eiglimit = *number* *(default* = 1*)*	Limit value for retaining factors using the eigenvalue comparison (where "n = mineigen").
varlimit = *number* *(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 = *integer* *(default* = 100*)*	Number of parallel analysis repetitions. For parallel analysis only ("n = parallel").
pquant = *number*	Quantile value for parallel analysis comparison (if not specified, the mean value will be employed). For parallel analysis only ("n = parallel").
pseed = *positive integer*	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").
prnd = *arg* *(default* = "kn" or method previously set using `rndseed` (p. 577) in the *Command and Programming Reference)*	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").

mfmethod = *arg* *(default = "user")*	Maximum number of components used by selection methods: "schwert" (Schwert's rule, *default*), "ah" (Ahn and Horenstein's (2013) suggestion), "rootsize" ($\min(\sqrt{N}, \sqrt{T})$), "size" ($\min(N, T)$), "user" (user specified value), where N is the number of series and T is the number of observations. (1) For use with all components retention methods apart from user-specified ("fsmethod = user"). (2) If setting "mfmethod = user", you may specify the maximum number of components using "rmax = ". (3) Schwert's rule sets the maximum number of components using the rule: let $$L(k) = k(T/100)^{1/4}$$ for $k \in \{2, 4, 6, 8, 10, 12\}$ and let $T^* = \min(N, T)$; then the default maximum lag is given by $$L^* = \max_k\{L(k) \mid L(k) < T^*\}$$
rmax = *arg* *(default* = all*)*	User-specified maximum number of factors to retain (for use when "mfmethod = user").
fsic = *arg (default* = avg*)*	Factor selection criterion (when "fsmethod = bn"): "icp1" (ICP1), "icp2" (ICP2), "icp3" (ICP3), "pcp1" (PCP1), "pcp2" (PCP1), "pcp3" (ICP3), "avg" (average of all criteria ICP1 through PCP3). Factor selection criterion (when "fsmethod = ah"): "er" (eigenvalue ratio), "gr" (growth ratio), "avg" (average of eigenvalue ratio and growth ratio). Factor selection criterion (when "fsmethod = simple"): "min" (minimum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "max" (maximum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "avg" (average the optimal number of factors as specified by the min and max rule, then round to the nearest integer).
demeantime	Demeans observations across time prior to component selection procedures, when "n = bn" or "n = ah".
sdizetime	Standardizes observations across time prior to component selection procedures, when "n = bn" or "n = ah".
demeancross	Demeans observations across cross-sections prior to component selection procedures, when "n = bn" or "n = ah".

sdizecross	Standardizes observations across cross-sections prior to component selection procedures, when "n = bn" or "n = ah".

Initial Communalities Options

priors = *arg*	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").
priorfrac = *number*	User-specified common fraction (between 0 and 1) to be used when "priors = frac".
priorunique = *arg*	Vector of initial *uniqueness* 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

cov = *arg* *(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.
wgt = *name* (optional)	Name of series containing weights.

wgtmethod = *arg (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.
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.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 60. "Factor Analysis," on page 1485 of *User's Guide II* for a general discussion of factor analysis. The various estimation methods are described in "Estimation Methods" on page 1521 of *User's Guide II*.

See also `Factor::ipf` (p. 288), `Factor::ml` (p. 297), `Factor::pace` (p. 304), `Factor::pf` (p. 309), `Factor::uls` (p. 324).

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

heywood = *arg (default* = "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).
maxit = *integer*	Maximum number of iterations.
conv = *scalar*	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.
prompt	Force the dialog to appear from within a program.
p	Print basic estimation results.

Number of Factors Options

n = *arg* or fsmethod = *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), "bn" (Bai and Ng (2002)), "ah" (Ahn and Horenstein (2013)), *integer* (user-specified integer value).
eiglimit = *number* *(default* = 1*)*	Limit value for retaining factors using the eigenvalue comparison (where "n = mineigen").
varlimit = *number* *(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 = *integer* *(default* = 100*)*	Number of parallel analysis repetitions. For parallel analysis only ("n = parallel").
pquant = *number*	Quantile value for parallel analysis comparison (if not specified, the mean value will be employed). For parallel analysis only ("n = parallel").
pseed = *positive integer*	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").
prnd = *arg* *(default* = "kn" or method previously set using `rndseed` (p. 577) in the *Command and Programming Reference)*	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").

mfmethod = *arg* *(default = "user")*	Maximum number of components used by selection methods: "schwert" (Schwert's rule, *default*), "ah" (Ahn and Horenstein's (2013) suggestion), "rootsize" ($\min(\sqrt{N}, \sqrt{T})$), "size" ($\min(N, T)$), "user" (user specified value), where N is the number of series and T is the number of observations. (1) For use with all components retention methods apart from user-specified ("fsmethod = user"). (2) If setting "mfmethod = user", you may specify the maximum number of components using "rmax = ". (3) Schwert's rule sets the maximum number of components using the rule: let $$L(k) = k(T/100)^{1/4}$$ for $k \in \{2, 4, 6, 8, 10, 12\}$ and let $T^* = \min(N, T)$; then the default maximum lag is given by $$L^* = \max_k\{L(k) \mid L(k) < T^*\}$$
rmax = *arg* *(default* = all*)*	User-specified maximum number of factors to retain (for use when "mfmethod = user").
fsic = *arg (default* = avg*)*	Factor selection criterion (when "fsmethod = bn"): "icp1" (ICP1), "icp2" (ICP2), "icp3" (ICP3), "pcp1" (PCP1), "pcp2" (PCP1), "pcp3" (ICP3), "avg" (average of all criteria ICP1 through PCP3). Factor selection criterion (when "fsmethod = ah"): "er" (eigenvalue ratio), "gr" (growth ratio), "avg" (average of eigenvalue ratio and growth ratio). Factor selection criterion (when "fsmethod = simple"): "min" (minimum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "max" (maximum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "avg" (average the optimal number of factors as specified by the min and max rule, then round to the nearest integer).
demeantime	Demeans observations across time prior to component selection procedures, when "n = bn" or "n = ah".
sdizetime	Standardizes observations across time prior to component selection procedures, when "n = bn" or "n = ah".
demeancross	Demeans observations across cross-sections prior to component selection procedures, when "n = bn" or "n = ah".

sdizecross	Standardizes observations across cross-sections prior to component selection procedures, when "n = bn" or "n = ah".

Initial Communalities Options

priors = *arg*	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").
priorfrac = *number*	User-specified common fraction (between 0 and 1) to be used when "priors = frac".
priorunique = *arg*	Vector of initial *uniqueness* 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

cov = *arg* *(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.
wgt = *name* (optional)	Name of series containing weights.
wgtmethod = *arg* *(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.
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.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 60. "Factor Analysis," on page 1485 of *User's Guide II* for a general discussion of factor analysis. The various estimation methods are described in "Estimation Methods" on page 1521 of *User's Guide II*.

See also `Factor::gls` (p. 283), `Factor::ml` (p. 297), `Factor::pace` (p. 304), `Factor::pf` (p. 309), `Factor::uls` (p. 324).

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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User's Guide I* for a discussion of labels.

See also `Factor::displayname` (p. 278).

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).
prompt	Force the dialog to appear from within a program (for loadings graphs only)
p	Print results.

Graph Options

mult =*arg* (*default* = "first")	Multiple series handling: plot first against remainder ("first"), plot as x-y pairs ("pair"), lower-triangular plot ("lt").
nocenter	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 1518 of *User's Guide II* for a general discussion of the factor model, and "Loadings Views" on page 1499 of *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. 280)) 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 = *arg* (*default* = "exact")	Exact coefficient ("exact"), coarse adjusted factor coefficients ("coefs"), coarse adjusted factor loadings ("loadings").
coef = *arg* (*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.
coarse = arg (*default* = "unrestrict")	Method for computing the coarse (-1, 0, 1) scores coefficients (Grice, 1991a): Unrestricted -- ("unrestrict") coef weights set based only on sign; Unique–recode ("recode") only element with highest value is coded to a non-zero value; Unique–drop ("drop") only elements with loadings not in excess of the threshold are set to non-zero values. For "type = coefs" and "type = loadings" specifications.

cutoff = *number (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 = *arg (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).
n = *arg*	*(Optional)* Name of group object to contain the factor score series.
coefout	*(Optional)* Name of matrix in which to save the factor score coefficient matrix.
prompt	Force the dialog to appear from within a program.

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 1494 of *User's Guide II* and "Scoring," on page 1529 of *User's Guide II*. See also `Factor::scores` (p. 320).

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

p	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. 275), `Factor::observed` (p. 302), and `Factor::partcor` (p. 308).

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

rescale	Rescale the uniqueness and loadings estimates so that they match the observed variances.
maxit = *integer*	Maximum number of iterations.
conv = *scalar*	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.
prompt	Force the dialog to appear from within a program.
p	Print basic estimation results.

Number of Factors Options

n = *arg* or fsmethod = *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), "bn" (Bai and Ng (2002)), "ah" (Ahn and Horenstein (2013)), *integer* (user-specified integer value).
eiglimit = *number* (*default* = 1)	Limit value for retaining factors using the eigenvalue comparison (where "n = mineigen").
varlimit = *number* (*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 = *integer* *(default* = 100*)*	Number of parallel analysis repetitions. For parallel analysis only ("n = parallel").
pquant = *number*	Quantile value for parallel analysis comparison (if not specified, the mean value will be employed). For parallel analysis only ("n = parallel").
pseed = *positive integer*	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").
prnd = *arg* *(default* = "kn" or method previously set using `rndseed` (p. 577) in the *Command and Programming Reference)*	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").
mfmethod = *arg* *(default* = *"user")*	Maximum number of components used by selection methods: "schwert" (Schwert's rule, *default*), "ah" (Ahn and Horenstein's (2013) suggestion), "rootsize" ($\min(\sqrt{N}, \sqrt{T})$), "size" ($\min(N, T)$), "user" (user specified value), where N is the number of series and T is the number of observations. (1) For use with all components retention methods apart from user-specified ("fsmethod = user"). (2) If setting "mfmethod = user", you may specify the maximum number of components using "rmax = ". (3) Schwert's rule sets the maximum number of components using the rule: let $$L(k) = k(T/100)^{1/4}$$ for $k \in \{2, 4, 6, 8, 10, 12\}$ and let $T^* = \min(N, T)$; then the default maximum lag is given by $$L^* = \max_k \{L(k) \mid L(k) < T^*\}$$
rmax = *arg* *(default* = all*)*	User-specified maximum number of factors to retain (for use when "mfmethod = user").

fsic = *arg (default* = avg)	Factor selection criterion (when "fsmethod = bn"): "icp1" (ICP1), "icp2" (ICP2), "icp3" (ICP3), "pcp1" (PCP1), "pcp2" (PCP1), "pcp3" (ICP3), "avg" (average of all criteria ICP1 through PCP3). Factor selection criterion (when "fsmethod = ah"): "er" (eigenvalue ratio), "gr" (growth ratio), "avg" (average of eigenvalue ratio and growth ratio). Factor selection criterion (when "fsmethod = simple"): "min" (minimum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "max" (maximum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "avg" (average the optimal number of factors as specified by the min and max rule, then round to the nearest integer).
demeantime	Demeans observations across time prior to component selection procedures, when "n = bn" or "n = ah".
sdizetime	Standardizes observations across time prior to component selection procedures, when "n = bn" or "n = ah".
demeancross	Demeans observations across cross-sections prior to component selection procedures, when "n = bn" or "n = ah".
sdizecross	Standardizes observations across cross-sections prior to component selection procedures, when "n = bn" or "n = ah".

Initial Communalities Options

priors = *arg*	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").
priorfrac = *number*	User-specified common fraction (between 0 and 1) to be used when "priors = frac".
priorunique = *arg*	Vector of initial *uniqueness* 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

cov = *arg* *(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.
wgt = *name* (optional)	Name of series containing weights.
wgtmethod = *arg* *(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.
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.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 60. "Factor Analysis," on page 1485 of *User's Guide II* for a general discussion of factor analysis. The various estimation methods are described in "Estimation Methods" on page 1521 of *User's Guide II*.

See also `Factor::gls` (p. 283), `Factor::ipf` (p. 288), `Factor::ml` (p. 297), `Factor::pace` (p. 304), `Factor::pf` (p. 309), `Factor::uls` (p. 324).

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 "Basic Diagnostic Views" on page 1506 of *User's Guide II* for discussion.

See also `Factor::partcor` (p. 308) and `Factor::anticov` (p. 275).

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.
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(scaled, p)
```

displays and prints the corresponding correlation matrix.

Cross-references

See "Observed Covariances" on page 1498 of *User's Guide II*.

See also Factor::anticov (p. 275), Factor::partcor (p. 308), and Factor::smc (p. 323).

olepush	Factor Procs

Push updates to OLE linked objects in open applications.

Syntax

factor_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

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.

Cross-references

See Chapter 60. "Factor Analysis," on page 1485 of *User's Guide II* for a general discussion of factor analysis. The various estimation methods are described in "Estimation Methods" on page 1521 of *User's Guide II*.

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.
prompt	Force the dialog to appear from within a program.
p	Print basic estimation results.

Number of Factors Options

n = *arg* or fsmethod = *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), "bn" (Bai and Ng (2002)), "ah" (Ahn and Horenstein (2013)), *integer* (user-specified integer value).
eiglimit = *number* *(default* = 1*)*	Limit value for retaining factors using the eigenvalue comparison (where "n = mineigen").
varlimit = *number* *(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 = *integer* *(default* = 100*)*	Number of parallel analysis repetitions. For parallel analysis only ("n = parallel").
pquant = *number*	Quantile value for parallel analysis comparison (if not specified, the mean value will be employed). For parallel analysis only ("n = parallel").
pseed = *positive integer*	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").
prnd = *arg* *(default* = "kn" or method previously set using `rndseed` (p. 577) in the *Command and Programming Reference)*	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").

mfmethod = *arg* *(default = "user")*	Maximum number of components used by selection methods: "schwert" (Schwert's rule, *default*), "ah" (Ahn and Horenstein's (2013) suggestion), "rootsize" ($\min(\sqrt{N}, \sqrt{T})$), "size" ($\min(N, T)$), "user" (user specified value), where N is the number of series and T is the number of observations. (1) For use with all components retention methods apart from user-specified ("fsmethod = user"). (2) If setting "mfmethod = user", you may specify the maximum number of components using "rmax = ". (3) Schwert's rule sets the maximum number of components using the rule: let $$L(k) = k(T/100)^{1/4}$$ for $k \in \{2, 4, 6, 8, 10, 12\}$ and let $T^* = \min(N, T)$; then the default maximum lag is given by $$L^* = \max_k\{L(k) \mid L(k) < T^*\}$$
rmax = *arg* *(default* = all*)*	User-specified maximum number of factors to retain (for use when "mfmethod = user").
fsic = *arg (default* = avg*)*	Factor selection criterion (when "fsmethod = bn"): "icp1" (ICP1), "icp2" (ICP2), "icp3" (ICP3), "pcp1" (PCP1), "pcp2" (PCP1), "pcp3" (ICP3), "avg" (average of all criteria ICP1 through PCP3). Factor selection criterion (when "fsmethod = ah"): "er" (eigenvalue ratio), "gr" (growth ratio), "avg" (average of eigenvalue ratio and growth ratio). Factor selection criterion (when "fsmethod = simple"): "min" (minimum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "max" (maximum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "avg" (average the optimal number of factors as specified by the min and max rule, then round to the nearest integer).
demeantime	Demeans observations across time prior to component selection procedures, when "n = bn" or "n = ah".
sdizetime	Standardizes observations across time prior to component selection procedures, when "n = bn" or "n = ah".
demeancross	Demeans observations across cross-sections prior to component selection procedures, when "n = bn" or "n = ah".

sdizecross	Standardizes observations across cross-sections prior to component selection procedures, when "n = bn" or "n = ah".

Covariance Options

cov = *arg* *(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.
wgt = *name (optional)*	Name of series containing weights.
wgtmethod = *arg* *(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.
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.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 60. "Factor Analysis," on page 1485 of *User's Guide II* for a general discussion of factor analysis. The various estimation methods are described in "Estimation Methods" on page 1521 of *User's Guide II*.

See also `Factor::gls` (p. 283), `Factor::ipf` (p. 288), `Factor::ml` (p. 297), `Factor::pf` (p. 309), `Factor::uls` (p. 324).

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 1498 of *User's Guide II*.

See also `Factor::anticov` (p. 275), `Factor::observed` (p. 302), and `Factor::smc` (p. 323).

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

prompt	Force the dialog to appear from within a program.
p	Print basic estimation results.

Number of Factors Options

n = *arg* or fsmethod = *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), "bn" (Bai and Ng (2002)), "ah" (Ahn and Horenstein (2013)), *integer* (user-specified integer value).
eiglimit = *number* *(default* = 1*)*	Limit value for retaining factors using the eigenvalue comparison (where "n = mineigen").

varlimit = *number* *(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 = *integer* *(default* = 100*)*	Number of parallel analysis repetitions. For parallel analysis only ("n = parallel").
pquant = *number*	Quantile value for parallel analysis comparison (if not specified, the mean value will be employed). For parallel analysis only ("n = parallel").
pseed = *positive integer*	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").
prnd = *arg* *(default* = "kn" or method previously set using `rndseed` (p. 577) in the *Command and Programming Reference)*	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").
mfmethod = *arg* *(default = "user")*	Maximum number of components used by selection methods: "schwert" (Schwert's rule, *default*), "ah" (Ahn and Horenstein's (2013) suggestion), "rootsize" ($\min(\sqrt{N}, \sqrt{T})$), "size" ($\min(N, T)$), "user" (user specified value), where N is the number of series and T is the number of observations. (1) For use with all components retention methods apart from user-specified ("fsmethod = user"). (2) If setting "mfmethod = user", you may specify the maximum number of components using "rmax = ". (3) Schwert's rule sets the maximum number of components using the rule: let $$L(k) = k(T/100)^{1/4}$$ for $k \in \{2, 4, 6, 8, 10, 12\}$ and let $T^* = \min(N, T)$; then the default maximum lag is given by $$L^* = \max_k \{L(k) \mid L(k) < T^*\}$$

rmax = *arg* *(default* = all*)*	User-specified maximum number of factors to retain (for use when "mfmethod = user").
fsic = *arg (default* = avg*)*	Factor selection criterion (when "fsmethod = bn"): "icp1" (ICP1), "icp2" (ICP2), "icp3" (ICP3), "pcp1" (PCP1), "pcp2" (PCP1), "pcp3" (ICP3), "avg" (average of all criteria ICP1 through PCP3). Factor selection criterion (when "fsmethod = ah"): "er" (eigenvalue ratio), "gr" (growth ratio), "avg" (average of eigenvalue ratio and growth ratio). Factor selection criterion (when "fsmethod = simple"): "min" (minimum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "max" (maximum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "avg" (average the optimal number of factors as specified by the min and max rule, then round to the nearest integer).
demeantime	Demeans observations across time prior to component selection procedures, when "n = bn" or "n = ah".
sdizetime	Standardizes observations across time prior to component selection procedures, when "n = bn" or "n = ah".
demeancross	Demeans observations across cross-sections prior to component selection procedures, when "n = bn" or "n = ah".
sdizecross	Standardizes observations across cross-sections prior to component selection procedures, when "n = bn" or "n = ah".

Initial Communalities Options

priors = *arg*	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").
priorfrac = *number*	User-specified common fraction (between 0 and 1) to be used when "priors = frac".
priorunique = *arg*	Vector of initial *uniqueness* 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

cov = *arg* *(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.
wgt = *name (optional)*	Name of series containing weights.
wgtmethod = *arg* *(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.
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=pace) 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=pace) 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 60. "Factor Analysis," on page 1485 of *User's Guide II* for a general discussion of factor analysis. The various estimation methods are described in "Estimation Methods" on page 1521 of *User's Guide II*.

See also `Factor::gls` (p. 283), `Factor::ipf` (p. 288), `Factor::ml` (p. 297), `Factor::pace` (p. 304), `Factor::uls` (p. 324).

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 1498 of *User's Guide II*.

See also `Factor::fitted` (p. 281).

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 Chapter 60. "Factor Analysis," on page 1485 of *User's Guide II* for a general discussion of factor analysis. The various estimation methods are described in "Estimation Methods" on page 1521 of *User's Guide II*.

See also `fit` (p. 452).

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:

type = *arg* *(default* = "orthog"*)*	Orthogonal ("orthog") or oblique ("oblique") rotation (ignored if method is not supported, *e.g,* "orthogonal Harris-Kaiser" or "oblique Entropy Ratio").
method = *arg* *(default* = "varimax"*)*	Method (objective) for the rotation. See keywords below
param = *arg*	Rotation parameter, if applicable (see description below).
preparam = *arg* *(default* = 1, Varimax)	Orthomax pre-rotation parameter (for "method = hk" and "method = promax").

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 1526 of *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. *(default* = 0, Quartimax*)*
Generalized Crawford-Ferguson	4	Vector of weights for (in order): total squares, variable complexity, factor complexity, diagonal quartics. *(no default)*
Geomin	1	Epsilon offset. *(default* = 0.01*)*

Harris-Kaiser (case II)	2	Power parameter *(default* = 0, independent cluster solution*),* Orthomax pre-rotation parameter. *(default* = 1, Varimax*)*
Oblimin	1	Deviation from orthogonality. *(default* = 0, Quartimin*)*
Orthomax	1	Factor complexity weight. *(default* = 1, Varimax*)*
Promax	2	Power parameter *(default* = 3*)*, Orthomax pre-rotation parameter *(default* = *1*, Varimax*)*.
Simplimax	1	Fraction of near-zero loadings. *(default* = 0.75*)*
Target	1	Name of $p \times m$ matrix of target loadings. Missing values correspond to unrestricted elements. *(no default)*

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"*)*	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.
preps = *integer (default* = 25*)*	Number of random prior rotations to evaluate (maximum 10000).
pname = *arg*	Name of matrix containing prior rotation.
pseed = *positive integer*	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.

prnd = *arg* (*default* = "kn" or method previously set using `rndseed` (p. 577) in the *Command and Programming Reference*)	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").
m = *integer*	Maximum number of iterations.
c = *scalar*	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.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the rotation output.
p	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 1493 of *User's Guide II* for a discussion of factor rotation.

See also `Factor::rotateout` (p. 319) and `Factor::rotateclear` (p. 318).

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 1493 of *User's Guide II* for a discussion of factor rotation.

See also `Factor::rotate` (p. 314) and `Factor::rotateout` (p. 319).

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

```
f1.rotate
f1.output
f1.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 1493 of *User's Guide II* for a discussion of factor rotation.

See also `Factor::rotate` (p. 314) and `Factor::rotateclear` (p. 318).

setattr	Factor Procs

Set the object attribute.

Syntax

factor_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the "revised" attribute in the object A to the string "never", and extracts the attribute into the string object S.

Cross-references

See "Adding Custom Attributes in the Label View" on page 123 and "Adding Your Own Label Attributes" on page 70 of *User's Guide I*.

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).
type = *arg* *(default* = "exact"*)*	Exact coefficient ("exact"), coarse adjusted factor coefficients ("coefs"), coarse adjusted factor loadings ("loadings").
coef = *arg* *(default* = "reg"*)*	Method for computing the exact or coarse adjusted 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.
coarse = arg *(default* = "unrestrict"*)*	Method for computing the coarse (-1, 0, 1) scores coefficients (Grice, 1991a): Unrestricted -- ("unrestrict") coef weights set based only on sign; Unique–recode ("recode") only element with highest value is coded to a non-zero value; Unique–drop ("drop") only elements with loadings not in excess of the threshold are set to non-zero values. For "type = coefs" and "type = loadings" specifications.
cutoff = *number* *(default* = 0.3*)*	Cutoff value for coarse scores coefficient calculations (Grice, 1991a). For "type = coefs" specifications, the cutoff value represents the fraction of the largest absolute coefficient weight per factor against which the 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.
moment = *arg* *(default* = "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.

df	Degrees-of-freedom correct the observables variances computed when "moment = obs" (divide sums-of-squares by $n-1$ instead of n).
coefout	(Optional) Name of matrix in which to save factor score coefficient matrix.
prompt	Force the dialog to appear from within a program.
p	Print results.

Graph Options

mult = *arg* *(default* = "first"*)*	Multiple series handling for graphs: plot first against remainder ("first"), plot as x-y pairs ("pair"), lower-triangular plot ("lt")
nocenter	Do not center graphs around the origin.
labels = *arg*, *(default* = "outlier"*)*	Observation labels for scores: outliers only ("outlier"), all points ("all"), none ("none").
labelprob = *number*	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).
userscale = *arg*	User-scale factor to be applied to the unscaled loadings (setting this option overrides the automatic scaling).
autoscale = *arg* *(default* = 1*)*	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 1494 and "Scoring," on page 1529 of *User's Guide II*.

See also `Factor::makescores` (p. 294).

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

p	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 "*Indeterminacy Indices*," on page 1530 and "Communality Estimation," on page 1522 of *User's Guide II*.

See also `Factor::observed` (p. 302), `Factor::anticov` (p. 275), and `Factor::maxcor` (p. 297).

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 1499 of *User's Guide II* for details.

See Factor::rotate (p. 314) and Factor::loadings (p. 293).

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

rescale	Rescale the uniqueness and loadings estimates so that they match the observed variances.
maxit = *integer*	Maximum number of iterations.
conv = *scalar*	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.
prompt	Force the dialog to appear from within a program.
p	Print basic estimation results.

Number of Factors Options

n = *arg* or fsmethod = *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), "bn" (Bai and Ng (2002)), "ah" (Ahn and Horenstein (2013)), *integer* (user-specified integer value).
eiglimit = *number* *(default* = 1*)*	Limit value for retaining factors using the eigenvalue comparison (where "n = mineigen").
varlimit = *number* *(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 = *integer* *(default* = 100*)*	Number of parallel analysis repetitions. For parallel analysis only ("n = parallel").
pquant = *number*	Quantile value for parallel analysis comparison (if not specified, the mean value will be employed). For parallel analysis only ("n = parallel").
pseed = *positive integer*	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").

prnd = *arg* *(default* = "kn" or method previously set using `rndseed` (p. 577) in the *Command and Programming Reference)*	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").
mfmethod = *arg* *(default = "user")*	Maximum number of components used by selection methods: "schwert" (Schwert's rule, *default*), "ah" (Ahn and Horenstein's (2013) suggestion), "rootsize" ($\min(\sqrt{N}, \sqrt{T})$), "size" ($\min(N, T)$), "user" (user specified value), where N is the number of series and T is the number of observations. (1) For use with all components retention methods apart from user-specified ("fsmethod = user"). (2) If setting "mfmethod = user", you may specify the maximum number of components using "rmax = ". (3) Schwert's rule sets the maximum number of components using the rule: let $$L(k) = k(T/100)^{1/4}$$ for $k \in \{2, 4, 6, 8, 10, 12\}$ and let $T^* = \min(N, T)$; then the default maximum lag is given by $$L^* = \max_k \{L(k) \mid L(k) < T^*\}$$
rmax = *arg* *(default* = all*)*	User-specified maximum number of factors to retain (for use when "mfmethod = user").
fsic = *arg (default* = avg*)*	Factor selection criterion (when "fsmethod = bn"): "icp1" (ICP1), "icp2" (ICP2), "icp3" (ICP3), "pcp1" (PCP1), "pcp2" (PCP1), "pcp3" (ICP3), "avg" (average of all criteria ICP1 through PCP3). Factor selection criterion (when "fsmethod = ah"): "er" (eigenvalue ratio), "gr" (growth ratio), "avg" (average of eigenvalue ratio and growth ratio). Factor selection criterion (when "fsmethod = simple"): "min" (minimum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "max" (maximum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "avg" (average the optimal number of factors as specified by the min and max rule, then round to the nearest integer).

demeantime	Demeans observations across time prior to component selection procedures, when "n = bn" or "n = ah".
sdizetime	Standardizes observations across time prior to component selection procedures, when "n = bn" or "n = ah".
demeancross	Demeans observations across cross-sections prior to component selection procedures, when "n = bn" or "n = ah".
sdizecross	Standardizes observations across cross-sections prior to component selection procedures, when "n = bn" or "n = ah".

Initial Communalities Options

priors = *arg*	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").
priorfrac = *number*	User-specified common fraction (between 0 and 1) to be used when "priors = frac".
priorunique = *arg*	Vector of initial *uniqueness* 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

cov = *arg* *(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.
wgt = *name (optional)*	Name of series containing weights.

wgtmethod = *arg (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.
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.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 60. "Factor Analysis," on page 1485 of *User's Guide II* for a general discussion of factor analysis. The various estimation methods are described in "Estimation Methods" on page 1521 of *User's Guide II*.

See also `Factor::gls` (p. 283), `Factor::ipf` (p. 288), `Factor::ml` (p. 297), `Factor::pace` (p. 304), `Factor::pf` (p. 309).

Geomap

Geomap object. Specialized object to display shapefile map data.

Geomap Declaration

geomap .....declare Geomap object (p. 345).

Geomap Views

attr .....table of shapefile area attributes (p. 338).
label .....label view (p. 346).
display .....display default geomap view (p. 341).
map .....displays the map view of a geomap object (p. 349).

Geomap Procs

addtext.....place text in geomap (p. 331).
addtitle .....place an individual title in geomap (p. 335).
autocrop .....calculates and resets the minimum viewable size of a geomap (p. 339).
clearhist.....clear the contents of the history attribute (p. 339).
clearremarks .....clear the contents of the remarks attribute (p. 340).
copy.....creates a copy of the geomap (p. 340).
displayname.....set display name (p. 341).
formatshapelabel...adjusts the appearance and location of one or more shape labels (p. 342).
hide .....hides the specified area(s) in the geomap (p. 345).
legend.....set legend specific options (p. 346).
link.....link an attribute in the geomap to a series in the workfile (p. 347).
load .....load a shapefile from disk (p. 348).
makeattrser.....make alpha series containing linked attribute values (p. 348).
mask.....make invisible specified areas in the geomap (p. 350).
movetitle.....moves the title in the geomap to the specified location (p. 350).
olepush.....push updates to OLE linked objects in open applications (p. 351).
options.....change display options settings for the geomap (p. 352).
save .....save default view to a graphics file (p. 353).
setattr .....set the vaue of an object attribute (p. 355).
setfillcolor.....define the fill (background) color used in geomap shapes using values in a series (p. 356).
setfont.....set the font for the geomap text (p. 362).
setjust .....set the horizontal justification for multi-line area labels (p. 363).

setshapelabel ..... set which attribute to use or create a custom label to use when labeling shapes (p. 363).
show ..... shows the specified area(s) in the geomap (p. 364).
unmask ..... make visible specified areas in the geomap (p. 365).

Geomap Data Members

Scalar Values

@minx ..... minimum horizontal position of the geomap.
@maxx ..... maximum horizontal position of the geomap.
@miny ..... minimum vertical position of the geomap.
@maxy ..... maximum vertical position of the geomap.
@count ..... number of areas in the geomap.
@id("attr","val") .. the id number of the area which has the matching attribute value for the specified attribute name.

String values

@attr("*arg*") ..... string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@description ..... string containing the geomap object's description (if available).
@detailedtype ..... string with the object type: "GEOMAP".
@displayname ..... string containing the geomap object's display name. If the geomap has no display name set, the name is returned.
@ids("attr", "val") .....space delimited string containing the ID numbers of all the areas which has the matching attribute value for the specified attribute name.
@name ..... string containing the geomap object's name.
@remarks ..... string containing the geomap object's remarks (if available).
@type ..... string with the object type: "GEOMAP".
@updatetime ..... string representation of the time and date at which the geomap was last updated.

Geomap Entries

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

addtext	Geomap Procs

Place text in geomaps.

When adding text in one of the four predefined positions (left, right, top, bottom), EViews deletes any existing text that is in that position before adding the new text. Use the **keep** option to preserve the existing text.

Syntax

geomap_name.**addtext***(options)* "*text*"

Follow the `addtext` keyword with the *text* to be placed in the geomap, 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 geomap.*

font*([face], [pt], [+/- b], [+/- i], [+/- u], [+/- s])*	Set characteristics of text font. The font name *(face)*, size *(pt)*, and characteristics are all optional. *face* should be a valid font name, enclosed in double quotes. *pt* 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.
textcolor(*arg*)	Sets the color of the text. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 334.
fillcolor(*arg*)	Sets the background fill color of the text box. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 334.
framecolor(*arg*)	Sets the color of the text box frame. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 334.
keep	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, ac	Top (above and centered over the geomap).
l	Left rotated.
r	Right rotated.
b, bc	Below and centered over the geomap.
bl	Below and left side of the geomap.
br	Below and right side of the geomap.
al	Above and left side of the geomap.
ar	Above and right side of the geomap.
ibl	Inside near the bottom left corner of the geomap.
ibr	Inside near the bottom right corner of the geomap.
itl	Inside near the top left corner of the geomap.
itr	Inside near the top right corner of the geomap.
just(*arg*)	Set the justification of the text, where *arg* may be: “c” (center), “l” (left - default), “r” (right).
x, lb	Enclose text in a large box.
sb	Enclose text in a small 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 geomap, 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 geomaps 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 geomap. 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 geomap, the text is applied to the whole geomap, not to each individual geomap.

Color definitions

color_arg specifies the color to be employed in the arguments above. The color may be specified using predefined color names, by specifying the individual red-green-blue (RGB) components using the special "@RGB" function, or by specifying the individual red-green-blue (RGB) components in hexadecimal using the special "@HEX" function.

The predefined colors are given by the keywords (with their RGB and HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
ltred	@rgb(255, 168, 168)	@hex(ffa8a8)
green	@rgb(0, 128, 0)	@hex(008000)
black	@rgb(0, 0, 0)	@hex(000000)
white	@rgb(255, 255, 255)	@hex(ffffff)
purple	@rgb(128, 0, 128)	@hex(800080)
orange	@rgb(255, 128, 0)	@hex(ff8000)
yellow	@rgb(255, 255, 0)	@hex(ffff00)
gray	@rgb(128, 128, 128)	@hex(808080)
ltgray	@rgb(192, 192, 192)	@hex(c0c0c0)

Examples

```
geomap1.addtext(ctm) "Fig 1: California Population"
```

places the text "Fig1: California Population" centered above of geomap1.

```
geomap1.addtext(400, 350, rap, X) "Population surge"
```

places the text "Population surge" in a box within geomap1, where the text is center above the point located at (400, 350).

```
geomap1.addtext(c, ltm, textcolor(@rgb(green)), fillcolor(grey),
    framecolor(black)) "Great Lake\nWater Levels"
```

adds the text "Great Lakes" with "Water Levels" just below on a second line at the top left corner of the map. The text is colored green and is enclosed in a grey box with a large black frame.

Cross-references

See "Geomaps" on page 719 of *User's Guide I* for a discussion of geomaps. See "Value-Based Text and Fill Coloring" on page 186 of *User's Guide I* for discussion of color settings.

See also `Geomap::setfillcolor` (p. 356).

addtitle	Geomap Procs

Place an individual title in geomaps.

A geomap is capable of storing one title. When adding a title to a geomap that already has a title, the existing title will be deleted.

Syntax

geomap_name.**addtitle**(*options*) "*text*"

Follow the `addtitle` keyword with the text to be placed in the geomap, 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 `addtitle` 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 geomap.*

font*([face], [pt], [+/- **b**], [+/- **i**], [+/- **u**], [+/- **s**])*	Set characteristics of text font. The font name *(face)*, size *(pt)*, and characteristics are all optional. *face* should be a valid font name, enclosed in double quotes. *pt* 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.
textcolor(*arg*)	Sets the color of the text. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 334.
fillcolor(*arg*)	Sets the background fill color of the text box. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 334.
framecolor(*arg*)	Sets the color of the text box frame. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 334.

The following options control the position of the text:

ctm	Center top map.
ltm	Left top map.
rtm	Right top map.
cbm	Center bottom map.
lbm	Left bottom map.
rbm	Right bottom map.
lmm	Left middle map.
rmm	Right middle map.
lmp	Left-middle of point.
rmp	Right-middle of point.
lap	Left above point.
rap	Right above point.
lbp	Left below point.
rbp	Right below point.
cap	Center above point.
cbp	Center below point.
just(*arg*)	Set the justification of the text, where *arg* may be “c” (center), “l” (left - default), “r” (right).
x, lb	Enclose text in a large 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 geomap, 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 map units. Use the geomap data members @minx, @miny, @maxx, and @maxy to obtain the position limits.

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.

Color definitions

color_arg specifies the color to be employed in the arguments above. The color may be specified using predefined color names, by specifying the individual red-green-blue (RGB) components using the special “@RGB” function, or by specifying the individual red-green-blue (RGB) components in hexadecimal using the special “@HEX” function.

The predefined colors are given by the keywords (with their RGB and HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
ltred	@rgb(255, 168, 168)	@hex(ffa8a8)
green	@rgb(0, 128, 0)	@hex(008000)
black	@rgb(0, 0, 0)	@hex(000000)
white	@rgb(255, 255, 255)	@hex(ffffff)
purple	@rgb(128, 0, 128)	@hex(800080)
orange	@rgb(255, 128, 0)	@hex(ff8000)
yellow	@rgb(255, 255, 0)	@hex(ffff00)
gray	@rgb(128, 128, 128)	@hex(808080)
ltgray	@rgb(192, 192, 192)	@hex(c0c0c0)

Examples

```
geomap1.addtitle(ctm) "Fig 1: United States Population"
```

places the text "Fig 1: United States Population" centered above of GEOMAP1.

```
geomap1.addtitle(400, 350, rap, X) "Population surge"
```

places the text “Population surge” in a box within GEOMAP1, where the text is center above the point located at (400, 350).

```
geomap1.addtitle(c, ltm, textcolor(green), fillcolor(gray),
    framecolor(black)) "Great Lake\nWater Levels"
```

adds the text “Great Lakes” with “Water Levels” just below on a second line at the top left corner of the map. The text is colored green and is enclosed in a gray box with a large black frame.

Cross-references

See “Geomaps” on page 719 of *User’s Guide I* for a discussion of geomaps. See “Value-Based Text and Fill Coloring” on page 186 of *User’s Guide I* for a discussion of color settings.

See also `Geomap::setfillcolor` (p. 356).

attr	Geomap Views

Display table view of the shapefile attributes for each of the areas defined in the geomap.

Syntax

geomap_name.**attr**

Options

p	Print view.

Examples

```
gmap1.attr(p)
```

displays and prints the default `attr` view.

Cross-references

See "Geomaps" on page 719 of *User's Guide I* for a discussion of geomaps.

autocrop

Geomap Procs

Calculates and resets the minimum viewable size of a geomap.

When adding, moving, or removing objects such as text or shapes from a geomap, the minimum viewable size of a geomap may change. This may cause distortions in the whitespace surrounding the map with respect to the on-screen scrollbar or external output like file saving. Autocrop will readjust the size of the geomap to make the whitespace proportional.

Syntax

geomap_name.**autocrop**

Examples

```
geomap1.autocrop
```

resizes and recrops geomap1.

clearhist

Geomap Procs

Clear the contents of the history attribute.

Removes the geomap's history attribute, as shown in the label view of the scalar.

Syntax

geomap_name.**clearhist**

Examples

```
g1.clearhist
g1.label
```

The first line removes the history from the geomap G1, and the second line displays the label view of G1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Geomap::label` (p. 346).

clearremarks	Geomap Procs

Clear the contents of the remarks attribute.

Removes the scalar's remarks attribute, as shown in the label view of the scalar.

Syntax

scalar_name.**clearremarks**

Examples

```
s1.clearremarks
s1.label
```

The first line removes the remarks from the scalar S1, and the second line displays the label view of S1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Geomap::label` (p. 346).

copy	Geomap Procs

Creates a copy of the geomap.

Creates either a named or unnamed copy of the geomap.

Syntax

geomap_name.**copy**

geomap_name.**copy** *dest_name*

Examples

```
g1.copy
```

creates an unnamed copy of the geomap G1.

```
g1.copy g2
```

creates G2, a copy of the geomap G1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

display	Geomap Views

Display the default map view of the geomap.

Syntax

geomap_name.**display**

Examples

```
gmap1.display(p)
```

displays and prints the default map view.

Options

a	All observations (ignore sample)
p	Print.

Cross-references

See "Geomaps" on page 719 of *User's Guide I* for a discussion of geomaps.

displayname	Geomap Procs

Display name for a geomap object.

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

Syntax

geomap_name.**displayname** *display_name*

Display names are case-sensitive, and may contain various characters, such as spaces, that are not allowed in geomap object names.

Examples

```
g1.displayname My Geomap
g1.label
```

The first line attaches a display name “My Geomap” to the geomap object G1, and the second line displays the label view of G1, including its display name.

Cross-references

See “Labeling Objects” on page 123 of *User’s Guide I* for a discussion of labels and display names. See also `Geomap::label` (p. 346).

formatshapelabel	Geomap Procs

Adjusts the appearance and location of one or more shape labels.

Syntax

geomap_name.**formatshapelabel**(*options*) [*shapes*]

where *options* is the set of the label characteristics to be changed, *shapes* is the list of shapes whose labels are to be modified.

geomap_name.**formatshapelabel** (*shape_attribute*, *label_attributes*) *attr_value*

where *shape_attribute* is the name of the shape attribute used to match *attr_value*.

Options

font*([face], [pt], [+/- **b**], [+/- **i**], [+/- **u**], [+/- **s**])*	Set characteristics of text font. The font name *(face)*, size *(pt)*, and characteristics are all optional. *face* should be a valid font name, enclosed in double quotes. *pt* 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.
textcolor(*arg*)	Sets the color of the text. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 334.

fillcolor(*arg*)	Sets the background fill color of the text box. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 334.
framecolor(*arg*)	Sets the color of the text box frame. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 334.
keep	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.

Examples

```
geomap1.formatshapelabel(x) 2 5 10
```

will enclose the label in a large box for the second, fifth, and tenth shape in the GEOMAP1 geomap object.

```
geomap1.formatshapelabel(attr=statename, textcolor(red),
    pos(100,100)) nevada "south dakota"
```

will set the text color to red and move the label to (100, 100) for the shapes in GEOMAP1 whose *state* attribute matches *nevada* and *south dakota*.

geomap	Geomap Declaration

Declare a geomap object.

Syntax

geomap *geomap_name*

Declare a geomap object that is capable of reading Esri (www.esri.com) shapefiles. Geomaps are useful for visualizing data that are tied to specific geographic regions.

Examples

```
geomap gmap
```

Cross-references

See "Geomaps" on page 719 of *User's Guide I* for a discussion of geomaps.

hide	Geomap Procs

Hides the specified area(s) in the geomap.

Syntax

geomap_name.**hide** *area_list*

where *area_list* is a list of areas that will be made hidden in addition to what is already hidden. *area_list* may be a list of area IDs or "all." The area ID for each area is listed in the attributes view of the geomap and corresponds to the EViews ID column.

Examples

```
geomap01.hide 1 4 6
```

hides areas numbered 1, 4, and 6 in the geomap object GEOMAP1.

```
geomap01.hide all
```

hides all the areas in the geomap object GEOMAP1.

Cross-references

See also `Geomap::show` (p. 364).

label	Geomap Views

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

Syntax

geomap_name.**label**

geomap_name.**label***(options) text*

Options

To modify the label, you should specify one of the following options along with optional text. If there is no text provided, the specified field will be cleared:

c	Clears all text fields in the label.
d	Sets the description field to *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of the geomap GMAP with "Annual rainfall for region":

```
gmap.label(r)
gmap.label(r) Annual rainfall in region
```

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels.

legend	Geomap Procs

Set legend specific options in geomaps.

Syntax

geomap_name.**legend***(options) legend_arg*

where *legend_arg* may include:

*[+/-]***display**	Make the legend visible or invisible

Options

vertical	Set legend orientation to be vertical

Examples

```
gmap1.legend display
```

displays the legend horizontally.

```
gmap1.legend(vertical) +display
```

displays the legend vertically.

```
gmap1.legend -display
```

hides the legend.

Cross-references

See "Geomaps" on page 719 of *User's Guide I* for a discussion of geomaps.

link

Geomap Procs

Link an attribute in the geomap object to a series in the workfile.

Syntax

geomap_name.**link** *attribute_name series_name*

associates individual shapes in the geomap with observations in the workfile by matching attribute values for shapes (defined in the loaded shapefile), with the values of the series.

If there are multiple matching series observations for a given attribute value, the first matching series observation value will be used.

Examples

```
gmap1.link fips statesfips
```

links the values of the FIPS attribute of each area to the first matching observation in the STATEFIPS series.

Cross-references

See "Geomaps" on page 719 and of *User's Guide I* for a discussion of geomaps. See "Value-Based Text and Fill Coloring" on page 186 of *User's Guide I* for discussion of color settings.

See also Geomap::setfillcolor (p. 356).

load	Geomap Procs

Load a shapefile from disk.

Syntax

geomap_name.**load** *file_description*

where *file_description* is a local file found on disk.

Examples

```
gmap1.load c:\temp\california.prj
```

loads the "c:\temp\california.prj" shapefile. Note the corresponding ".dbf", ".shp", and ".shx" files must also be present in the same directory (for this example, "California.dbf", "California.shp", and "California.shx").

Cross-references

See "Geomaps" on page 719 and of *User's Guide I* for a discussion of geomaps.

makeattrser	Geomap Procs

Create series containing linked attribute values from the shapefile.

Saves the values of the specified attributes linked to each observation in the workfile in series.

Syntax

geomap_name.**makeattrser** *attr_name [series_name]*

geomap_name.**makeattrser***(options) attr_name1 series_name1 [attr_name2 series_name2 …]*

geomap_name.**makeattrser***(options)* **@all** *[name_pattern]*

You must first provide *attr_name,* the name of an attribute that exists in the shapefile. You may optionally provide a name for the series in the workfile to hold the linked values for the attribute. By default, EViews will use the attribute name for the series, or will construct a valid EViews name from the series.

If you wish to save linked values for more than one attribute in the shapefile, you should enter the names in pairs. You may optionally provide a name for a group to contain the series.

You may instruct EViews to save all of the attributes by using the keyword "@all". EViews will attempt to use the attribute name for the series, unless you provide a wildcard pattern, in which case, EViews will construct the name from the pattern and the attribute names.

Options

n = *arg*	(optional) Name of group to contain the saved attribute series.

Examples

```
gmap1.makeattrser(n=grp1) @all
```

will create attribute series corresponding to all of the attribute types in the shapefile, using the names of the attributes for the series names.

```
gmap1.makeattrser rainfall rfall
```

will create an attribute series RFALL containing the linked values of the "rainfall" attribute.

```
gmap1.makeattrser ranfall rfall population pop acreage size
```

will create the series RFALL, POP, and SIZE in the workfile, and will fill these series with the "rainfall", "population", and "acreage" attributes from the shapefile.

Cross-references

See "Geomaps" on page 719 of *User's Guide I* for a discussion of geomaps.

map

Geomap Views

Displays the map view of a geomap object.

Syntax

geomap_name.**map**(*options*)

Options

p	Print the map view.

Examples

```
geomap1.map(p)
```

displays and prints map view of the geomap GEOMAP1.

mask	Geomap Procs

Make specified areas in the geomap invisible.

Syntax

geomap_name.**mask** *area_list*

where *area_list* is a space delimited integer list of the area IDs. To obtain the ID number of an area, see the **@id** geomap data member.

The keyword "@unlinked" may be used to specify all shapes that are not linked to observations in the workfile.

Examples

```
gmap1.mask 2 4 10 20
```

turns off display for areas 2, 4, 10, and 20 in the geomap object GMAP1.

```
gmap1.mask @unlinked
```

turns off display for all unlinked shapes from the shapefile.

Cross-references

See "Geomaps" on page 719 of *User's Guide I* for a discussion of geomaps.

See also `Geomap::unmask` (p. 365).

movetitle	Geomap Procs

Moves the title in the geomap to the specified location.

Syntax

geomap_name.**movetitle**(*position*)

where *position* can be one of the following:

ctm	Center top map.
ltm	Left top map.
rtm	Right top map.
cbm	Center bottom map.
lbm	Left bottom map.
rbm	Right bottom map.

lmm	Left middle map.
rmm	Right middle map.
lmp	Left-middle of point.
rmp	Right-middle of point.
lap	Left above point.
rap	Right above point.
lbp	Left below point.
rbp	Right below point.
cap	Center above point.
cbp	Center below point.

You can also 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 map units. Use the geomap data members @minx, @miny, @maxx, and @maxy to obtain the position limits.

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.

Examples

```
geomap01.movetitle(lbm)
```

moves the current title of the geomap object GEOMAP1 to the left bottom corner.

```
geomap01.movetitle(400, 400, rmp)
```

moves the title of the geomap object GEOMAP1 to the right of the point (400, 400) in the map.

Cross-references

See "Geomaps" on page 719 of *User's Guide I* for a discussion of geomaps.

olepush	Geomap Procs

Push updates to OLE linked objects in open applications.

Syntax

geomap_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

options	Geomap Procs

Set display options for a geomap object including legend and outline color.

Syntax

geomap_name.**options** *options_list*

Options

legend / -legend	Turn on and off legend.
lineclr(*color_arg*)	Specifies the boundary line color of the areas using the *color_arg* color specification.

color_arg specifies the color to be employed. The color may be specified using predefined color names, by specifying the individual red-green-blue (RGB) components using the special "@RGB" function, or by specifying the individual red-green-blue (RGB) components in hexadecimal using the special "@HEX" function.

The predefined colors are given by the keywords (with their RGB equivalents):

blue	@rgb(0, 0, 255)
red	@rgb(255, 0, 0)
ltred	@rgb(255, 168, 168)
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

```
gmap1.options -legend
```

hides the legend in the GEOMAP1 object.

```
gmap1.options legend lineclr(black)
```

displays the legend and sets the area boundary line color to black.

Cross-references

See "Geomaps" on page 719 of *User's Guide I* for a discussion of geomaps.

See `Geomap::setfillcolor` (p. 356) for setting the shape fill color.

save	Geomap Procs

Save the default view of a geomap object to disk as a Windows metafile (.EMF or .WMF), PostScript (.EPS), bitmap (.BMP), Graphics Interchange Format (.GIF), Joint Photographic Experts Exchange (.JPEG), Portable Network Graphics (.PNG), Portable Document Format (.PDF), LaTeX (.TEX), or Markdown (.MD).

Syntax

geomap_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 geomap may be saved with an EMF, WMF, EPS, BMP, GIF, JPG, PNG, PDF, or MD extension. The MD (Markdown) setting uses very basic syntax and should be usable in most editors.

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 = *file_type*	Specifies the file type, where *file_type* 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"), Portable Network Graphics ("png"), Portable Document Format ("pdf"), LaTeX ("tex"), or Markdown ("md"). Files will be saved with the ".emf", ".wmf", ".eps", ".bmp", ".gif", ".jpeg", ".png", ".pdf", ".tex", or ".md" extensions, respectively.
u = *units*	Specify units of measurement, where *units* 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 = *width*	Set width of the graphic in the selected units.
h = *height*	Set height of the graphic in the selected units.
c / -c	[Save / Do not save] the geomap in color.
trans / -trans	[Set / Do not set] background to transparent (for graph formats which support transparency).
d = *dpi*	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 geomap 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 geomap 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 Options

box / -box	[Save / Do not save] the geomap with a bounding box. The bounding box is an invisible rectangle placed around the graphic to indicate its boundaries. The default is taken from the global graph export settings.
land	Save the geomap in landscape orientation. The default uses portrait mode.
prompt	Force the dialog to appear from within a program.

LaTeX Options

texspec / -texspec	[Include / Do not include] the full LaTeX documentation specification in the LaTeX output. The default behavior is taken from the global default settings.

Examples

```
gmap1.save(t=ps, -box, land) c:\data\MyGra1
```

saves GMAP1 as a PostScript file “MyGra1.EPS”. The geomap is saved in landscape orientation without a bounding box.

```
gmap2.save(t=emf, u=pts, w=300, h=300) MyGra2
```

saves GMAP2 in the default directory as an Enhanced Windows metafile “MyGra2.EMF”. The image will be scaled to 300×300 points.

```
gmap3.save(t=png, u=in, w=5, d=300) MyGra3
```

saves GMAP3 in the default directory as a PNG file “MyGra3.PNG”. The image will be 5 inches wide at 300 dpi.

Cross-references

See “Geomaps” on page 719 of *User’s Guide I* for a discussion of geomaps.

setattr	Geomap Procs

Set the object attribute.

Syntax

geomap_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the “revised” attribute in the object A to the string “never”, and extracts the attribute into the string object S.

Cross-references

See “Adding Custom Attributes in the Label View” on page 123 and “Adding Your Own Label Attributes” on page 70 of *User’s Guide I*.

setfillcolor	Geomap Procs

Define the fill (background) color used in geomap shapes using values in a series.

Syntax

geomap_name.**setfillcolor**(**t** = *type*) *fillcolor_args*

where:

type = *arg*	Type of fill coloring for spreadsheet cells: “single” (single color), “posneg” (positive-negative single threshold), “range” (single range coloring), “hilo” (high-low-median), “custom” (custom coloring).

General Arguments

To specify the series or expression whose values will determine the background color:

- **mapser**(*spec*)

where *spec* is a series name or expression.

To specify the minimum and maximum values where the coloring begins and ends:

- **min**(*color_arg*)
- **max**(*color_arg*)

To set the missing value (NA) background color:

- **naclr**(*color_arg*)

where *color_arg* is described below in “Color definitions” on page 359. If omitted, the color defaults to “white”.

Type-specific Arguments

There are optional type-specific arguments that correspond to each of the type choices:

Single color

To set the single background color:

clr(*color_arg*)

where *color_arg* is described below in "Color definitions" on page 359. If omitted, the color defaults to "white".

Positive-negative single threshold

You may set the color for both the non-negative (**posclr**) and the negative (**negclr**) values

posclr(*color_arg*)
negclr(*color_arg*)

where *color_arg* is described below in "Color definitions" on page 359. If omitted, the non-negative color defaults to "white" and the negative color defaults to light-red.

Single range

To specify the range, you must specify the range endpoints:

range(*lower_val, upper_val[, range_def)*

where *range_def* specifies the range endpoints:

cright	closed on the right only
cboth	closed on both sides
cleft	closed on the left only
oboth	open on both sides

By default, the range will be open on the lower and closed on the upper threshold limits.

You should provide a color specification for the inside range color (**inclr**) and outside range color (**outclr**):

inclr(*color_arg*)
outclr(*color_arg*)

where *color_arg* is described below in "Color definitions" on page 359. If omitted, the interior color defaults to light-red, and the exterior defaults to white.

High-Low-Median

When "type = hilo" you may specify the high, low, and median coloring values:

highclr(*color_arg*)
lowclr(*color_arg*)

medianclr(*color_arg*)

where *color_arg* is described below in "Color definitions" on page 359. If omitted, the colors default to light-red.

Custom

When "type = custom" you may specify custom coloring options.

You may optionally set a base background color, and then add one or more custom threshold or range color specifications. Multiple threshold and range specifications will layer, with the first applied first, followed by the second, and so on.

Custom Base Color

To set the base color (optional):

clr(*color_arg*)

as described below in "Color definitions" on page 359. If omitted, the color defaults to "white".

Custom Threshold

To add a threshold specification:

thresh(**limit***(threshold_value, threshold_spec)*, **lowclr***(below_arg)*, **highclr***(above_arg)*, *threshold_name]*)

where *threshold_spec* is one of

cright	closed on the right
cleft	closed on the left

and the *below_arg* and *above_arg* are one of

color_arg	solid color specification
@grad(*color_arg*)	gradient using color specification
@trans	transparent

and *color_arg* are as described below in "Color definitions" on page 359. If omitted, the color defaults to "white".

The optional *threshold_name* argument may be used to attach a name to the corresponding definition.

Custom Range

To add a range specification:

range(**limit***(low_value, high_value, range_spec)*, **inclr***(inside_arg)*, **outclr***(outside_arg)[*, *range_name]*)

where *range_spec* is one of

cright	closed on the right only
cboth	closed on both sides
cleft	closed on the left only
oboth	open on both sides

inside_arg is one of

color_arg	solid color specification
@grad(*color_arg1*, *color_arg2*)	gradient using color specification, where *color_arg1 and color_arg2* are the low and high colors, res*pectively*.
@trans	transparent

outside_arg is one of

color_arg	solid color specification
@grad(*color_arg*)	gradient using color specification
@trans	transparent

color_arg1 and *color_arg2* are as described below in "Color definitions" on page 359.

The optional *range_name* argument may be used to attach a name to the corresponding definition.

Color definitions

color_arg specifies the color to be employed in the arguments above. The color may be specified using predefined color names, by specifying the individual red-green-blue (RGB) components using the special "@RGB" function, or by specifying the individual red-green-blue (RGB) components in hexadecimal using the special "@HEX" function.

The predefined colors are given by the keywords (with their RGB and HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
ltred	@rgb(255, 168, 168)	@hex(ffa8a8)
green	@rgb(0, 128, 0)	@hex(008000)
black	@rgb(0, 0, 0)	@hex(000000)
white	@rgb(255, 255, 255)	@hex(ffffff)
purple	@rgb(128, 0, 128)	@hex(800080)

orange	@rgb(255, 128, 0)	@hex(ff8000)
yellow	@rgb(255, 255, 0)	@hex(ffff00)
gray	@rgb(128, 128, 128)	@hex(808080)
ltgray	@rgb(192, 192, 192)	@hex(c0c0c0)

Examples

To set a gray fill color for the shapes, you may use:

```
gmap.setfillcolor(type=single) clr(gray)
```

To set a fill color for negative values, you may use

```
gmap.setfillcolor(type=posneg) mapser(ser1)
```

which sets the fill color to white for non-negative values and light red for negative values of SER1.

Similarly,

```
gmap.setfillcolor(type=posneg) mapser(ser1) posclr(@rgb(10, 20,
    30)) negclr(purple)
```

sets the background sheet fill color to @rgb(10, 20, 30) for non-negative values and purple for negative values of SER1.

Range coloring may be specified using the "type = range" option. The command

```
gmap.setfillcolor(type=range) mapser(ser1) clr(ltgray) range(10,
    20, cleft) inclr(@rgb(128, 0, 128)) outclr(ltred) naclr(green)
```

sets the background fill to @rgb(128, 0, 128) for values between 10 and 20, light-red to values outside of the range 10 to 20, and green, for missing values.

Custom coloring allows you to construct more complex background filling:

```
gmap.setfillcolor(type=custom) mapser(ser1) clr(@rgb(10, 0, 0))
    range(limit(-10, 10, oboth), inclr(green), outclr(white)))
    thresh(limit(-1, oleft), highclr(grey), lowclr(@trans))
```

Cross-references

See "Geomaps" on page 719 and of *User's Guide I* for a discussion of geomaps. See "Value-Based Text and Fill Coloring" on page 186 of *User's Guide I* for discussion of color settings.

See also `Geomap::options` (p. 352) for options to control the shape border color and legend.

options	Geomap Procs

Set display options for a geomap object including legend and outline color.

Syntax

geomap_name.**options** *options_list*

Options

legend / -legend	Turn on and off legend.
lineclr(*color_arg*)	Specifies the boundary line color of the areas using the *color_arg* color specification.

color_arg specifies the color to be employed. The color may be specified using predefined color names, by specifying the individual red-green-blue (RGB) components using the special "@RGB" function, or by specifying the individual red-green-blue (RGB) components in hexadecimal using the special "@HEX" function.

The predefined colors are given by the keywords (with their RGB equivalents):

blue	@rgb(0, 0, 255)
red	@rgb(255, 0, 0)
ltred	@rgb(255, 168, 168)
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

```
gmap1.options -legend
```

hides the legend in the GEOMAP1 object.

```
gmap1.options legend lineclr(black)
```

displays the legend and sets the area boundary line color to black.

Cross-references

See "Geomaps" on page 719 of *User's Guide I* for a discussion of geomaps.

See `Geomap::setfillcolor` (p. 356) for setting the shape fill color.

setfont	Geomap Procs

Set the font for area labels in the geomap.

Syntax

table_name.**setfont** *font_args*

The *font_args* may include one or more of the following:

*type([face], [pt], [+/- **b**], [+/- **i**], [+/- **u**], [+/- **s**])*	Set characteristics of the font for the graph element *type*. The font name *(face)*, size *(pt)*, and characteristics are all optional. *face* should be a valid font name, enclosed in double quotes. *pt* 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.

and *type* is one of "all", "axes", "legend", "text", "obs", where "axes" refers to the axes labels, "legend" refers to the graph legend, "text" refers to the text objects, "obs" refers to the observation scale, and "all" refers to all of the elements.

Examples

```
gmap1.setfont "Times New Roman" +i
```

sets the font to Times New Roman Italic.

```
gmap1.setfont 8pt
```

changes all of text in the region to 8 point.

```
gmap1.setfont +b -i
```

removes the italic, and adds boldface to the area labels.

The commands:

```
gmap1.setfont -s +u 14pt
gmap1.setfont "Batang" 14pt +u
```

he first command changes the point size to 14, removes strike-through and adds underscoring. The second changes the typeface to Batang, and adds underscoring,

Cross-references

See "Geomaps" on page 719 of *User's Guide I* for a discussion of geomaps.

See `Geomap::setfillcolor` (p. 356) for details on changing fill color, and `Geomap::options` (p. 352) for information on changing outline colors.

setjust	Geomap Procs

Set the horizontal justification for multi-line area labels.

Syntax

geomap_name.**setjust** *format_arg*

where *format_arg* may be set to left, center, right, or auto (strings are left-justified and numbers are right-justified). Default display settings can be set in General Options; see "Spreadsheet Data Display" on page 1019 of *User's Guide I*.

Note that justification settings have no effect on single-line labels.

Examples

```
geo1.setjust left
```

left-justifies the lines in the area labels in the geomap GEO1.

Cross-references

See "Geomaps" on page 719 of *User's Guide I* for a discussion of geomaps.

setshapelabel	Geomap Procs

Set which attribute to use or create a custom label to use when labeling shapes in the geomap.

Syntax

geomap_name.**setshapelabel***(attribute_name) custom_label*

where *attribute_name* is the name of an attributed in the geomap. In the case where *attribute_name* is equal to "custom", the *custom_label* will be used. In all other cases, the command will be ignored.

Examples

```
geomap1.setshapelabel(name)
```

will label the areas in the GEOMAP1 geomap object using areas name attribute.

```
geomap1.setshapelabel (none)
```

turn off all area labels in the GEOMAP1 geomap object.

```
geomap1.setshapelabel (fillvalues)
```

will label the areas in the GEOMAP1 geomap object using the numerical values used for determining area fill color. This only applicable after a colormap has been applied. All values will otherwise be NA.

```
geomap1.Setshapelabel(custom)
    Area:[county],[state]\nPop:[fillvalues]
```

will create a custom 2 line label for the areas in the GEOMAP1 geomap object. The first line of the label will read "Area:" followed by the areas county attribute, a comma, and then the areas state. The first line will look similar to "Area:Suffolk,NY". The second line of the label will "Pop:" followed by the value used to color the area.Examples

Cross-references

See "Geomaps" on page 719 of *User's Guide I* for a discussion of geomaps.

show	Geomap Procs

Shows the specified area(s) in the geomap.

Syntax

geomap_name.**show** *area_list*

where *area_list* is a list of areas that will be shown in addition to what is already shown. *area_list* may be a list of area IDs or "all." The area ID for each area is listed in the attributes view of the geomap and corresponds to the EViews ID column.

Examples

```
geomap01.show 3 5 10
```

shows areas numbered 3, 5, and 10 in the geomap object GEOMAP1.

```
geomap01.show all
```

shows all the areas in the geomap object GEOMAP1.

Cross-references

See also `Geomap::hide` (p. 345).

unmask	Geomap Procs

Make visible specified areas in the geomap.

Syntax

geomap_name.**unmask** *area_list*

where *area_list* is a space delimited integer list of the area IDs. To obtain the ID number of an area, see the **@id** geomap data member.

The keyword "@unlinked" may be used to specify all shapes that are not linked to observations in the workfile.

Examples

```
gmap1.unmask 2 4 10 20
```

turns on display for areas 2, 4, 10, and 20 in the geomap object GMAP1.

```
gmap1.unmask @unlinked
```

turns on display for all unlinked shapes in the shapefile.

Cross-references

See "Geomaps" on page 719 of *User's Guide I* for a discussion of geomaps.

See also `Geomap::mask` (p. 350).

Graph

Graph object. Specialized object used to hold graphical output.

Graph Declaration

freeze ..... freeze graphical view of object (p. 457).
graph ..... create graph object using graph command or by merging existing graphs (p. 399).

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 Command Summary" on page 1267.

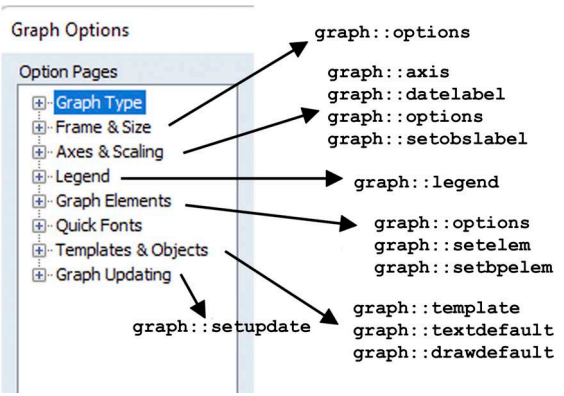
area ..... area graph (p. 1269).
band ..... area band graph (p. 1272).
bar ..... bar graph (p. 1275).
boxplot ..... boxplot graph (p. 1279).
bubble ..... bubble plot graph (p. 1281).
bubbletrip ..... bubble plot graph specified as triplets (p. 1282).
distplot ..... distribution graph (p. 1283).
dot ..... dot plot graph (p. 1290).
errbar ..... error bar graph (p. 1294).
hilo ..... high-low(-open-close) graph (p. 1296).
line ..... line-symbol graph (p. 1298).
mixed ..... mixed-type graph (p. 1301).
pie ..... pie chart (p. 1304).
qqplot ..... quantile-quantile graph (p. 1306).
scat ..... scatterplot (p. 1310).

Graph View

Graph Procs

setbpelem ..... set options for element of a boxplot graph (p. 416).
setelem..... set individual line, symbol, bar and legend options for each series in the graph (p. 417).
setfont..... set the font for the text in a graph (p. 423).
setobslabel ..... set custom axis labels for observation scale of a graph (p. 424).
setupdate..... set update options for the graph (p. 426).
sort ..... sort the series in a graph (p. 427).
template ..... use template graph (p. 428).
textdefault ..... set default settings for text objects in the graph (p. 429).
update..... update graph with data changes (p. 432).

The relationship between the elements of the graph dialog and the associated graph procs is illustrated below:



Graph Data Members

Scalar Values

@axismin(*axis*) ....returns the minimum value for the specified axis. Acceptable values for *axis* are “t”, “l”, “b”, “r”, for top, left, bottom, right.
@axismax(*axis*).... .....returns the maximum value for the specified axis. Acceptable values for *axis* are “t”, “l”, “b”, “r”, for top, left, bottom, right.
@axispos(*value*, *axis*) .....returns the location in virtual inches of the specified data value on the graph. *value* is in the same units as the specified axis. When specifying a date for *value*, the string must be quoted. Acceptable values for *axis* are “t”, “l”, “b”, “r” for top, left, bottom, right.

String Values

@attr("*arg*").....string containing the value of the *arg* attribute, where the argument is specified as a quoted string.

@description.....returns a string containing the object description (if available).

@detailedtype.....returns a string with the object type: "GRAPH".

@displayname.....returns a string containing the Graph's displayname. If the Graph has no display name set, the name is returned.

@members.....string containing a space delimited list of the names of the series contained in the Graph.

@name.....returns a string containing the Graph's name.

@remarks.....returns a string containing the Graph's remarks (if available).

@type.....returns a string with the object type: "GRAPH".

@updatetime.....returns a string representation of the time and date at which the Graph was last updated.

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.

addarrow	Graph Procs

Draw a line or arrow on a graph.

Syntax

graph_name.**addarrow** [**pos**(*x1*,*y1*,*x2*,*y2*) **axispos**(*x1*,*y1*,*x2*,*y2*,*x-axis*,*y-axis*) **axispos**(*x1*,*y1*,*x2*,*y2*,*y-axis*) **axispt**(*x2*,*y2*,*angle*,*length*,*x-axis*,*y-axis*)] **linewidth***(lwidth)* **arrowwidth***(awidth)* **color***(color)* **pattern**(*pattern*) **startsym**(*ssym*) **endsym**(*esym*) **label**(*str*) **labelpos**(*position*) **frame**(*size*) **indicator**

Follow the `addarrow` keyword a set of specifications determining the position and style of the line/arrow to be drawn.

The position and size of the arrow/line can be specified with one of the `pos`, `axispos` or `axispt` arguments.

The `pos` argument specifies coordinates of the line in virtual space. *x1* is the starting X (horizontal) coordinate, and *y1* is the starting Y (vertical) coordinate. Similarly *x2* and *y2* are the end point coordinates. Coordinates are set 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 *x1* number specifies how many virtual inches to offset to the right from the origin. The second number *y1* specifies how many virtual inches to offset below the origin. The start point of the line will be set at `the` specified coordinates.

The `axispos` argument specifies coordinates in units of the graph scale. *x1* is the starting X (horizontal) coordinate, and *y1* is the starting Y (vertical) coordinate. Similarly *x2* and *y2* are the end point coordinates.

For time-series graphs you must also specify which non-time based axis the y-coordinates's scale are based on, using l,t,r,b for left, top, right, bottom respectively. x-coordinates should be specified as a date/time.

For non-time series graphs you must specify the axis of scale of both x and y coordinates.

The `axispt` argument specifies the end point coordinates of the line, along with the angle and length of the line. Angles are measured in degrees, and length in virtual inches.

The `linewidth` argument specifies the thickness of the line. *lwidth* should be a number between ".25" and "5", indicating the width in points.

`Arrowwidth` determines the size of the arrow head on the line. *awidth* can be either "small", "medium" or "large".

`color` specifies the color of the line. The *color* value may set by using one of the color keywords (e.g., "blue"), by using the RGB values (e.g., "@RGB(255, 255, 0)"), or by specifying the components in hexadecimal (e.g., "@HEX(ff0000)").

The predefined colors are given by the keywords (with their RGB and HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
ltred	@rgb(255, 168, 168)	@hex(ffa8a8)
green	@rgb(0, 128, 0)	@hex(008000)
black	@rgb(0, 0, 0)	@hex(000000)
white	@rgb(255, 255, 255)	@hex(ffffff)
purple	@rgb(128, 0, 128)	@hex(800080)
orange	@rgb(255, 128, 0)	@hex(ff8000)
yellow	@rgb(255, 255, 0)	@hex(ffff00)
gray	@rgb(128, 128, 128)	@hex(808080)
ltgray	@rgb(192, 192, 192)	@hex(c0c0c0)

The `pattern` argument specifies the line pattern. `pattern` can take a numerical value, or one of the corresponding keywords:

(1) solid
(2) dash1
(3) dash2
(4) dash3
(5) dash4
(6) dash5
(7) dash6
(8) dash7
(9) dash8
(10) dash9
(11) dash10
(12) none

The `startsym` and `endsym` arguments define the arrowhead at the start or end of the line. You may specify "none", "filled", "outline", or "rangeline".

`label` adds a text label to the start point of the arrow. `labelpos` specifies the location of the text relative to the start point of the line. The following *position*s are available:

Vert	left or right of the start point depending on the angle of the line
Horz	left or right of the start point depending on the angle of the line
AR	above and right of the start point
AL	above and left of the start point
BR	below and right of the start point
BL	below and left of the start point
L	left of the start point
R	right of the start point
A	above the start point
B	below the start point

`Frame` encloses the text in a box. *Size* specifies whether the box should be a small box (*sb*) or a large box (*lb*).

`Indicator` places a red indicator within the text `frame`, indicating the start point location relative to the text. NOTE: The indicator will only appear if the label position (`labelpos`) is set to *AR*, *AL*, *BR*, or *BL*

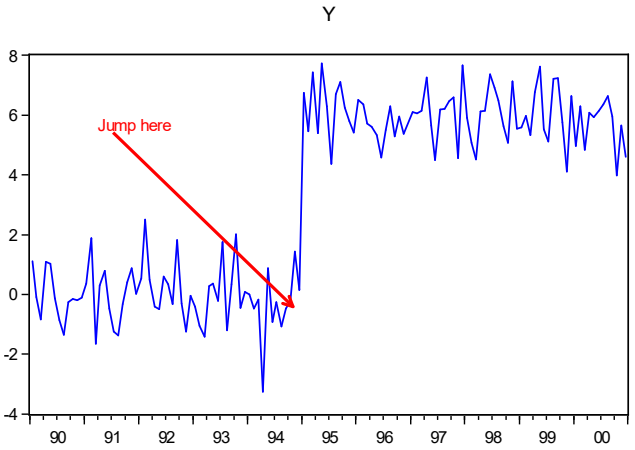
Examples

The commands

```
create m 1990 2000
smpl 1990 1995
series y=nrnd
smpl 1995 2000
y = 6+nrnd
smpl @all
freeze(gr) y.line

gr.addarrow pos(0.7,0.65, 2.2,2.1) color(red) arrowwidth(large)
    endsym(outline) linewidth(2) label(Jump here)
```

create a graph and draw an arrow at the specified positions:



The command

```
gr.addarrow axispos(94, 3, 97, 4.2, 1)
```

adds a second arrow starting at the point corresponding to the year 1994 on the x-axis and the y-axis value of 3, and ending at the year 1997 with a y-value of 4.2.

Cross-references

See "Adding lines and shading" on page 51 of *User's Guide I* for discussion.

See `Graph::addellipse` (p. 373), `Graph::addrect` (p. 376), `Graph::addtext` (p. 378), and `Graph::delete` (p. 392).

See also `Graph::legend` (p. 402) and `Graph::textdefault` (p. 429).

addellipse	Graph Procs

Draw an ellipse on a graph.

Syntax

graph_name.**addellipse** [**pos**(*x1*,*y1*,*x2*,*y2*) **axisctr**(*x1*,*y1*,*x-axis*,*y-axis*) **axispos**(*x1*,*y1*,*y-axis*)] **linewidth**(*lwidth*) **color**(*color*) **pattern**(*pattern*) **height**(*height*) **width**(*width*) **angle**(*angle*)

Follow the `addellipse` keyword a set of specifications determining the position and style of the ellipse to be drawn.

The position and size of the ellipse can be specified with either the `pos or axisctr` arguments.

The `pos` argument specifies coordinates of the center of the ellipse in virtual space. *x1* is the center point X (horizontal) coordinate, and *y1* is the center point Y (vertical) coordinate. Coordinates are set 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 *x1* number specifies how many virtual inches to offset to the right from the origin. The second number *y1* specifies how many virtual inches to offset below the origin.

The `axisctr` argument specifies coordinates in units of the graph scale. *x1* is the center point X (horizontal) coordinate, and *y1* is the center point Y (vertical) coordinate.

For time-series graphs you must also specify which non-time based axis the y-coordinates's scale are based on, using l,t,r,b for left, top, right, bottom respectively. x-coordinates should be specified as a date/time.

For non-time series graphs you must specify the axis of scale of both x and y coordinates.

The `height` argument specifies the height of the ellipse. Similarly the `width` argument specifies its width. `angle` controls the rotation of the ellipse (in degrees).

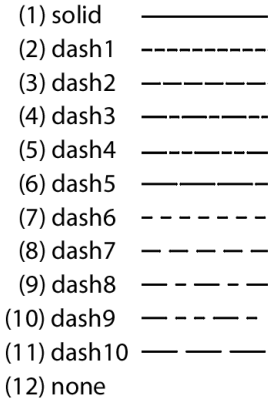
The `linewidth` argument specifies the thickness of the ellipse outline. *lwidth* should be a number between ".25" and "5", indicating the width in points.

`color` specifies the color of the ellipse outline. The *color* value may set by using one of the color keywords (e.g., "blue"), by using the RGB values (e.g., "@RGB(255, 255, 0)"), or by specifying the components in hexadecimal (e.g., "@HEX(ff0000)").

The predefined colors are given by the keywords (with their RGB and HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
ltred	@rgb(255, 168, 168)	@hex(ffa8a8)
green	@rgb(0, 128, 0)	@hex(008000)
black	@rgb(0, 0, 0)	@hex(000000)
white	@rgb(255, 255, 255)	@hex(ffffff)
purple	@rgb(128, 0, 128)	@hex(800080)
orange	@rgb(255, 128, 0)	@hex(ff8000)
yellow	@rgb(255, 255, 0)	@hex(ffff00)
gray	@rgb(128, 128, 128)	@hex(808080)
ltgray	@rgb(192, 192, 192)	@hex(c0c0c0)

The `pattern` argument specifies the ellipse outline pattern. `pattern` can take a numerical value, or one of the corresponding keywords:



Examples

The commands

```
create m 1990 2000
smpl 1990 1995
series y=nrnd
smpl 1995 2000
y = 6+nrnd
smpl @all
freeze(gr) y.line
gr.addellipse pos(1,1)  width(2) height(.7) angle(110) color(red)
    pattern(2) linewidth(3)
```

create a graph and adds a red ellipse that is centered 1 virtual inch from the top and 1 virtual inch from the left of the graph that is 2 virtual inches wide and 0.7 virtual inches tall. It uses a 3 pt dash1 line pattern. The ellipse is also rotated 110 degrees

The command

```
gr.addellipse axisctr(1995, @mean(x),1)  width(30) height(.2)
   angle(-50) color(blue)
```

adds to a blue ellipse that is centered at 1995 and the mean of x in left axis units. It is 30 observations wide and 0.2 left axis units tall. It is also rotated -50 degrees

Cross-references

See "Drawing Lines and Arrows" on page 907 of *User's Guide I* for discussion.

See Graph::addarrow (p. 370), Graph::addrect (p. 376), Graph::addtext (p. 378) and Graph::delete (p. 392).

See also Graph::legend (p. 402) and Graph::textdefault (p. 429).

addrect	Graph Procs

Draw a rectangle on a graph.

Syntax

graph_name.**addrect**[**pos**(*x1*,*y1*,*x2*,*y2*) **axisctr**(*x1*,*y1*,*x-axis*,*y-axis*) **axispos**(*x1*,*y1*,*y-axis*)] **linewidth***(lwidth)* **color***(color)* **pattern**(*pattern*) **height**(*height*) **width**(*width*) **angle**(*angle*)

Follow the `addrect` keyword a set of specifications determining the position and style of the rectangle to be drawn.

The position and size of the rectangle can be specified with either the `pos or axisctr` arguments.

The `pos` argument specifies coordinates of the center of the rectangle in virtual space. *x1* is the center point X (horizontal) coordinate, and *y1* is the center point Y (vertical) coordinate. Coordinates are set 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 *x1* number specifies how many virtual inches to offset to the right from the origin. The second number *y1* specifies how many virtual inches to offset below the origin.

The `axisctr` argument specifies coordinates in units of the graph scale. *x1* is the center point X (horizontal) coordinate, and *y1* is the center point Y (vertical) coordinate.

For time-series graphs you must also specify which non-time based axis the y-coordinates's scale are based on, using l,t,r,b for left, top, right, bottom respectively. x-coordinates should be specified as a date/time.

For non-time series graphs you must specify the axis of scale of both x and y coordinates.

The `height` argument specifies the height of the rectangle. Similarly the `width` argument specifies its width. `angle` controls the rotation of the rectangle (in degrees).

The `linewidth` argument specifies the thickness of the rectangle outline. *lwidth* should be a number between ".25" and "5", indicating the width in points.

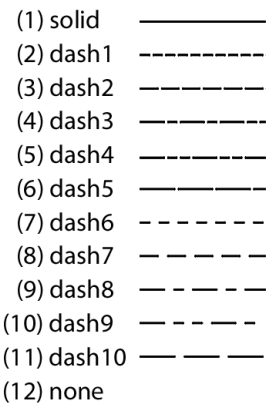
`arrowwidth` determines the size of the arrow head on the line. *awidth* can be either "small", "medium" or "large".

`color` specifies the color of the rectangle outline. The *color* value may set by using one of the color keywords (e.g., "blue"), by using the RGB values (e.g., "@RGB(255, 255, 0)"), or by specifying the components in hexadecimal (e.g., "@HEX(ff0000)").

The predefined colors are given by the keywords (with their RGB and HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
ltred	@rgb(255, 168, 168)	@hex(ffa8a8)
green	@rgb(0, 128, 0)	@hex(008000)
black	@rgb(0, 0, 0)	@hex(000000)
white	@rgb(255, 255, 255)	@hex(ffffff)
purple	@rgb(128, 0, 128)	@hex(800080)
orange	@rgb(255, 128, 0)	@hex(ff8000)
yellow	@rgb(255, 255, 0)	@hex(ffff00)
gray	@rgb(128, 128, 128)	@hex(808080)
ltgray	@rgb(192, 192, 192)	@hex(c0c0c0)

The `pattern` argument specifies the rectangle outline pattern. `pattern` can take a numerical value, or one of the corresponding keywords:



Examples

The commands

```
create m 1990 2000
smpl 1990 1995
series y=nrnd
smpl 1995 2000
y = 6+nrnd
smpl @all
freeze(gr) y.line
gr.addrect pos(1,1)  width(2) height(.7) angle(110) color(red)
    pattern(2) linewidth(3)
```

create a graph and adds a red rectangle that is centered 1 virtual inch from the top and 1 virtual inch from the left of the graph that is 2 virtual inches wide and 0.7 virtual inches tall. It uses a 3 pt dash1 line pattern. The rectangle is also rotated 110 degrees

The command

```
gr.addrect axisctr(1995, @mean(x),1)  width(30) height(.2) angle(-
    50) color(blue)
```

adds to a blue rectangle that is centered at 1995 and the mean of x in left axis units. It is 30 observations wide and 0.2 left axis units tall. It is also rotated -50 degrees

Cross-references

See "Drawing Lines and Arrows" on page 907 of *User's Guide I* for discussion.

See `Graph::addarrow` (p. 370), `Graph::addellipse` (p. 373), `Graph::addtext` (p. 378), and `Graph::delete` (p. 392).

See also `Graph::legend` (p. 402) and `Graph::textdefault` (p. 429).

addtext	Graph Procs

Place text in graphs.

When adding text in one of the four predefined positions (left, right, top, bottom), EViews 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*.

font*([face], [pt], [+/- **b**], [+/- **i**], [+/- **u**], [+/- **s**])*	Set characteristics of text font. The font name *(face)*, size *(pt)*, and characteristics are all optional. *face* should be a valid font name, enclosed in double quotes. *pt* 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.
textcolor(*arg*)	Sets the color of the text. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 381.
fillcolor(*arg*)	Sets the background fill color of the text box. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 381.

framecolor(*arg*)	Sets the color of the text box frame. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 381.
keep	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.

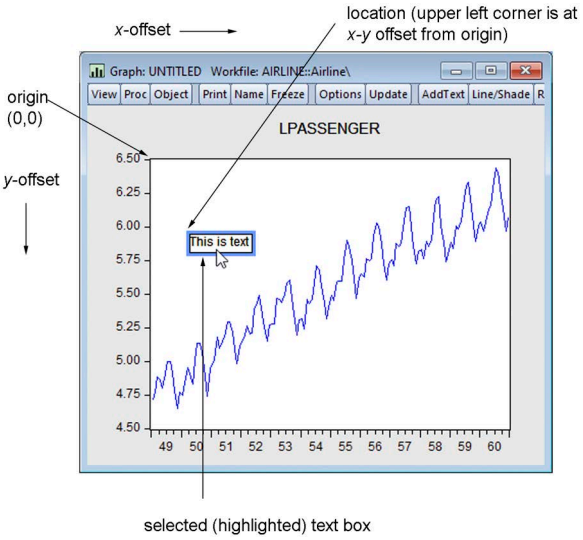
The following options control the position of the text:

t, ac	Top (above and centered over the graph).
l	Left rotated.
r	Right rotated.
b, bc	Below and centered over the graph.
bl	Below and left side of the graph.
br	Below and right side of the graph.
al	Above and left side of the graph.
ar	Above and right side of the graph.
ibl	Inside near the bottom left corner of the graph.
ibr	Inside near the bottom right corner of the graph.
itl	Inside near the top left corner of the graph.
itr	Inside near the top right corner of the graph.
just(*arg*)	Set the justification of the text, where *arg* may be: "c" (center), "l" (left - default), "r" (right).
x, lb	Enclose text in a large box.
sb	Enclose text in a small 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.

Color definitions

color_arg specifies the color to be employed in the arguments above. The color may be specified using predefined color names, by specifying the individual red-green-blue (RGB) components using the special "@RGB" function, or by specifying the individual red-green-blue (RGB) components in hexadecimal using the special "@HEX" function.

The predefined colors are given by the keywords (with their RGB and HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
ltred	@rgb(255, 168, 168)	@hex(ffa8a8)
green	@rgb(0, 128, 0)	@hex(008000)
black	@rgb(0, 0, 0)	@hex(000000)

white	@rgb(255, 255, 255)	@hex(ffffff)
purple	@rgb(128, 0, 128)	@hex(800080)
orange	@rgb(255, 128, 0)	@hex(ff8000)
yellow	@rgb(255, 255, 0)	@hex(ffff00)
gray	@rgb(128, 128, 128)	@hex(808080)
ltgray	@rgb(192, 192, 192)	@hex(c0c0c0)

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),
    framecolor(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 "Adding and Editing Text" on page 904 of *User's Guide I* for discussion.

See Graph::addarrow (p. 370), Graph::addellipse (p. 373), Graph::addrect (p. 376), and Graph::delete (p. 392).

See also Graph::legend (p. 402) and Graph::textdefault (p. 429).

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 sec-

ond 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 14. "Graphing Data," beginning on page 733 of *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. 1310), `xyarea` (p. 1327), `xybar` (p. 1330), `xyline` (p. 1333), `xypair` (p. 1337)).
top / t	Top axis for XY and scatter graphs (`scat` (p. 1310), `xyarea` (p. 1327), `xybar` (p. 1330), `xyline` (p. 1333), `xypair` (p. 1337)).

zerotop / zero-back	Draw zero line on [top / bottom] of other graph elements.
all / a	All axes.

Options

The options list may include any of the following options:

Data scaling options

linear	Linear data scaling *(default)*.
linearzero	Linear data scaling (include zero when auto range selection is employed).
log	Logarithmic scaling.
norm	Norm (standardize) the data prior to plotting.
range(*arg*)	Specifies the endpoints for the scale, where *arg* may be: "auto" (automatic choice), "minmax" (use the maximum and minimum values of the data), "*n1, n2*" (set minimum to *n1* and maximum to *n2*, *e.g.* "range(3, 9)").
overlap / -overlap	[Overlap / Do not overlap] scales on dual scale graphs.
invert / -invert	[Invert / do not invert] scale.
units(*arg*)	Specifies the units of the data, where *arg* may be: "n" (native), "p" (percent), "k" (thousands), "m" (millions), "b" (billions), "t" (trillions).
format(*option1 [,option2, ...]*)	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 = *arg*" (set number of decimal places, where *arg* may be an integer or "a" for auto), "prefix = *c*" (add a prefix character, where *c* may be a single quoted character or "" to remove the prefix), "suffix = *c*" (add a suffix character, where *c* 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.

zerotop / -zerotop	[Draw / Do not draw] the zero line on top of the graph.
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.
ticksauto	Allow EViews to determine whether to draw tickmarks on or between observations.
tickson	Draw tickmarks on observations.
ticksbtw	Draw tickmarks between observations.
ticksbtwns	Draw tickmarks between observations, removing space at the axis ends.
minor / -minor	[Allow / Do not allow] minor tick marks.
label / -label	[Place / Do not place] labels on the axes.
duallevel / -duallevel	[Allow / Do not allow] two row date labels on the observation axis.
font*([face], [pt], [+/- **b**], [+/- **i**], [+/- **u**], [+/- **s**])*	Set characteristics of axis font. The font name *(face)*, size *(pt)*, and characteristics are all optional. *face* should be a valid font name, enclosed in double quotes. *pt* 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.
textcolor(*arg*)	Sets the color of the axis text. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 386.

mirror / -mirror	[Label / Do not label] both left and right axes with duplicate axes (single scale graphs only).
angle(*arg*)	Set label angle, where *arg* 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.

Color definitions

color_arg specifies the color to be employed in the arguments above. The color may be specified using predefined color names, by specifying the individual red-green-blue (RGB) components using the special "@RGB" function, or by specifying the individual red-green-blue (RGB) components in hexadecimal using the special "@HEX" function.

The predefined colors are given by the keywords (with their RGB and HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
ltred	@rgb(255, 168, 168)	@hex(ffa8a8)
green	@rgb(0, 128, 0)	@hex(008000)
black	@rgb(0, 0, 0)	@hex(000000)
white	@rgb(255, 255, 255)	@hex(ffffff)
purple	@rgb(128, 0, 128)	@hex(800080)
orange	@rgb(255, 128, 0)	@hex(ff8000)
yellow	@rgb(255, 255, 0)	@hex(ffff00)
gray	@rgb(128, 128, 128)	@hex(808080)
ltgray	@rgb(192, 192, 192)	@hex(c0c0c0)

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 of mirroring of axes in single scale graphs.

```
mygra1.axis font("Times", 12, b, i) textcolor(blue)
```

sets the axis font to blue “Times” 12pt bold italic.

```
gra1.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 16. “Graph Objects,” on page 899 of *User’s Guide I* for a discussion of graph options.

See also `Graph::datelabel` (p. 390), `Graph::options` (p. 408), and `Graph::setelem` (p. 417).

bplabel	Graph Procs

Specify labeling of a boxplot axis.

Note that `bplabel` is no longer supported. See instead, `Graph::setobslabel` (p. 424).

clearhist	Graph Procs

Clear the contents of the history attribute for graph objects.

Removes the graph’s history attribute, as shown in the label view of the graph.

Syntax

graph_name.**clearhist**

Examples

```
g1.clearhist
g1.label
```

The first line removes the history from the graph G1, and the second line displays the label view of G1, including the now blank history field.

Cross-references

See “Labeling Objects” on page 123 of the *User’s Guide I* for a discussion of labels and display names.

See also `Graph::label` (p. 401).

clearremarks	Graph Procs

Clear the contents of the remarks attribute.

Removes the graph's remarks attribute, as shown in the label view of the graph.

Syntax

graph_name.**clearremarks**

Examples

```
g1.clearremarks
g1.label
```

The first line removes the remarks from the graph G1, and the second line displays the label view of G1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Graph::label` (p. 401).

copy	Graph Procs

Creates a copy of the graph.

Creates either a named or unnamed copy of the graph.

Syntax

graph_name.**copy**

graph_name.**copy** *dest_name*

Examples

```
g1.copy
```

creates an unnamed copy of the graph G1.

```
g1.copy g2
```

creates G2, a copy of the graph G1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

datalabel	Graph Procs

Control labeling of the data points in graphs.

`datelabel` sets options that are control the labeling of individual data points in graphs

Syntax

graph_name.**datalabel** *option_list*

Options

pos(*arg*)	Label position relative to the data point on the graph. The following positions are available: L – Left of point (vertically centered) R – Right of point (vertically centered) A – Above the point (horizontally centered) B – Below the point (horizontally centered) C – Centered on pointed (vertically and horizontally) Auto – Auto position
point(*arg*)	Which data points to label, where *arg* can be: All – label all data points First – label only the first visible data point Last – label only the last visible data point
label(*arg*)	Specifies the contents of the label, where *arg* is the custom label string containing text and keywords. The following keywords are replaced with their appropriate values: @legend – the legend label of the series @xlabel – the x-axis label of the data point @ylabel – the y-axis label of the data point
color(*arg*)	The label text color. Use '*none*' to set the text color as black. Nothing specified will set the text color to match the line color. If the *color* option is not specified, the color will be unchanged.
point(*arg*)	Which data points to label, where *arg* can be: All – label all data points First – label only the first visible data point Last – label only the last visible data point

Examples

```
graph1.datalabel point(first) pos(r) label(Start of
    recession\n(@xlabel(), @ylabel())) color()
```

will label the first data point of all the series in graph1. The label will be located to the right of the data point. The label will contain the 2 line string where the first line will read "Start of recession" and the second line will contain the comma separated *x-value* and *y-value* of the first data point enclosed in parenthesis (example: "(1948Q4,1.1)"). The label color will match the line color.

```
graph2.datalabel point(all) pos(a) label((@xlabel():@ylabel()))
    color(none)
```

will label all of the data points in graph2. The labels will appear above the associated data points in black and will be of the form "(x-value:y-value)" (example: "(1976:900.3)").

```
graph3.datalabel point(last) pos(l) label(@legend()-@ylabel())
    color()
```

will label all the series in graph3 but only the last data point for each series. The labels will appear the left of the associated data points and will be of the form "*legend label-y-value*" (example: "Nevada-35.6"). The label color will match the line color.

Cross-references

datalabel	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. 383) command with the "bottom" option. These options include tick control, label and font options, and grid lines.

Syntax

graph_name.**datelabel** *option_list*

Options

format("*datestring*")	*datestring* should be one of the supported data formats describing how the date should appear. The datestring 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. You may use the special single space datestring " " to indicate automatic formatting. You may also add "\n" to denote a new line providing the option to make the date string 2 lines. For example, "Month\nyear" will place the month on the first line and the year on the second. Note: there is a 2 line maximum. A second "\n" will therefore create an error. EViews provides considerable flexibility in formatting your dates. See "Date Formats" on page 106 of the *Command and Programming Reference* for a complete description.
interval(*step_size [,steps][,align_date]*)	where *step_size* takes one of the following values: "auto" (*steps* and *align_date* are ignored), "ends" (only label endpoints; *steps* and *align_date* are ignored), "all" (label every point; the *steps* and *align_date* options are ignored), "obs" (steps are one observation), "year" (steps are one year), "m" (steps are one month), "q" (steps are one quarter). *steps* is a number *(default* = 1*)* indicating the number of steps between labels. *align_date* is a date specified to receive a label. Note, the *align_date* should be in the units of the data being graphed, but may lie outside the current sample or workfile range.
span(*arg*)	Specify date label spanning: "auto" (automatic determination), "on" (turn spanning on; label start of period, tick on obs.), "between" (center label on period), "trimbetween" (center label on period, trim spaces at axis ends). Consider the case of a yearly label with monthly ticks. If `span` is on, the label is centered on the 12 monthly ticks. If the `span` option is off, year labels are put on the first quarter or month of the year.
end / -end	[Use / Do not use] end-of-period labeling.
duallevel / -duallevel	[Allow / Do not allow] two row date labels on the observation axis.

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[q]mm)
```

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.

```
graph1.datelabel format("Month dd\nYYYY")
```

specifies time axis label will have 2 lines. The first line will contain the full month name and day and the second line will contain the 4 digit year.

Cross-references

See Chapter 16. "Graph Objects," on page 899 of *User's Guide I* for a discussion of graph options.

See also `Graph::axis` (p. 383), `Graph::options` (p. 408), and `Graph::setelem` (p. 417).

dates	Graph Procs

See the replacement command `Graph::datelabel` (p. 390).

delete	Graph Procs

Removes all objects of specified type from a graph object.

Syntax

```
graph_name.delete object_type
```

where *object_type* includes one or more of the following: 'line', 'shade', 'text', 'ellipse', 'rectangle', and 'arrow'.

Examples

The following removes all line and shade objects from GRA1

```
gra1.delete line shade
```

To remove all text objects from GRA1:

```
gra1.delete text
```

Cross-references

See "Drawing Lines and Arrows" on page 907 of *User's Guide I* for discussion.

See also `Graph::addarrow` (p. 370), `Graph::addellipse` (p. 373), `Graph::addrect` (p. 376), `Graph::addtext` (p. 378), and `Graph::delete` (p. 392)

display	Graph View

Display table, graph, or spool output in the graph object window.

Display the contents of a table, graph, or spool in the window of the graph object.

Syntax

graph_name.**display** *object_name*

Examples

```
graph1.display tab1
```

Display the contents of the table TAB1 in the window of the object GRAPH1.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names.

See also `Graph::label` (p. 401).

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 123 of *User’s Guide I* for a discussion of labels and display names.

See also `Graph::label` (p. 401) and `Graph::legend` (p. 402).

draw	Graph Procs

Place horizontal or vertical lines and shaded areas on the graph.

Syntax

graph_name.**draw**(*draw_type*, *axis_id [,options]*) *position [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.

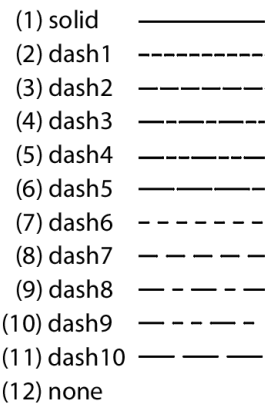
axis_id may take the values:

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 *position*. If drawing a shaded area, you can either specify a start and end position (*position* and *position2*), sample object, or sample string 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(*arg*)	Specifies the color of the line or shade. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions," beginning on page 395. 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(*index*)	Sets the line pattern to the type specified by *index*. *index* can be an integer from 1 to 12 or one of the matching keywords ("solid", "dash1" through "dash10", "none").  (1) solid (2) dash1 (3) dash2 (4) dash3 (5) dash4 (6) dash5 (7) dash6 (8) dash7 (9) dash8 (10) dash9 (11) dash10 (12) none The "none" keyword turns on solid lines.
width(*n1*)	Specify the width, where *n1* is the line width in points (used only if object_type is "line" or "dashline"). The default is 0.5 points.
top	Specifies that the line be drawn on top of the graph. (Note that this option has no effect on shades.)

Color definitions

color_arg specifies the color to be employed in the arguments above. The color may be specified using predefined color names, by specifying the individual red-green-blue (RGB) com-

ponents using the special "@RGB" function, or by specifying the individual red-green-blue (RGB) components in hexadecimal using the special "@HEX" function.

The predefined colors are given by the keywords (with their RGB and HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
ltred	@rgb(255, 168, 168)	@hex(ffa8a8)
green	@rgb(0, 128, 0)	@hex(008000)
black	@rgb(0, 0, 0)	@hex(000000)
white	@rgb(255, 255, 255)	@hex(ffffff)
purple	@rgb(128, 0, 128)	@hex(800080)
orange	@rgb(255, 128, 0)	@hex(ff8000)
yellow	@rgb(255, 255, 0)	@hex(ffff00)
gray	@rgb(128, 128, 128)	@hex(808080)
ltgray	@rgb(192, 192, 192)	@hex(c0c0c0)

Examples

```
graph1.draw(line, left, @rgb(0,0,255)) 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(shade, bottom, @rgb(255,0,0)) 1980:1 1990:2 if x>.5
```

draws red shaded vertical regions bounded by the dates "1980:1" and "1990:2" where the series x has a value greater than .5.

```
graph1.draw(shade, bottom, @rgb(0,128,0)) mysample
```

draws green shaded vertical regions that match the `mysample` object.

```
graph1.draw(line, bottom, pattern(dash1)) 1985:1
```

draws a vertical dashed line at "1985:1".

Cross-references

See Chapter 16. "Graph Objects," on page 899 of *User's Guide I* for a discussion of graph options.

See `Graph::drawdefault` (p. 397) 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.

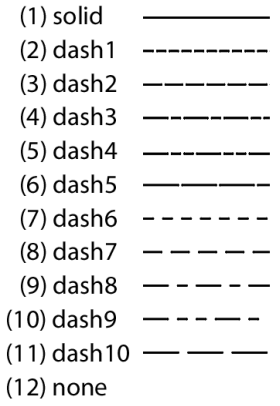
Syntax

graph_name.**drawdefault** *draw_options*

where *draw_options* may include one or more of the following:

linecolor(*arg*)	Sets the default color for lines. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a full description of the keywords, see "Color definitions," beginning on page 398.
shadecolor(*arg*)	Sets the default color for shades. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a full description of the keywords, see "Color definitions," beginning on page 398
width(*n1*)	Specify the width, where *n1* is the line width in points (used only if object_type is "line" or "dashline"). The default is 0.5 points.

pattern(*index*)	Sets the default line pattern to the type specified by *index*. *index* can be an integer from 1 to 12 or one of the matching keywords ("solid", "dash1" through "dash10", "none"). Sets the line pattern to the type specified by *index*. *index* can be an integer from 1 to 12 or one of the matching keywords ("solid", "dash1" through "dash10", "none").



The "none" keyword turns on solid lines.

existing	Apply the default settings to all existing line/shade objects in the graph.

Color definitions

color_arg specifies the color to be employed in the arguments above. The color may be specified using predefined color names, by specifying the individual red-green-blue (RGB) components using the special "@RGB" function, or by specifying the individual red-green-blue (RGB) components in hexadecimal using the special "@HEX" function.

The predefined colors are given by the keywords (with their RGB and HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
ltred	@rgb(255, 168, 168)	@hex(ffa8a8)
green	@rgb(0, 128, 0)	@hex(008000)
black	@rgb(0, 0, 0)	@hex(000000)
white	@rgb(255, 255, 255)	@hex(ffffff)
purple	@rgb(128, 0, 128)	@hex(800080)
orange	@rgb(255, 128, 0)	@hex(ff8000)

yellow	@rgb(255, 255, 0)	@hex(ffff00)
gray	@rgb(128, 128, 128)	@hex(808080)
ltgray	@rgb(192, 192, 192)	@hex(c0c0c0)

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 16. “Graph Objects,” on page 899 of *User’s Guide I* for a discussion of graph options.

See Graph::draw (p. 394).

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. 1269)).
band	Area band graph (band (p. 1272)).

bar	Bar graph (bar (p. 1275)).
boxplot	Boxplot graph (boxplot (p. 1279)).
distplot	Distribution graph (distplot (p. 1283)).
dot	Dot plot graph (dot (p. 1290)).
errbar	Error bar graph (errbar (p. 1294)).
hilo	High-low(-open-close) graph (hilo (p. 1296)).
line	Line graph (line (p. 1298)).
pie	Pie graph (pie (p. 1304)).
qqplot	Quantile-Quantile graph (qqplot (p. 1306)).
scat	Scatterplot—same as XY, but lines are initially turned off, symbols turned on, and a 3×3 frame is used (scat (p. 1310)).
scatmat	Matrix of scatterplots (scatmat (p. 1315)).
scatpair	Scatterplot pairs graph (scatpair (p. 1318)).
seasplot	Seasonal line graph (seasplot (p. 1322)).
spike	Spike graph (spike (p. 1323)).
xyarea	XY line-symbol graph with one X plotted against one or more Y's using existing line-symbol settings (xyarea (p. 1327)).
xybar	XY line-symbol graph with one X plotted against one or more Y's using existing line-symbol settings (xybar (p. 1330)).
xyline	Same as XY, but symbols are initially turned off, lines turned on, and a 4×3 frame is used (xyline (p. 1333)).
xypair	Same as XY but sets XY settings to display pairs of X and Y plotted against each other (xypair (p. 1337)).

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. 1275) 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_scat gr_pie
```

creates a multiple graph object named MYGRA that merges three graph objects named GR_LINE, GR_SCAT, and GR_PIE.

Cross-references

See Chapter 16. "Graph Objects," on page 899 of *User's Guide I* for a general discussion of graphs.

See also `freeze` (p. 457) and `Graph::merge` (p. 406).

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 *text*.
s	Sets the source field to *text*.

u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User's Guide I* for a discussion of labels.

See also `Graph::displayname` (p. 393).

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. 417) for setting legends for individual graphs in a multiple graph.

Syntax

graph_name.**legend** *option_list*

Options

columns*(arg) (default* = "auto"*)*	Columns for legend: "auto" (automatically choose number of columns), *int* (put legend in specified number of columns).
display/–display	Display/do not display the legend.
inbox/–inbox	Put legend in box/remove box around legend.

position(*arg*)	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), “(*h*, *v*)” (the first number *h* specifies the number of virtual inches to offset to the right from the origin. The second number *v* specifies the virtual inch offset below the origin. The origin is the upper left hand corner of the graph).
font*([face], [pt], [+/- **b**], [+/- **i**], [+/- **u**], [+/- **s**])*	Set characteristics of legend font. The font name *(face)*, size *(pt)*, and characteristics are all optional. *face* should be a valid font name, enclosed in double quotes. *pt* 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.
textcolor(*arg*)	Sets the color of the legend text. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the “@RGB” or “@HEX” functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the “@HEX” function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see “Color definitions” on page 404.

fillcolor(*arg*)	Sets the background fill color of the legend box. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 404.
framecolor(*arg*)	Sets the color of the legend box frame. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 404.

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.

Color definitions

color_arg specifies the color to be employed in the arguments above. The color may be specified using predefined color names, by specifying the individual red-green-blue (RGB) components using the special "@RGB" function, or by specifying the individual red-green-blue (RGB) components in hexadecimal using the special "@HEX" function.

The predefined colors are given by the keywords (with their RGB and HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
ltred	@rgb(255, 168, 168)	@hex(ffa8a8)
green	@rgb(0, 128, 0)	@hex(008000)
black	@rgb(0, 0, 0)	@hex(000000)
white	@rgb(255, 255, 255)	@hex(ffffff)
purple	@rgb(128, 0, 128)	@hex(800080)

orange	@rgb(255, 128, 0)	@hex(ff8000)
yellow	@rgb(255, 255, 0)	@hex(ffff00)
gray	@rgb(128, 128, 128)	@hex(808080)
ltgray	@rgb(192, 192, 192)	@hex(c0c0c0)

Examples

```
mygra1.legend display position(l) inbox
```

places the legend of MYGRA1 in a box to the left of the graph.

```
mygra1.legend position(.2,.2) -inbox
```

places the legend of MYGRA1 within the graph, indented slightly from the upper left corner with no box surrounding the legend text.

```
mygra1.legend font("Times", 12, b, i) textcolor(red)
    fillcolor(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 16. "Graph Objects," on page 899 of *User's Guide I* for a discussion of graph objects in EViews.

See `Graph::addtext` (p. 378) and `Graph::textdefault` (p. 429). See `Graph::setelem` (p. 417) for changing legend text and other graph options.

makegroup — Graph Procs

Creates a group object containing all the series in the graph.

Syntax

graph_name.**makegroup** *group_name*

group_name is an optional new group name. Group will be untitled if *group_name* is not specified.

Examples

```
mygraph.makegroup mynewgroup
```

Creates new group called mynewgroup.

```
mygraph.makegroup
```

Creates an untitled group.

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 16. “Graph Objects,” on page 899 of *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. 414), 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.scat id w h
g2.name(1)
g2.name(2) weight
g2.name(3) height
g2.legend(l)
```

The first line creates a scatter diagram named G2. The second line removes the legend of the horizontal axis, and the third and fourth lines replace the legends of the variables on the vertical axis. The last line moves the legend to the left side of the graph.

Cross-references

See Chapter 16. "Graph Objects," on page 899 of *User's Guide I* for a discussion of working with graphs.

See also `Graph::displayname` (p. 393).

olepush

Graph Procs

Push updates to OLE linked objects in open applications.

Syntax

graph_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

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*

Options

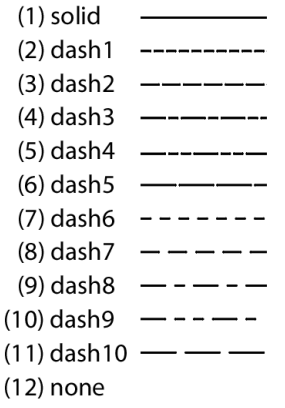
Basic Graph Options

legend / -legend	Turn on and off legend.
size(*w*, *h*)	Specifies the size of the plotting frame in virtual inches (*w* = width, *h* = height).
lineauto	Use solid lines when drawing in color and use patterns and grayscale when drawing in black and white.
linesolid	Always use solid lines.
linepat	Always use line patterns.
color / -color	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.
barlabelabove / -barlabelabove	[Place / Do not place] text value of data above bar in bar graph.
barlabelinside / -barlabelinside	[Place / Do not place] text value of data inside bar in bar graph.
barlabelnone	Remove text value of data from bar graph.
outlinebars / -outlinebars	[Outline / Do not outline] bars in a bar graph.
outlinearea / -outlinearea	[Outline / Do not outline] areas in an area graph.
outlineband / -outlineband	[Outline / Do not outline] bands in an area band graph.

barspace /-barspace	[Put / Do not put] space between bars in bar graph.
pielabel / -pielabel	[Place / Do not place] text value of data in pie chart.
automult/-automult	[Auto reduce / Do not autoreduce] frame size in multiple graphs to make text appear larger
dual/-dual	[Overlap / Do not overlap] scales (no cross).
barfade(*arg*)	Sets the fill fade of the bars in a bar graph. *arg* may be: "none" (solid fill - *default*), "3d" (3D rounded fill), "lzero" (light at zero), "dzero" (dark at zero).
antialias(*arg*)	Sets anti-aliasing to smooth the appearance of data lines in the graph. *arg* may be: "auto" (EViews uses anti-aliasing where appropriate - *default*), "on", or "off".
interpolate(*arg*)	Sets the interpolation method to estimate values between two known data points in the graph. *arg* may be: "linear" (no interpolation), "mild" (mild spline), "medium" (medium spline), or "full" (full spline).
stackposneg / -stackposneg	For bar graphs, stack positive and negative values separately (Excel style).

Graph Grid Options

gridl / -gridl	[Turn on / Turn off] grid lines on the left scale.
gridr / -gridr	[Turn on / Turn off] grid lines on the right scale.
gridb / -gridb	[Turn on / Turn off] grid lines on the bottom scale.
gridt / -gridt	[Turn on / Turn off] grid lines on the top scale.
gridnone	No grid lines (turns of time scale grid).
gridauto	Allow EViews to place grid lines at automatic intervals.
gridcust(*freq [,step]*)	Place grid lines at custom intervals, specified by *freq*. *freq* may be: "obs" or "o" (Step = One obs), "year" or "y" (Step = Year), "quarter" or "q" (Step = Quarter), "month" or "m" (Step = Month), "day" or "d" (Step = Day), "user" or "u" (Step = custom). You may optionally specify a step for the interval. If not specified, the default is the last grid step used for this graph, or 1 if a step has never been specified.

gridcolor(*arg*)	Sets the grid line color. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 412.
gridwidth(*n*)	Sets the width of the grid lines in points. *n* should be a number between 0.25 and 5.
gridpat(*index*)	Sets the default line pattern to the type specified by *index*. *index* can be an integer from 1 to 12 or one of the matching keywords ("solid", "dash1" through "dash10", "none"). Sets the line pattern to the type specified by *index*. *index* can be an integer from 1 to 12 or one of the matching keywords ("solid", "dash1" through "dash10", "none").  The "none" keyword turns on solid lines.
gridontop / -gridontop	[Draw / Do not draw] the grid lines on top of the graph.

Background and Frame Options

fillcolor(*arg*)	Sets the fill color of the graph frame. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 412.
backcolor(*arg*)	Sets the background color of the graph. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 412.
framecolor(*arg*)	Sets the background color of the graph frame. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 412..
fillfade(*arg*)	Sets the fill fade of the graph frame. *arg* may be: "none" (solid frame fill - *default*), "ltop" (light at top), "dtop" (dark at top).
backfade(*arg*)	Sets the background fade of the graph. *arg* may be: "none" (solid background - *default*), "ltop" (light at top), "dtop" (dark at top).
framewidth(*n*)	Sets the width of the graph frame in points. *n* should be a number between 0.25 and 5.

frameaxes(*arg*)	Specifies which frame axes to display. *arg* 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.
indenth(*n*)	Sets the horizontal indentation of the graph from the graph frame in virtual inches. *n* should be a number between 0 and 0.75.
indentv(*n*)	Sets the vertical indentation of the graph from the graph frame in virtual inches. *n* should be a number between 0 and 0.75.
inbox / -inbox	[Show / Do not show] the graph frame on axes that do not have data assigned to them.
background / -background	[Include / Do not include] the background color when exporting or printing the graph.

Sample Break and NA Handling

drop *(default)*	For a graph with a non-contiguous sample, drop the excluded observations from the graph scale.
connect	For a graph with missing values or a non-contiguous sample, connect non-missing observations.
disconnect	For a graph with missing values or a non-contiguous sample, disconnect non-missing observations.
pad	For a graph with a non-contiguous sample, pad the graph scale with the excluded observations
segment	For a graph with a non-contiguous sample, drop the excluded observations from the graph scale and draw vertical lines at the seams in the observation scale.

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.

Color definitions

color_arg specifies the color to be employed in the arguments above. The color may be specified using predefined color names, by specifying the individual red-green-blue (RGB) components using the special “@RGB” function, or by specifying the individual red-green-blue (RGB) components in hexadecimal using the special “@HEX” function.

The predefined colors are given by the keywords (with their RGB and HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
ltred	@rgb(255, 168, 168)	@hex(ffa8a8)
green	@rgb(0, 128, 0)	@hex(008000)
black	@rgb(0, 0, 0)	@hex(000000)
white	@rgb(255, 255, 255)	@hex(ffffff)
purple	@rgb(128, 0, 128)	@hex(800080)
orange	@rgb(255, 128, 0)	@hex(ff8000)
yellow	@rgb(255, 255, 0)	@hex(ffff00)
gray	@rgb(128, 128, 128)	@hex(808080)
ltgray	@rgb(192, 192, 192)	@hex(c0c0c0)

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 16. "Graph Objects," on page 899 of *User's Guide I* for a discussion of graph options in EViews.

See also `Graph::axis` (p. 383), `Graph::datelabel` (p. 390), and `Graph::setelem` (p. 417).

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), Portable Network Graphics (.PNG), Portable Document Format (.PDF), LaTeX (.TEX), Markdown (.MD), or MPEG-4 (.mp4).

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, JPG, PNG, PDF, TEX, MD, or MP4 extension. The MD (Markdown) setting uses very basic syntax and should be usable in most editors.

If an explicit path is not specified, the file will be stored in the default directory, as set in the **File Locations** global options.

Graph Options

t = *file_type*	Specifies the file type, where *file_type* 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"), Portable Network Graphics ("png"), Portable Document Format ("pdf"), LaTeX ("tex"), Markdown ("md"), or MPEG-4 ("mp4"). Files will be saved with the ".emf", ".wmf", ".eps", ".bmp", ".gif", ".jpeg", ".png", ".pdf", ".tex", ".md", or ".mp4" extensions, respectively.
u = *units*	Specify units of measurement, where *units* 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 = *width*	Set width of the graphic in the selected units.
h = *height*	Set height of the graphic in the selected units.
c / -c	[Save / Do not save] the graph in color.

trans / -trans	[Set / Do not set] background to transparent (for graph formats which support transparency).
d = *dpi*	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 Options

box / -box	[Save / Do not save] the graph with a bounding box. The bounding box is an invisible rectangle placed around the graphic to indicate its boundaries. The default is taken from the global graph export settings.
land	Save the graph in landscape orientation. The default uses portrait mode.
prompt	Force the dialog to appear from within a program.

LaTeX Options

texspec / -texspec	[Include / Do not include] the full LaTeX documentation specification in the LaTeX output. The default behavior is taken from the global default settings.

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 16. "Graph Objects," beginning on page 899 of *User's Guide I* for a discussion of graphs.

scale	Graph Procs

The **scale** command is supported for backward compatibility, but has been replaced by the `Graph::axis` (p. 383) command, which handles all axis and scaling options.

setattr	Graph Procs

Set the object attribute.

Syntax

graph_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the "revised" attribute in the object A to the string "never", and extracts the attribute into the string object S.

Cross-references

See "Adding Custom Attributes in the Label View" on page 123 and "Adding Your Own Label Attributes" on page 70 of *User's Guide I*.

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*(arg)* *(default* = "fixed"*)*	Set the width settings for the boxplots, where *arg* is one of: "fixed" (uniform width), "n" (proportional to sample size), "rootn" (proportional to the square root of sample size).
ci(*arg)* *(default* = "shade"*)*	Set the display method for the median confidence intervals, where *arg* is one of: "none" (do not display), "shade" (shaded intervals), "notch" (notched intervals).

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 852 of *User's Guide I* for a description of boxplots.

See `Graph::setelem` (p. 417) to modify line and symbol attributes. See also `Graph::options` (p. 408) and `Graph::axis` (p. 383).

setelem	Graph Procs

Set individual line, bar and legend options for each series in the graph.

Syntax

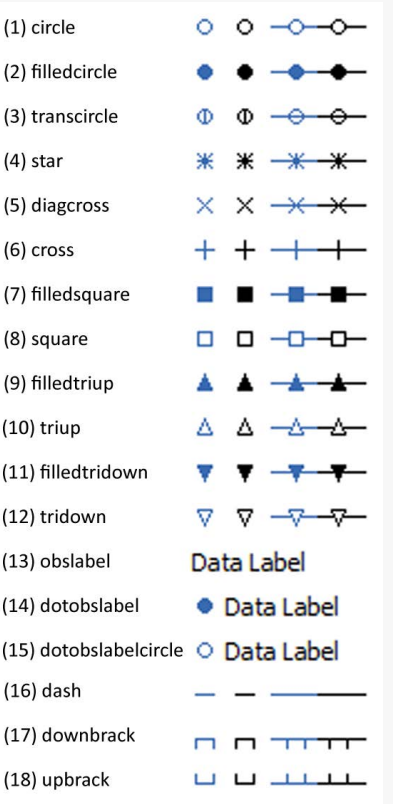
graph_name.**setelem**(*graph_elem*) *argument_list*

where *graph_elem* is the identifier for the graph element whose options you wish to modify:

integer	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.
box_elem	Boxplot element to be modified: box ("b"), median ("med"), mean ("mean"), near outliers ("near" or "no"), far outliers ("far" or "fo"), whiskers ("w"), staples ("s"). For boxplot graphs only.

The *argument* list for `setelem` may contain one or more of the following:

symbol(*arg*)



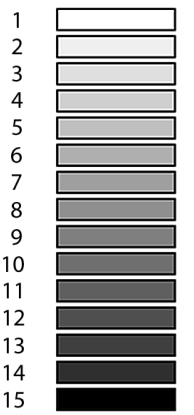
Sets the drawing symbol: *arg* can be an integer from 1–18, or one of the matching keywords. "obslabel" and "dotobslabel", and "dotobslabelcircle" use the observation label as the symbol.

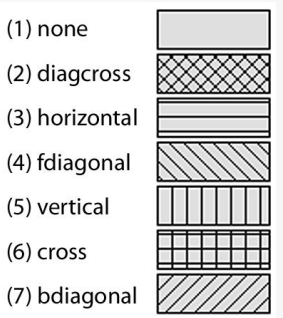
Selecting a symbol automatically turns on symbol use.

The "none" option turns off symbol use.

symbolsize(*arg*), symsize(*arg*)	Sets the symbol size. *arg* 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).
linecolor(*arg*), lcolor(*arg*)	Sets the line and symbol color. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the “@RGB” or “@HEX” functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the “@HEX” function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see “Color definitions” on page 422.
linewidth(*n1*), lwidth(*n1*)	Sets the line and symbol width: *n1* should be a number between “.25” and “5”, indicating the width in points.
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. 408), for details). You may need to set the graph option for “linepat” to enable the display of line patterns. See `Graph::options` (p. 408). (1) solid (2) dash1 (3) dash2 (4) dash3 (5) dash4 (6) dash5 (7) dash6 (8) dash7 (9) dash8 (10) dash9 (11) dash10 (12) none 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.

lineopacity(*arg1[, arg2]*)	Sets the line opacity to the value or setting specified by the *arg* values. The *arg* can be an value from 0 to 1, or one of the keywords “auto”, “on”, “off”. If the opacity setting currently “on” or “auto” (default), setting the level to 0.0 will make (in relevant cases) the object completely transparent (0% opacity) while a value of 1.0 will make the object completely opaque (100% opacity). Setting opacity to “off” will make the object fully opaque.
fillcolor(*arg*), fcolor(*arg*)	Sets the fill color for symbols, bars, and pies. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the “@RGB” function or “@HEX” functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the “@HEX” function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see “Color definitions” on page 422.
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).



fillhatch(*arg*), hatch(*arg*)	Sets the hatch characteristics for bars and pies: *arg* can be an integer from 1–7, or one of the matching keywords. 
fillopacity(*arg1[, arg2]*),	Sets the fill opacity to the opacity to the value or setting specified by the *arg* values. The *arg* can be an value from 0 to 1, or one of the keywords "auto", "on", "off". If the opacity setting currently "on" or "auto" (default), setting the level to 0.0 will make (in relevant cases) the object completely transparent (0% opacity) while a value of 1.0 will make the object completely opaque (100% opacity). Setting opacity to "off" willmake the object fully opaque.
preset(*n1*)	Sets line and fill characteristics to the specified EViews preset values, where *n1* is an integer from 1–30. Simultaneously sets "linecolor", "linepattern", "linewidth", "symbol", "fillcolor", "fillgray", and "fillhatch" to the EViews predefined definitions for graph element *n1*. When applied to boxplots, the line color of the specified element will be applied to the box, whiskers, and staples.
default(*n1*)	Sets line and fill characteristics to the specified user-defined default settings where *n1* 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 *n1*. When applied to boxplots, the line color of the specified settings will be applied to the box, whiskers, and staples.
axis(*arg*), scale(*arg*)	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. 1310), `xyarea` (p. 1327), `xybar` (p. 1330), `xyline` (p. 1333), `xypair` (p. 1337)).
legend(*str*)	Assigns legend text for the element. *str* will be used in the legend to label the element.

Color definitions

color_arg specifies the color to be employed in the arguments above. The color may be specified using predefined color names, by specifying the individual red-green-blue (RGB) components using the special "@RGB" function, or by specifying the individual red-green-blue (RGB) components in hexadecimal using the special "@HEX" function.

The predefined colors are given by the keywords (with their RGB and HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
ltred	@rgb(255, 168, 168)	@hex(ffa8a8)
green	@rgb(0, 128, 0)	@hex(008000)
black	@rgb(0, 0, 0)	@hex(000000)
white	@rgb(255, 255, 255)	@hex(ffffff)
purple	@rgb(128, 0, 128)	@hex(800080)
orange	@rgb(255, 128, 0)	@hex(ff8000)
yellow	@rgb(255, 255, 0)	@hex(ffff00)
gray	@rgb(128, 128, 128)	@hex(808080)
ltgray	@rgb(192, 192, 192)	@hex(c0c0c0)

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(l)
graph1.setelem(2) scale(l)
graph1.setelem(3) scale(r)
```

create a dual scale graph where the first two series are scaled together and labeled on the left axis, and the third series is scaled and labeled on the right axis.

```
graph1.setelem(2) legend("gross domestic product")
```

sets the legend for the second graph element.

Cross-references

See Chapter 16. "Graph Objects," on page 899 of *User's Guide I* for a discussion of graph options in EViews.

See also Graph::axis (p. 383), Graph::datelabel (p. 390) and Graph::options (p. 408).

setfont	Graph Procs

Set the font for text in the graph.

Syntax

graph_name.**setfont** *font_args*

The *font_args* may include one or more of the following:

*type([face], [pt], [+/- **b**], [+/- **i**], [+/- **u**], [+/- **s**])*	Set characteristics of the font for the graph element *type*. The font name *(face)*, size *(pt)*, and characteristics are all optional. *face* should be a valid font name, enclosed in double quotes. *pt* 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.

and *type* is one of "all", "axes", "legend", "text", "obs", where "axes" refers to the axes labels, "legend" refers to the graph legend, "text" refers to the added text, "obs" refers to the observation scale, and "all" refers to all of the elements.

Examples

```
mygraph.setfont axes("Times", 20, b)
```

sets the font to Times, 20pt, bold for all of the graph elements.

```
mygraph.setfont text("Arial") legend("Helvetica")
```

sets the added text font to Arial and the legend font to Helvetica.

Cross-references

See Chapter 16. "Graph Objects," on page 899 of *User's Guide I* for a discussion of graph options.

See also Graph::datelabel (p. 390), Graph::axis (p. 383), Graph::options (p. 408) and Graph::setelem (p. 417).

setobslabel	Graph Procs

Sets custom axis labels for the observation scale of a graph.

Syntax

graph_name.**setobslabel***([step_options,] init_options) [string1 string2 ...]*

Follow the keyword with a list of axis labels, or the name of a series when the "series" *init_option* is used.

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

start[, step]	*start* should be the observation number of the first label to modify. *step* defines the number of observations to skip between applying labels.

Init options

init_options (default = "blank"*)*	*init_options* sets initialization options for the labels. For a frozen graph (updating off), you may use the keywords: "current" (keep current labels, or initialize the labels with standard observation labels if custom labels do not currently exist, then add the labels provided), "obsnum" (initialize with observation numbers), or "blank" (set all labels to empty strings, then add the labels provided). For live or frozen graphs, you may use the keywords: "series" (initialize the labels with the values of a series; follow the command with the name of a series instead of labels), or "clear" (delete custom labels if they exist and return to automatic labeling).

Examples

Given a graph GRA1 with updating turned off, change the first label to "CA" using 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 in the frozen graph GRAPH2 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.

Say we have an alpha series in our workfile, ALPHA01, whose values are: “CA”, “OR”, “WA”, etc. To use these values as axis labels, use the *series* option and specify a series name in place of labels:

```
gra3.setobslabel(series) alpha01
```

This command creates labels on the time axis, using values in ALPHA01 to label the observations with: “CA”, “OR”, “WA”, etc.

Cross-references

See Chapter 16. “Graph Objects,” on page 899 of *User’s Guide I* for a discussion of graph options.

See also `Graph::datelabel` (p. 390), `Graph::axis` (p. 383), `Graph::options` (p. 408) and `Graph::setelem` (p. 417).

setupdate	Graph Procs

Set the update state of a graph object.

Syntax

graph_name.**setupdate***(options) [sample]*

Follow the name of the graph with a period, the keyword `setupdate`, and the update setting.

Optionally, include a sample with the "manual" or "automatic" options to restrict updates to data changes made within the sample period. If you do not include a sample, updates will occur according to changes in the workfile sample.

Options

"off" or "o"	Turn updating off.
"manual" or "m"	Update when requested (with the `Graph::update` (p. 432) command), or when the graph type is changed.
"auto" or "a"	Update whenever the update condition is met. If a sample is specified, an update will occur when data changes within the sample. If no sample is specified, updates will occur when data or the workfile sample changes.

Examples

```
gr1.setupdate(o)
```

This command turns off updating for graph GR1.

```
gr1.setupdate(a)
```

turns on automatic updating for graph GR1, according to the workfile sample. Whenever the underlying data or the workfile sample changes, GR1 will be updated with the changes.

```
gr2.setupdate(m) 1992 1993
```

turns on manual updating for graph GR2, for the sample period 1992 to 1993. When the graph is manually updated, using the `update` (p. 432) command, changes in data between 1992 and 1993 will be updated.

Cross-references

See Chapter 16. "Graph Objects," beginning on page 900 of *User's Guide I* for a discussion of graph updating options.

See Graph::update (p. 432).

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

```
gra1.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,

```
gra1.sort(1,-2)
```

sorts first in ascending order by X and then in descending order by Y.

```
gra1.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 *template* is the name of a graph in the workfile.
e	Apply template settings to existing text and line/fill options.
b / -b	[Apply / Remove] bold modifiers of the specified *predefined* template style.
w / -w	[Apply / Remove] wide modifiers of the specified *predefined* template style.
axis / -axis	[Apply / Remove] axis modifiers of the specified template.
legend / -legend	[Apply / Remove] legend modifiers of the specified template.

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. 378) or Graph::draw (p. 394) 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 925 of *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:

font(*[face], [pt], [+/- b], [+/- i], [+/- u], [+/- s]*)	Set characteristics of default text font. The font name *(face)*, size *(pt)*, and characteristics are all optional. *face* should be a valid font name, enclosed in double quotes. *pt.* 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.
textcolor(*arg*)	Sets the default color of the text. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 431.
fillcolor(*arg*)	Sets the default background fill color of the text box. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 431.
framecolor(*arg*)	Sets the default color of the text box frame. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 431.
existing	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.

Color definitions

color_arg specifies the color to be employed in the arguments above. The color may be specified using predefined color names, by specifying the individual red-green-blue (RGB) components using the special "@RGB" function, or by specifying the individual red-green-blue (RGB) components in hexadecimal using the special "@HEX" function.

The predefined colors are given by the keywords (with their RGB and HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
ltred	@rgb(255, 168, 168)	@hex(ffa8a8)
green	@rgb(0, 128, 0)	@hex(008000)
black	@rgb(0, 0, 0)	@hex(000000)
white	@rgb(255, 255, 255)	@hex(ffffff)
purple	@rgb(128, 0, 128)	@hex(800080)
orange	@rgb(255, 128, 0)	@hex(ff8000)
yellow	@rgb(255, 255, 0)	@hex(ffff00)
gray	@rgb(128, 128, 128)	@hex(808080)
ltgray	@rgb(192, 192, 192)	@hex(c0c0c0)

Examples

```
graph1.textdefault font("Arial", b) fillcolor(gray)
    framecolor(@rgb(0,0,255)) 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 and blue frame color. 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 16. "Graph Objects," on page 899 of *User's Guide I* for a discussion of graph options.

See `Graph::addtext` (p. 378) and `Graph::legend` (p. 402).

update	Graph Procs

Update graph.

This command updates a graph that has updating turned on.

Syntax

graph_name.**update**

Examples

```
graph1.update
```

If GRAPH1 is a graph with manual updating enabled, this command instructs the graph to update its data. If the graph has automatic updating enabled, this command is unnecessary, as it will simply repeat the automatic update. For a graph with updating off, this command does nothing.

Cross-references

See Chapter 16. “Graph Objects,” beginning on page 900 of *User’s Guide I* for a discussion of graph updating options.

See `Graph::setupdate` (p. 426).

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. 476).

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
```

You may use the wildcard characters "*" and "?" to match more than one series in the workfile, and you may use the keywords "and" and "not" to specify that certain items should be excluded from the group:

```
group g a* and *1
```

makes a group G containing all series whose names begin with the letter "a" and end with "1", while

```
group g a* b* not *1 *2
```

makes a group G containing all series whose names begin with either letter "a" or "b" that do not end with either "1" or "2".

Additionally, a number of object procedures will automatically create a group.

Note: to convert data between groups and matrices, see "Copying Data Between Matrices and Series/Groups" on page 290, `stom` (p. 598), `stomna` (p. 599), `mtos` (p. 522), all in the *Command and Programming Reference*.

Group Views

cause..... pairwise Granger causality tests (p. 439).
coint ..... test for cointegration between series in a group (p. 442).
cor ..... correlation matrix between series (p. 451).
correl ..... correlogram of the first series in the group (p. 455).
cov..... covariance matrix between series (p. 455).
cross ..... cross correlogram of the first two series (p. 459).
display ..... display table, graph, or spool in object window (p. 469).
dtable..... dated data table (p. 472).
dups..... duplicates display for observations in the group (p. 472).
freq ..... frequency table n-way contingency table (p. 473).
label..... label information for the group (p. 478).

Group Procs

resample ..... resample from rows of group (p. 491).
setattr..... set the value of an object attribute (p. 493).
setfillcolor ..... set custom spreadsheet fill coloring for the group (p. 493).
setformat..... set the display format in the group spreadsheet for the specified series (p. 498).
setindent ..... set the indentation in the group spreadsheet for the specified series (p. 502).
setjust ..... set the justification for cells in the spreadsheet view of the group object (p. 502).
settextcolor..... set custom spreadsheet text coloring for the group (p. 503).
setwidth ..... set the column width in the group spreadsheet for the specified series (p. 508).
sort ..... change display order for group spreadsheet (p. 510).

Group Graph Views

Graph creation types are discussed in detail in "Graph Creation Command Summary" on page 1267.

area..... area graph of the series in the group (p. 1269).
band ..... area band graph (p. 1272).
bar ..... single or multiple bar graph view of all series (p. 1275).
boxplot..... boxplot of each series in the group (p. 1279).
distplot..... distribution graph (p. 1283).
dot ..... dot plot graph (p. 1290).
errbar..... error bar graph view (p. 1294).
hilo ..... high-low(-open-close) chart (p. 1296).
line ..... single or multiple line graph view of all series (p. 1298).
mixed..... mixed-type graph (p. 1301).
pie ..... pie chart view (p. 1304).
qqplot ..... quantile-quantile plots (p. 1306).
scat ..... scatterplot (p. 1310).
scatmat ..... matrix of all pairwise scatter plots (p. 1315).
scatpair ..... scatterplot pairs graph (p. 1318).
seasplot..... seasonal line graph (p. 1322).
spike ..... spike graph (p. 1323).
xyarea..... XY area graph (p. 1327).
xybar ..... XY bar graph (p. 1330).
xyline..... XY line graph (p. 1333).
xyerrbar ..... XY error bar graph (p. 1332).
xypair ..... XY pairs graph (p. 1337).

Group Data Members

(i) ..... *i*-th series in the group. Simply append "(i)" to the group name (without a "."). *For use as argument to functions that take a series, not as a series object.*

Scalar Values

@comobs ..... number of observations in the current sample for which each series in the group has a non-missing value (observations in the common sample).

@count ..... number of series in the group.

@minobs ..... number of non-missing observations in the current sample for the shortest series in the group.

@maxobs ..... number of non-missing observations in the current sample for the the longest series in the group.

String Values

@attr("*arg*") ..... string containing the value of the *arg* attribute, where the argument is specified as a quoted string.

@description ..... string containing the object description (if available).

@depends ..... string containing a list of the series in the current workfile on which this group depends.

@detailedtype ..... string with the object type: "GROUP".

@displayname ..... string containing the Group's display name. If the Group has no display name set, the name is returned.

@members ..... string containing a space delimited list of the names of the series contained in the Group.

@name ..... string containing the Group's name.

@remarks ..... string containing the Group's remarks (if available).

@seriesname(i) ..... string containing the name of the *i*-th series in the group.

@type ..... string with the object type: "GROUP".

@updatetime ..... string representation of the time and date at which the Group was last updated.

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 nsers=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 139 of *User's Guide I* for additional discussion of groups. "Cross-section Identifiers" on page 1249 of *User's Guide II* discusses pool identifiers.

See also `Group::drop` (p. 471).

cause	Group Views

Granger causality test.

Performs pairwise Granger causality tests between (all possible) pairs of the group of series. If performed on series in a panel workfile, you may optionally choose to perform the Dumitrescu-Hurlin (2012) version of the test.

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.

Panel Options

dh	Perform the Dumitrescu-Hurlin test.

General Options:

prompt	Force the dialog to appear from within a program.
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, dh)
```

print the result of six pairwise Dumitrescu-Hurlin causality tests for the three series in the MACRO group in a panel workfile.

Cross-references

See "Granger Causality" on page 712 of *User's Guide I* for a discussion of Granger's approach to testing hypotheses about causality. See "Panel Causality Testing" on page 1447 of *User's Guide II* for discussion of testing in panel settings.

cdfplot	Group Views

Empirical distribution plot.

The `cdfplot` command is no longer supported. See `distplot` (p. 1283).

clearcontents	Group Procs

Clear (i.e., replace with NAs) a contiguous block of observations in a group.

Syntax

group_name.**clearcontents**(*start_point*, *col_range*) *n*

where *start_point* specifies the first of *n* observations to clear. If *n* is negative, *start_point* specifies the last of |*n*| observations to clear. For dated workfiles, *start_point* should be entered as a date. For panels and undated workfiles, *start_point* should be an observation number.

The *col_range* option is used to specify the columns to be cleared in the group. 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[:]*last_col*	Colon-delimited range of columns (from low to high, *e.g.*, "3:5"). Apply to all series corresponding to the column range.
first_series[:]*last_series*	Colon-delimited range of columns (from low to high, *e.g.*, "series01:series05") specified by series names. Apply to all series corresponding to the column range.

Examples

```
grp.clearcontents(1952Q2, 1) 10
```

clears 10 observations starting at 1952 quarter 2 in the first series in the group.

```
grp.clearcontents(10, gdp) -5
```

clears 5 observations ending at observation number 10 in the series GDP in the group.

```
grp.clearcontents(1990M2, @all) 8
```

clears 8 observations starting at February of 1990 in every series in the group.

clearhist	Group Procs

Clear the contents of the history attribute for group objects.

Removes the group's history attribute, as shown in the label view of the graph.

Syntax

group_name.**clearhist**

Examples

```
g1.clearhist
g1.label
```

The first line removes the history from the group G1, and the second line displays the label view of G1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Group::label` (p. 478).

clearremarks	Group Procs

Clear the contents of the remarks attribute.

Removes the group's remarks attribute, as shown in the label view of the group.

Syntax

group_name.**clearremarks**

Examples

```
g1.clearremarks
g1.label
```

The first line removes the remarks from the group G1, and the second line displays the label view of G1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Group::label` (p. 478).

coint	Group Views

Perform either (1) Johansen's system cointegration test, (2) Engle-Granger or Phillips-Ouliaris single equation cointegration testing, or (3) Pedroni, Kao, or Fisher panel cointegration testing for the series in the group.

There are three forms for the coint command depending on which form of the test you wish to perform.

Johansen Cointegration Test Syntax

group_name.**coint***(options) [lag_spec] [@ x1 x2 x3 ...] [@***exogsr** *sx1 sx2 sx3 ...] [@***exoglr** *lx1 lx2 lx3 ...] [@***exogboth** *bx1 bx2 bx3 ...]*

uses the `coint` keyword followed by *options*, and optionally,

- a *lag_spec* consisting of one or more pairs of lag intervals, where the lag orders are for the differences in the error correction representation of the VEC, not the levels representation of the VAR.
- an "@"-sign or "@exogsr" followed by a list of exogenous variables in the short-run equation only
- "@exoglr" followed by a list of exogenous variables in the long-run relation only
- "@exogboth" followed by a list of exogenous variables in both the long-run relation and the short-run equations

(This type of cointegration testing may be used in a non-panel workfile except when performing Fisher combined testing using the Johansen framework.)

Note that the output for Johansen cointegration tests displays p-values for the rank test statistics. These p-values are computed using the response surface coefficients as estimated in MacKinnon, Haug, and Michelis (1999). The 0.05 critical values are also 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 for the Johansen Test

Deterministic Trend Option

There are 8 different deterministic trend assumptions that you may specify using the "determ = *arg*" option.

These cases correspond to whether the intercept ("c") and the trend ("t") are either

- not included ("n")
- in the long-run cointegrating relation only ("l")
- in the short-run equation only ("s")
- in both the long and short-run equations ("b")

The values of *arg* are text shortcuts formed by joining a text shortcut for the intercept specification with a text shortcut for the trend specification.

The individual intercept and trend specifications are formed by joining the "c" and the "t" with the appropriate letter describing inclusion in the long and short-run equations.

For example,

- "cb" indicates that the constant is in both the long and short-run equation
- "tl" indicates that the trend is in the long-run cointegrating equation only

so that

- "cbtl" indicates that the constant is in both the long and short-run and the trend is in the long-run only

Using this convention (along with a special "none" option), we may easily describe options arguments for all 8 deterministic cases:

cntn, none	Case 1: No deterministic terms. Corresponding VAR model has no deterministic terms.
cltn	Case 2: Restricted constant. Constant only in the cointegrating relations. Corresponding VAR has a constant.
cbtn *(default)*	Case 3 (JHJ): Unrestricted constant Constant included both in the short-run equation and (artificially) in the cointegrating relations via orthogonalization. Corresponding VAR has a constant and trend.
cstn	Case 3: Unrestricted constant Constant only in the short-run equation. Corresponding VAR has a trend.

cbtl	Case 4 (JHJ): Unrestricted constant and restricted trend Constant included both in the short-run equation and (artificially) in the cointegrating relations via orthogonalization, and trend included only in the cointegrating relations. Corresponding VAR has a constant and trend.
cstl	Case 4: Unrestricted constant and restricted trend Constant only in the short-run equation, and trend only in the cointegrating relation. Corresponding VAR has a trend.
cbtb	Case 5 (JHJ): Unrestricted constant and trend Constant and trend both included in the short-run equation and (artificially) in the cointegrating relations via orthogonalization. Corresponding VAR has a constant, linear, and quadratic trend.
csts	Case 5: Unrestricted constant and trend Constant and trend both included in the short-run equation. Corresponding VAR has a linear and quadratic trend.

or you may use the "determsummary" option to compute tests under all deterministic assumptions.

Other Johansen Options

determsummary	Summarize all deterministic trend cases.
restrict	Impose restrictions as specified by the "restspec = " option.
restspec = "*spec*"	Define the restricted VEC specification where *spec* is a space a space delimited list of VEC coefficient restrictions.
m = *integer*, maxit = *integer*	Maximum number of iterations for restricted estimation (only valid if you choose the restrict option).
c = *scalar*, cvg = *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".
prompt	Force the dialog to appear from within a program.
p	Print results.

Single Equation Test Syntax

group_name.**coint**(**method** = *arg, options*) *[*@**determ** *determ_spec] [*@**regdeterm** *regdeterm_spec]*

where

method = *arg*	Test method: Engle-Granger residual test ("eg"), Phillips-Ouliaris residual test ("po").

Cointegrating equation specifications that include a constant, linear, or quadratic trends, should use the "trend = " option to specify those terms. If any of those terms are in the stochastic regressors equations but not in the cointegrating equation, they should be specified using the "regtrend = " option.

Deterministic trend regressors that are not covered by the list above may be specified using the keywords **@determ** and **@regdeterm**. To specify deterministic trend regressors that enter into the regressor and cointegrating equations, you should add the keyword **@determ** followed by the list of trend regressors. To specify deterministic trends that enter in the regres-

sor equations but not the cointegrating equation, you should include the keyword **@regdeterm** followed by the list of trend regressors.

Note that the p-values for the test statistics are based on simulations, and do not account for any user-specified deterministic regressors.

This type of cointegration testing may be used in a non-panel workfile. The remaining options for the single equation cointegration tests are outlined below.

Options for Single Equation Tests

Options for the Engle-Granger Test

The following options determine the specification of the Engle-Granger test (Augmented Dickey-Fuller) equation and the calculation of the variances used in the test statistic.

trend = *arg* *(default* = "const"*)*	Specification for the powers of trend to include in the cointegrating equation: None ("none"), Constant ("const"), Linear trend ("linear"), Quadratic trend ("quadratic"). Note that the specification implies all trends up to the specified order so that choosing a quadratic trend instructs EViews to include a constant and a linear trend term along with the quadratic.
regtrend = *arg* *(default* = "none"*)*	Additional trends to include in the regressor equations (but not the cointegrating equation): None ("none"), Constant ("const"), Linear trend ("linear"), Quadratic trend ("quadratic"). Only trend orders higher than those specified by "trend = " will be considered. Note that the specification implies all trends up to the specified order so that choosing a quadratic trend instructs EViews to include a constant and a linear trend term along with the quadratic.
lag = *arg* *(default* = "a"*)*	Method of selecting the 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).
lagtype = *arg* *(default* = "sic"*)*	Information criterion or method 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), "tstat" (t-statistic).
maxlag = *integer*	Maximum lag length to consider when performing automatic lag-length selection *default* = $\text{int}(\min((T-k)/3, 12) \cdot (T/100)^{1/4})$ where k is the number of coefficients in the cointegrating equation. Applicable when "lag = a".

lagpval = *number* *(default* = 0.10*)*	Probability threshold to use when performing automatic lag-length selection using a *t*-test criterion. Applicable when both "lag = a" and "lagtype = tstat".
nodf	Do not degree-of-freedom correct estimates of the variances.
prompt	Force the dialog to appear from within a program.
p	Print results.

Options for the Phillips-Ouliaris Test

The following options control the computation of the symmetric and one-sided long-run variances in the Phillips-Ouliaris test.

Basic Options

trend = *arg* *(default* = "const"*)*	Specification for the powers of trend to include in the cointegrating equation: None ("none"), Constant ("const"), Linear trend ("linear"), Quadratic trend ("quadratic"). Note that the specification implies all trends up to the specified order so that choosing a quadratic trend instructs EViews to include a constant and a linear trend term along with the quadratic.
regtrend = *arg* *(default* = "none"*)*	Additional trends to include in the regressor equations (but not the cointegrating equation): None ("none"), Constant ("const"), Linear trend ("linear"), Quadratic trend ("quadratic"). Only trend orders higher than those specified by "trend = " will be considered. Note that the specification implies all trends up to the specified order so that choosing a quadratic trend instructs EViews to include a constant and a linear trend term along with the quadratic.
nodf	Do not degree-of-freedom correct the coefficient covariance estimate.
prompt	Force the dialog to appear from within a program.
p	Print results.

HAC Whitening Options

lag = *arg (default* = 0*)*	Lag specification: *integer* (user-specified lag value), "a" (automatic selection).
infosel = *arg* *(default* = "aic"*)*	Information criterion for automatic selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn) (if "lag = a").

maxlag = *integer*	Maximum lag-length for automatic selection *(optional)* (if "lag = a"). The default is an observation-based maximum.

HAC Kernel Options

kern = *arg* *(default* = "bart"*)*	Kernel shape: "none" (no kernel), "bart" (Bartlett, *default*), "bohman" (Bohman), "daniell" (Daniel), "parzen" (Parzen), "parzriesz" (Parzen-Riesz), "parzgeo" (Parzen-Geometric), "parzcauchy" (Parzen-Cauchy), "quadspec" (Quadratic Spectral), "trunc" (Truncated), "thamm" (Tukey-Hamming), "thann" (Tukey-Hanning), "tparz" (Tukey-Parzen).
bw = *arg* *(default* = "nwfixed"*)*	Bandwidth: "fixednw" (Newey-West fixed), "andrews" (Andrews automatic), "neweywest" (Newey-West automatic), *number* (User-specified bandwidth).
nwlag = *integer*	Newey-West lag-selection parameter for use in nonparametric bandwidth selection (if "bw = neweywest").
bwoffset = *integer* *(default* = 0*)*	Apply integer offset to bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").
bwint	Use integer portion of bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").

Panel Test Syntax

group_name.**coint***(option)*

The `coint` command tests for cointegration among the series in the group. This form of the command should be used with panel structured workfiles.

Options for the Panel Tests

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.
determ = *arg*	Indicate deterministic trends as detailed above in "Options for the Johansen Test" on page 443. Applicable to Fisher tests.

Additional Options:

hac = *arg* *(default* = "bt"*)*	Method of estimating the frequency zero spectrum: "bt" (Bartlett kernel), "pr" (Parzen kernel), "qs" (Quadratic Spectral kernel). Applicable to Pedroni and Kao tests.
bw = *arg* *(default* = "nw"*)*	Method of selecting the bandwidth, where *arg* may be "nw" (Newey-West automatic variable bandwidth selection), or a number indicating a user-specified common bandwidth. Applicable to Pedroni and Kao tests.
lag = *arg*	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.
infosel = *arg* *(default* = "sic"*)*	Information criterion to use when computing automatic lag length selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn). Applicable to Pedroni and Kao tests.
maxlag = *int*	Maximum lag length to consider when performing automatic lag length selection, where *int* is an integer. The default is $$\text{int}(\min(T_i/3, 12) \cdot (T_i/100)^{1/4})$$ where T_i is the length of the cross-section. Applicable to Pedroni and Kao tests.
disp = *arg* *(default* = 500*)*	Maximum number of individual results to be displayed.
prompt	Force the dialog to appear from within a program.
p	Print results.

Examples

Johansen Test

```
gr1.coint(determsummary) 1 4
```

summarizes the results of the Johansen cointegration test for the series in the group GR1 for all specifications of trend. The test equation uses lags of up to order four.

Engle-Granger Test

```
gr1.coint(method=eg)
```

performs the default Engle-Granger test on the residuals from a cointegrating equation which includes a constant. The number of lags is determined using the SIC criterion and an observation-based maximum number of lags.

```
gr1.coint(method=eg, trend=linear, lag=a, lagtype=tstat,
    lagpval=.15, maxlag=10)
```

employs a cointegrating equation that includes a constant and linear trend, and uses a sequential *t*-test starting at lag 10 with threshold probability 0.15 to determine the number of lags.

```
gr1.coint(method=eg, lag=5)
```

conducts an Engle-Granger cointegration test on the residuals from a cointegrating equation with a constant, using a fixed lag of 5.

Phillips-Ouliaris Test

```
gr1.coint(method=po)
```

performs the default Phillips-Ouliaris test on the residuals from a cointegrating equation with a constant, using a Bartlett kernel and Newey-West fixed bandwidth.

```
gr1.coint(method=po, bw=andrews, kernel=quadspec, nodf)
```

estimates the long-run covariances using a Quadratic Spectral kernel, Andrews automatic bandwidth, and no degrees-of-freedom correction.

```
gr1.coint(method=po, trend=linear, lag=1, bw=4)
```

estimates a cointegrating equation with a constant and linear trend, and performs the Phillips-Ouliaris test on the residuals by computing the long-run covariances using AR(1) prewhitening, a fixed bandwidth of 4, and the Bartlett kernel.

Panel Tests

For a panel structured workfile,

```
grp1.coint(pedroni,maxlag=3,infosel=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 Chapter 59. "Cointegration Testing," on page 1461 of *User's Guide II* for details on the various cointegration tests.

See also `Equation::coint` (p. 97) and `Var::coint` (p. 1149).

copy	Group Procs

Creates a copy of the group.

Creates either a named or unnamed copy of the group.

Syntax

group_name.**copy**

group_name.**copy** *dest_name*

Examples

```
g1.copy
```

creates an unnamed copy of the group G1.

```
g1.copy g2
```

creates G2, a copy of the group G1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

cor	Group Views

Compute covariances, correlations and other measures of association for the series in a group.

You may compute measures related to Pearson product-moment (ordinary) covariances and correlations, Spearman rank covariances, or Kendall's tau along with test statistics for evaluating whether the correlations are equal to zero.

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 `Group::cor` is equivalent to the `Group::cov` (p. 455) command with a different default setting.

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.
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

wgt = *name* (optional)	Name of series containing weights.
wgtmethod = *arg* (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.
pairwise	Compute using pairwise deletion of observations with missing cases (pairwise samples).
df	Compute covariances with a degree-of-freedom correction to account for estimated means (for centered specifications), and any partial conditioning variables.

multi = *arg* (default = "none")	Adjustment to p-values for multiple comparisons: none ("none"), Bonferroni ("bonferroni"), Dunn-Sidak ("dunn").
outfmt = *arg* (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").
prompt	Force the dialog to appear from within a program.
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 `Group::cov` (p. 455). For simple forms of the calculation, see `@cor` (p. 766), and `@cov` (p. 767) in the *Command and Programming Reference*.

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 as the first option.

Options

d = *integer (default = 0)*	Compute correlogram for specified difference of the data.
prompt	Force the dialog to appear from within a program.
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 489 and "Partial Autocorrelations (PAC)" on page 490 of *User's Guide I* for a discussion of autocorrelation and partial correlation functions, respectively.

cov	Group Views

Compute covariances, correlations and other measures of association for the series in a group.

You may compute measures related to Pearson product-moment (ordinary) covariances and correlations, Spearman rank covariances, or Kendall's tau along with test statistics for evaluating whether the correlations are equal to zero.

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).

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 `Group::cov` is equivalent to the `Group::cor` (p. 451) command with a different default setting.

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.
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

wgt = *name (optional)*	Name of series containing weights.
wgtmethod = *arg (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.
pairwise	Compute using pairwise deletion of observations with missing cases (pairwise samples).
df	Compute covariances with a degree-of-freedom correction to account for estimated means (for centered specifications), and any partial conditioning variables.

multi = *arg* *(default* = "none")	Adjustment to p-values for multiple comparisons: none ("none"), Bonferroni ("bonferroni"), Dunn-Sidak ("dunn").
outfmt = *arg* *(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").
prompt	Force the dialog to appear from within a program.
p	Print the result.

Examples

```
group grp1 height weight age
grp1.cov
```

displays a 3×3 Pearson 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 `Group::cor` (p. 451). For simple forms of the calculation, see `@cor` (p. 766), and `@cov` (p. 767) in the *Command and Programming Reference*.

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:

prompt	Force the dialog to appear from within a program.
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 706 of *User's Guide I* for discussion.

ddloadtmpl	Group Proc

Loads a dated data table template for the group.

Syntax

group_name.**ddloadtmpl**(*options*) *template_name*

template_name should be the name of a previously saved dated data table template.

Options

type = *arg*	Specify which settings to apply. *type = trans* loads the column group frequency, data display, table default transformation, and table default frequency conversion settings. *type = appear* loads the table default appearance settings. This includes the table default fonts, color, header options, label options, and formats. By default both types are loaded.
series	Load series specific settings. This option is ignored if *type = trans* is used.

Examples

```
group01.ddloadtmpl template1
```

loads all table settings from the template template1 and applies them to the dated data table of group GROUP01.

```
group01.ddloadtmpl(series) template1
```

loads both table settings and series specific settings from the template.

Cross-references

See "Dated Data Table" on page 648 of *User's Guide I* for a description of dated data tables and formatting options.

See also `dtable` (p. 472) and `ddtabopts` (p. 464).

ddrowopts

Group Proc

Set row-specific options for dated date tables.

This proc sets row specific options for the group's dated data table view. To set default settings for the dated data table, use the `ddtabopts` (p. 464) proc.

Syntax

group_name.**ddrowopts**(*series, row*) *args*

You should provide integers to indicate the *series* and *row* number you wish to modify as an option to the command, followed by a list of arguments containing the display options for that row.

Arguments

transform(*trans*)	Set the transformation method for the row. *trans* can be: "l"(level), "d"(1 period diff), "yd"(year difference), "pc"(1 period % change), "pca"(1 period % chg-AR), "pcy"(year % chg), "tabdefault" (table default setting), "none" (don't apply transformation)
freqconv(*conv*)	Set the frequency conversion method for the row. *conv* can be "avgtran"(avg then transform), "tranavg"(transform then avg), "sumtran"(sum then transform), "first"(first period), "last"(last period), "tabdefault" (table default setting).
format(fmt = *new_format*, units = *new_units*, prefix = " ", suffix = " ", +/-thousand, +/-comma, +/-parens)	Assign a custom prefix/suffix to the number, add a separator (comma or point) to denote thousands, replace a comma with a decimal point, or bracket negative numbers with parenthesis: *fmt* can be: "f[.prec]"(fixed decimal), "c[.prec]"(fixed characters), "auto", "serformat"(series format). *units* can be: "N"(native), "P"(percent), "T"(thousands), "M"(millions), "B"(billions), "TR"(trillions)
customrow("*string*")	Add a custom row header containing the quoted text *string*. To use a blank row, simply leave *string* empty.
font("*name*", *size*, +/-b, +/-I, +/-s, +/-u))	Sets the font, size and style. *name* should be the name of the font, *size* should be an integer size value. You may use +b, +i, +s or +u to set bold, italic, strikeout or underline respectively. Use "tabdefault" to use the table default font setting.
textcolor(*arg*)	Sets the color of the text. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the "@RGB" or "@HEX" functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the "@HEX" function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see "Color definitions" on page 461.
rowlabel(*label*)	Sets a custom row label in place of the series name.

Color definitions

color_arg specifies the color to be employed in the arguments above. The color may be specified using predefined color names, by specifying the individual red-green-blue (RGB) com-

ponents using the special "@RGB" function, or by specifying the individual red-green-blue (RGB) components in hexadecimal using the special "@HEX" function.

The predefined colors are given by the keywords (with their RGB or HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
ltred	@rgb(255, 168, 168)	@hex(ffa8a8)
green	@rgb(0, 128, 0)	@hex(008000)
black	@rgb(0, 0, 0)	@hex(000000)
white	@rgb(255, 255, 255)	@hex(ffffff)
purple	@rgb(128, 0, 128)	@hex(800080)
orange	@rgb(255, 128, 0)	@hex(ff8000)
yellow	@rgb(255, 255, 0)	@hex(ffff00)
gray	@rgb(128, 128, 128)	@hex(808080)
ltgray	@rgb(192, 192, 192)	@hex(c0c0c0)

Examples

The following examples show the use of `dtable`, `ddtabopts` and `ddrowopts` together to customize dated display tables.

```
group cgrp cenergy cfood chealth
cgrp.dtable
cgrp.ddtabopts firstfreq(a) secfreq(none) display(year,1)
cgrp.ddrowopts(1,1) transform(l) format(fmt=f.1)
    customrow("Consumption Indicators")
cgrp.ddrowopts(1,2) transform(pc) format(fmt=f.2, parens)
    fillcolor(red)
cgrp.ddrowopts(2,2) transform(pcy) format(fmt=f.2) fillcolor(blue)
```

creates the group CGGRP from the series CENERGY, CFOOD AND CHEALTH, and displays the dated data table for that group. `ddtapopts` is used to set the first table frequency to annual and the second frequency to none, displaying one year of data per row.

The three `ddrowopts` commands set display options for CENERGY and CFOOD. For CENERGY the first row is transformed to levels, numbers are displayed to one decimal place, and row is placed above the series with the custom string "Consumption Indicators". The next command adds a red row to CENERGY with the data transformed to 1-period percent changes, rounded to two decimal places, surrounded by parentheses if negative. The last command adds a blue filled row to CFOOD containing 1-year percent changes rounded to two decimal places.

```
group fgrp houliab hounetworth houassets
fgrp.dtable
fgrp.ddtabopts firstfreq(q) secfreq(a)
fgrp.ddrowopts(1,1) freqconv(avgtran) format(units=t)
fgrp.ddrowopts(3,1) format(fmt=f.0) transform(d) textcolor(blue)
```

The `ddtabopts` command sets the table defaults to show blocks of quarterly and annual data in the dated data table. The `ddrowopts` commands change the way HOULIAB and HOUSALES are displayed in the dated data table view. HOULIAB is set to be averaged then transformed with the units set to thousands. HOUASSETS are set to zero decimal places, transformed to the first period difference and changed to a blue font color.

```
group ggpr govinv govpurchases govsav
ggpr.dtable
ggpr.ddtabopts qtrformat(qr) nalabel("NA") rowheader(+b)
ggpr.ddrowopts(1,1) transform(pca) freqconv(tabdefault)
ggpr.ddrowopts(2,1) transform(pc) customrow(" ")
ggpr.ddrowopts(3,2) transform(pcy)
```

The `ddtabopts` command sets the table defaults to show the quarter in short case roman numerals, then adds an "NA" to any missing data, it also bold the row headers. The `ddrowopts` command transforms the GOVINV display to percent change annual rate and sets the frequency conversion method to the table default. The proc also sets GOVPURCHASES to percent change, adds a blank row above the data, and adds a transformed 1-year percent change row to GOVSAV.

```
group igrp natincome persincome dispincome
igrp.dtable
igrp.ddtabopts font("arial",10) colheader(b)
igrp.ddrowopts(1,1) transform(pca) format(fmt=f.1) textcolor(red)
igrp.ddrowopts(2,1) transform(pca) format(fmt=parens)
igrp.ddrowopts(3,1) transform(pca) customstring("Disp. Income")
```

The `ddtabopts` command sets the font to Arial size 10 and bolds the column header. For the series NATINCOME the `ddrowopts` command transforms the series to percent change annual rate, sets the numerical format to one decimal place, and sets the text color to red. For PERSINCOME the command adds a parenthesis for negative numbers, and for DISPINCOME it adds a custom row above the series containing the text heading “Disp. Income”.

Cross-references

See “Dated Data Table” on page 648 of *User’s Guide I* for a description of dated data tables and formatting options.

See also `dtable` (p. 472) and `ddtabopts` (p. 464).

ddsavetmpl	Group Proc

Saves the current dated data table settings as a new template.

Syntax

group_name.**ddsavetmpl**(*options*) *template_name*

Options

overwrite	Overwrite an existing template with the same name. Without this option naming conflicts will result in an error.

Examples

```
group01.ddsavetmpl template1
```

saves the current dated data template settings of group GROUP01 as the new template template1.

Cross-references

See "Dated Data Table" on page 648 of *User's Guide I* for a description of dated data tables and formatting options.

See also `dtable` (p. 472) and `ddtabopts` (p. 464).

ddtabopts	Group Proc

Set table default options for dated data tables.

Specifies the table default options for the group's dated data table view. To set row specific options that override the defaults, use the `ddrowopts` (p. 460) proc.

Syntax

group_name.**ddtabopts** *args*

Arguments

display(*arg*, *n*)	Specify the data to display in each table row. *arg* can be "first", "last" or "year". "first" or "last" will display annual totals, plus the first, or last, *n* observations in each row. "year" will display observations for *n* years of data per row.

firstfreq(*freq*)	Sets the frequency for the first column grouping: *freq* can be “n”(native), “a”(annual), “q”(quarterly), “m”(monthly).
secfreq(*freq*)	Sets the frequency for the second column grouping: *freq* can be “none” (none), “n”(native), “a”(annual), “q”(quarterly), “m”(monthly).
nalabel(“*arg*”)	Sets the label for NA values to *arg*.
+/-displayname	Use display names as default labels.
transform(*row*, *trans*)	Set the transformation method for row *row*. *trans* can be: “l”(level), “d”(1 period diff), “yd”(year difference), “pc”(1 period % change), “pca”(1 period % chg-AR), “pcy”(year % chg).
freqconv(*row, conv*)	Set the frequency conversion method for the specified *row*. *conv* can be “avgtran” (avg then transform), “tranavg” (transform then avg), “sumtran” (sum then transform), “first” (first period), “last”(last period).
format(fmt = *new_format*, units = *new_units*, prefix = ” “, suffix = ” “, +/-thousand, +/-comma, +/-parens)	Assign a custom prefix/suffix to the number, add a separator (comma or point) to denote thousands, replace a comma with a decimal point, or bracket negative numbers with parenthesis: *fmt* can be: “f[.prec]”(fixed decimal), “c[.prec]”(fixed characters), “auto”, “serformat”(series format). *units* can be: “N”(native), “P”(percent), “T”(thousands), “M”(millions), “B”(billions), “TR”(trillions)
colheader(+/-b, +/- i)	Sets column headers to bold or italic style.
rowheader(+/-b, +/- i)	Sets row headers to bold or italic style.
fillcolor(*arg*)	Set the table row background color to that specified by *arg*. *arg* may be one of the predefined color keywords, or it may be specified using individual red-green-blue (RGB) components using the “@RGB” or “@HEX” functions. The arguments to the @RGB function are a set of three integers from 0 to 255, representing the RGB values of the color. The arguments to the “@HEX” function are a set of six characters representing the RGB values of the color in hexadecimal. Each two character set represents a red, green or blue component in the range '00' to 'FF'. For a description of the available color keywords see “Color definitions” on page 467.

altfillcolor(*colorspec*)	Set the table alternate row background color to *colorspec*. colorspec may consist of an `@rgb(r,g,b)` or `@hex` specification or it may be the name of a basic color such as “white”, “blue”, “red”, “black”, *etc.*
font(*row*, “*name*”, *size*, +/–b, +/–I, +/–s, +/–u)	Sets the font, size and style. *name* should be the quoted name of the font, *size* should be an integer size value. You may use +b, +i, +s or +u to set bold, italic, strikeout or underline respectively.
yrformat(*arg*)	Sets the date format for year date labels. *arg* may be “YYYY” (4-digit years) or “YY” (2 digit years).
qtrformat(*arg*)	Sets the date format for quarterly date labels. *arg* may be “QR” (upper-case Roman numerals), “qr” (lower-case Roman numerals), “[Q]Q” (“Q” followed by the quarter number), “Q” (quarter number), “Mon” (3 letter month abbreviation for first month in quarter), “Month” (full month name for first month in quarter).
monformat(*arg*)	Sets the date format for monthly date labels. *arg* may be “[M]mm” (“M” followed by month number), “mm” (month number), “MM” (month number with preceding zero), “[M]MM” (“M” followed by month number with preceding zero) “Mon” (3 letter month abbreviation), “Month” (full month name), “M” (upper-case first letter of month name), or “m” (lower-case first letter of month name).
qtryrformat(*arg*)	Sets the joint date format for quarter and year. Only applicable if “Display(first)” or “Display(last)” is used. *arg* may be: “YYYY[q]Q]”, “YYYY[Q]Q”, “YYYY:Q”, “YY[q]Q”, “YY[Q]Q”, “YY:Q”, “YYYY QR”, “YYYYqr”, “YYYY qr”, “YY QR”, “YYqr”, “YY qr”, “Mon YYYY”, “Mon YY”, or “Month YYYY”. See description of “yrformat” and “qtrformat” above for details.
monyrformat(*arg*)	Sets the joint date format for month and year. Only applicable if “Display(first)” or “Display(last)” is used. *arg* may be: “YYYY[m]mm]”, “YYYY[M]mm”, “YYYY[m]MM”, “YYYY:M”, “YY[m]mm”, “YY[M]mm”, “YY[m]MM”, “YY:mm”, “YY:MM”, “Mon YYYY”, “Mon YY”, “Month YYYY”, “Month YY”, “YYMon”, or “YY-Mon”. See description of “yrformat” and “monformat” above for details on each.
+/– endperiod	Use end of period date labels.

Color definitions

color_arg specifies the color to be employed in the arguments above. The color may be specified using predefined color names, by specifying the individual red-green-blue (RGB) components using the special "@RGB" function, or by specifying the individual red-green-blue (RGB) components in hexadecimal using the special "@HEX" function.

The predefined colors are given by the keywords (with their RGB and HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
ltred	@rgb(255, 168, 168)	@hex(ffa8a8)
green	@rgb(0, 128, 0)	@hex(008000)
black	@rgb(0, 0, 0)	@hex(000000)
white	@rgb(255, 255, 255)	@hex(ffffff)
purple	@rgb(128, 0, 128)	@hex(800080)
orange	@rgb(255, 128, 0)	@hex(ff8000)
yellow	@rgb(255, 255, 0)	@hex(ffff00)
gray	@rgb(128, 128, 128)	@hex(808080)
ltgray	@rgb(192, 192, 192)	@hex(c0c0c0)

Examples

```
group cgrp cenergy cfood chealth
cgrp.dtable
cgrp.ddtabopts firstfreq(a) secfreq(none) display(year,1)
```

creates the group CGRP from the series CENERGY, CFOOD AND CHEALTH, and then it displays the dated data table for that group. `ddtapopts` is used to set the first table block frequency to annual and the second frequency to none, with one year of data displayed in each row.

```
group fgrp houliab hounetworth houassets
fgrp.dtable
fgrp.ddtabopts +displayname firstfreq(q) secfreq(a) colheader(i)
    font("Calibri",10) altfillcolor(yellow) qtrformat([Q]Q)
```

The `ddtabopts` command sets the table options for the group FGRP to show the display-names of each series in place of the series names, sets the first block frequency to quarterly and the second to annual, sets the column header style to italics, changes the font to Calibri size 10, sets the alternative row color to yellow, and sets the display for quarterly data to "Q[q]".

```
group govgrp govinv govpurchases govsav
```

```
govgrp.dtable
govgrp.ddtabopts qtrformat(qr) nalabel("NA") rowheader(+b)
```

creates the group GOVGRP out of the series GOVINV, GOVPURCHASES, and GOVSAV and then the dated data table. The `ddtabopts` command is set to show the quarter in short case roman numerals, then adds an "NA" to any missing data, it also bold the row headers.

```
group hgrp starts singlestarts multistarts
hgrp.dtable
hgrp.ddtabopts firstfreq(a) secfreq(none) +displayname
    fillcolor(@rgb(205,201,201)) yrformat(YY) format(units=n,
    fmt=f.2)
```

The `ddtabopts` command sets table default options for the group HGRP, with the first column grouping frequency as annual and the second grouping to none. The table defaults will show displaynames in place of series names, will use a light gray row fill color specified by RGB. The year format is set to show only the last two digits of the year and the numerical display format is set to native with two decimal places.

```
group incgrp natincome persincome dispincome
incgrp.dtable
incgrp.ddtabopts font("arial",10) colheader(b)
```

sets the table default font to size 10 Arial and specifies bold column headers.

Cross-references

See "Dated Data Table" on page 648 of *User's Guide I* for a description of dated data tables and formatting options.

See also `dtable` (p. 472) and `ddrowopts` (p. 460).

deleteobs	Group Procs

Delete observations from a group.

Syntax

group_name.**deleteobs**(*col, row*) *[num_obs]*

where *col* is an integer specifying the column to delete, *row* is an integer specifying the first row to delete, and *num_obs* specifies the number of observations to delete from the table.

Examples

```
group1.deleteobs(2, 5) 15
```

deletes fifteen observations from the second series in the group starting at the fifth row.

display	Group Views

Display table, graph, or spool output in the group object window.

Display the contents of a table, graph, or spool in the window of the group object.

Syntax

group_name.**display** *object_name*

Examples

```
group1.display tab1
```

Display the contents of the table TAB1 in the window of the object GROUP1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

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 123 of *User's Guide I* for a discussion of labels and display names.

See also `Group::label` (p. 478).

distdata	Group Procs

Save a matrix containing distribution plot data computed from the group.

Saves the data used to display the kernel regression, nearest neighbor regression, or empirical quantile-quantile plot to the workfile.

Syntax

group_name.**distdata**(**dtype** = *dist_type, dist_options*) *matrix_name_pattern*

saves the distribution plot data specified by *dist_type* where *dist_type* must be one of the following keywords:

kernfit	Kernel regression *(default)*.
nnfit	Nearest neighbor (local) regression.
empqq	Empirical quantile-quantile plot.

The *matrix_name_pattern* is used to define a naming pattern for the output matrices; if the pattern is "NAME", the resulting matrices will be named "NAME01", "NAME02", … and so on, using the next available name.

Options

For the first two types ("kernfit" and "nnfit"), *dist_options* are any of the distribution type-specific options described in "Kernfit Options" on page 1345 and "Nnfit Options" on page 1345, respectively. The empirical quantile-quantile plot type ("empqq") takes the options described in qqplot (p. 1306) under "Empirical Options" on page 1309.

Note that the graph display specific options such as "fill," "nofill," "leg," and "noline" are not relevant for this procedure.

In addition, you may use the "mult" option to specify multiple series handling

mult = *mat_type*	Multiple series or column handling: where *mat_type* may be: "pairs" or "p" - pairs, "mat" or "m" - scatterplot matrix, "lower" or "l" - lower triangular matrix.

and the "prompt" option to force the dialog display

prompt	Force the dialog to appear from within a program.

Examples

```
group g w x y z
g.distdata(mult=first, dtype=kernel, k=e, ngrid=100) m
```

creates a group called G from the series X, Y and Z, then creates three matrices, M01, M02 and M03, where the first matrix contains the kernel fit (with an Epanechnikov kernel and 100 grid points) of W on X, the second contains the fit of W on Y, and the third matrix contains the kernel fit of Won Z.

```
g.distdata(mult=pairs, dtype=local, b=0.3, d=1, neval=100, s) n
```

creates two matrices, N1 and N2, where N1 contains the nearest neighbor fit of W on X computed using a bandwidth of 0.3 and polynomial degree of 1, 100 evaluation points and symmetric neighbors, and N2 contains the data for the nearest neighbor fit of Y on Z.

```
group g.drop z
g.distdata(mult=all, dtype=empqq, q=r) mat
```

drops Z from the group, then creates 3 matrices; MAT01, MAT02, MAT03, where MAT01 contains the empirical quantile-quantile for W and X, computed using the rankit quantile method, and MAT02 contains the qq-plot data for W and Y, and MAT03 contains the qq-plot data for X and Y.

Cross-references

For a description of distribution graphs and quantile-quantile graphs, see "Analytical Graph Types," on page 836 of *User's Guide I*.

See also qqplot (p. 1306) and "Auxiliary Spec" on page 1343.

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 139 of *User's Guide I* for additional discussion of groups.

See also Group::add (p. 438).

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 "Dated Data Table" on page 648 of *User's Guide I* for a description of dated data tables and formatting options.

See also `ddrowopts` (p. 460), `ddtabopts` (p. 464).

dups	Group Views

Duplicate observations display for observations in the group.

Syntax

group_name.**dups**(*opts*)

A duplicate is defined as two observations with the same values for all series in the group.

By default, EViews displays a summary table showing the number of duplicate groups of a given size, but you may use the options to display an alternative view.

Of particular note is that the spreadsheet and individual duplicates displays are interactive - clicking on rows in one will open the display to show the other. Thus, clicking on a duplicate in the spreadsheet view will jump to show all of the observations that share that dupli-

cate. Similarly, clicking on an observation in the shared individual duplicates view will jump to the corresponding observation in the full spreadsheet.

Options

graph	Display observation graph showing duplicates.
sheet	Display spreadsheet view of duplicates.
individ	Display first individual duplicates.

Examples

```
grp1.dups
```

displays the duplicates summary for the group GRP1.

```
grp1.dups(sheet)
```

displays a spreadsheet showing highlighted duplicates.

Cross-references

freq	Group Views

Compute frequency tables for the series in a group.

The `freq` command performs a one-way or *N*-way frequency tabulation.

- When used with a group containing a single series, `freq` performs a one-way frequency tabulation.
- When used with a group containing multiple series, `freq` produces an *N*-way frequency tabulation for all of the series in the group.

Frequencies are computed for the current sample of observations. Observations with NAs are excluded unless included by option. You may use options to control automatic binning (grouping) of numeric values and the order of the entries of the table.

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* = 1000*)*	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*	(optional) Make bins if average count per distinct value is less than the specified number.
b = *integer* *(default* = 50*)*	Maximum number of categories to bin into if performing automatic binning.
n, obs, count *(default)*	Display frequency counts.
nocount	Do not display frequency counts.
nolimit	Remove prompt warning for continuing when the total number of cells is very large.
sort = *arg* *(default* = "lohi"*)*	Sort order for entries in the frequency table: high data value to low ("hilo"), low data value to high ("lohi" – *default*), high frequency to low ("freqhilo"), low frequency to high ("freqlohi").
prompt	Force the dialog to appear from within a program.
p	Print the table.

Options for one-way tables

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.

Options for N-way tables

table *(default)*	Display in table mode.
list	Display in list mode.
rowm *(default)* / norowm	[Display / Do not display] row marginals.
colm *(default)* / nocolm	[Display / Do not display] column marginals.
tabm *(default)* / notabm	[Display / Do not display] table marginals—only for more than two series.
subm *(default)* / nosubm	[Display / Do not display] sub marginals—only for "*l*" option with more than two series.

full *(default)* / sparse	(Full/Sparse) tabulation in list display.
totpct / nototpct *(default)*	[Display / Do not display] percentages of total observations.
tabpct / notabpct *(default)*	[Display / Do not display] percentages of table observations—only for more than two series.
rowpct / norowpct *(default)*	[Display / Do not display] percentages of row total.
colpct / nocolpct *(default)*	[Display / Do not display] percentages of column total.
exp / noexp *(default)*	[Display / Do not display] expected counts under full independence.
tabexp / notabexp *(default)*	[Display / Do not display] expected counts under table independence—only for more than two series.
test *(default)* / notest	[Display / Do not display] tests of independence.

Examples

```
group g1 hrs
g1.freq(nov, noa)
```

tabulates each value (no binning) of HRS in ascending value order with counts, percentages, and cumulatives.

```
group g2 inc
g2.freq(v=200, b=100, keepna, noa)
```

tabulates INC including NAs. The values will be binned if INC has more than 200 distinct values; EViews will create at most 100 equal value-width bins. The number of bins may be less than 100.

```
group labor lwage gender race
labor.freq(v=10, norowm, nocolm)
```

displays tables of binned LWAGE against GENDER for each bin/value of RACE. The table will not contain row and column marginals.

```
labor.freq(v=10, norowm, nocolm, sort=freqhilo)
```

displays the same table with the table rows and columns ordered by values with highest frequency to lowest.

Cross-references

See "One-Way Tabulation" on page 467 and "N-Way Tabulation" on page 680 of *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.

The wildcard operator, *, may be used as part of the series list to include many series at once. The keywords AND or NOT can be used to specify certain series should not 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. 438).

```
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.

```
group g2 us_*
```

This line creates a group named G2 that contains any series whose name starts with the characters US_.

```
group g3 * not resid
```

This command makes a group, G3, containing all series in the workfile except for the resid series.

```
group g4 a* and *1
```

Makes a group named G4 containing all series whose names begin with the letter A and end with L.

```
group g5 a* b* not *1 *2
```

This line makes a group, G5, containing all series whose names begin with either letter A or B and do not end with either 1 or 2.

```
group g6 g1 and g2
```

Makes a group named G6 containing all series that are both in group G1 and group G2 (i.e. the intersection of the two groups).

Cross-references

See Chapter 12. "Groups," on page 641 of *User's Guide I* for additional discussion.

See also `Group::add` (p. 438) and `Group::drop` (p. 471).

insertobs	Group Procs

Shift the observations of the series in the group up or downwards, inserting blank observations.

Syntax

group_name.**insertobs***(startpoint, col_range) n*

Where *startpoint* specifies the first or last observation from which the observations are shifted. For dated workfiles, *startpoint* should be entered as a date. For panels and non-dated workfiles *startpoint* should be an observation number.

The *col_range* option is used to describe the columns to be shifted in the group. 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[:]*last_col*	Colon delimited range of columns (from low to high, *e.g.*, "3:5"). Apply to all series corresponding to the column range.
first_series[:]*last_series*	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.

n specifies the number of observations shifted.

Examples

```
g.insertobs(1952q2, 1) 2
```

Inserts 2 new observations beginning at observation 1952 quarter 2 into the first series in the group. The previous value associated with 1952Q2 for that series will now correspond to 1952Q4.

```
g.insertobs(10, gdp) -5
```

Inserts 5 new observations to the series GDP ending at observation number 10.

```
g.insertobs(1990m2, @all) 8
```

Inserts 8 new observations beginning at February 1990 for all series in the group.

kerfit	Group Views

Scatterplot with bivariate kernel regression fit.

The `kerfit` command is no longer supported. See `scat` (p. 1310).

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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User’s Guide I* for a discussion of labels.

See also Group::displayname (p. 469).

linefit	Group Views

Scatterplot with bivariate fit.

The linefit command is no longer supported. See scat (p. 1310).

lrcov	Group Views

Compute the symmetric, one-sided, or strict one-sided long-run covariance matrix for a group of series.

Syntax

Group View: group_name.**lrcov**(*options*)

Options

window = *arg*	Type of long-run covariance to compute: “sym” (symmetric), “lower” (lower - lags in columns), “slower” (strict lower - lags only), “upper” (upper - leads in columns), “supper” (strict upper - leads only)
noc	Do not remove means (center data).
rwgt = *arg*	Row weights.
out = *arg*	Name of output sym or matrix (optional).
panout = *arg*	Name of panel output matrix (optional).

prompt	Force the dialog to appear from within a program.
p	Print results.

Whitening Options

lag = *arg (default* = 0*)*	Lag specification: *integer* (user-specified number of lags), "a" (automatic selection).
infosel = *arg (default* = "aic"*)*	Information criterion for automatic selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn) (if "lag = a").
maxlag = *integer*	Maximum lag-length for automatic selection *(optional)* (if "lag = a"). The default is an observation-based maximum of $T^{1/3}$.

Kernel Options

kern = *arg (default* = "bart"*)*	Kernel shape: "none" (no kernel), "bart" (Bartlett, *default*), "bohman" (Bohman), "daniell" (Daniell), "parzen" (Parzen), "parzriesz" (Parzen-Riesz), "parzgeo" (Parzen-Geometric), "parzcauchy" (Parzen-Cauchy), "quadspec" (Quadratic Spectral), "trunc" (Truncated), "thamm" (Tukey-Hamming), "thann" (Tukey-Hanning), "tparz" (Tukey-Parzen), "user" (User-specified; see "kernwgt = " below).
kernwgt = *vector*	User-specified kernel weight vector (if "kern = user").
bw = *arg (default* = "nwfixed"*)*	Bandwidth: "fixednw" (Newey-West fixed), "andrews" (Andrews automatic), "neweywest" (Newey-West automatic), *number* (User-specified bandwidth).
nwlag = *integer*	Newey-West lag-selection parameter for use in nonparametric bandwidth selection (if "bw = neweywest").
bwoffset = *integer (default* = 0*)*	Apply integer offset to bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").
bwint	Use integer portion of bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").

Examples

```
grp1.lrcov(out=outsym)
```

computes the symmetric long-run covariance of the series in the group GRP1 and saves the results in the output sym matrix OUTSYM.

```
xgrp.lrcov(kern=quadspec, bw=andrews, rwgt=res)
```

computes the long-run covariance of the series in the group XGRP using the quadratic spectral kernel, Andrews automatic bandwidth, and the row-weight series RES.

```
xgrp.lrcov(kern=quadspec, lag=1, bw=andrews, rwgt=res)
```

performs the same calculation but uses VAR(1) prewhitening prior to computing the kernel estimator.

```
xgrp.lrcov(kern=none, window=upper, lag=a, infosel=aic,
    bw=andrews, rwgt=res)
```

computes parametric VAR estimates of the upper long-run covariance using an AIC based automatic bandwidth selection method.

Cross-references

See "Long-run Covariance," on page 706 of *User's Guide I*, "Panel Long-run Variances," on page 1449 of *User's Guide II*, and Appendix F. "Long-run Covariance Estimation," on page 1559 of *User's Guide II*.

See also `Series::lrvar` (p. 805).

olepush — Group Procs

Push updates to OLE linked objects in open applications.

Syntax

group_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

makepcomp — Group Procs

Save the scores from a principal components analysis of the series in a group.

Syntax

group_name.**makepcomp***(options) output_list*

where the *output_list* is a list of names identifying the saved components. 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

scale = *arg* *(default* = "normload"*)*	Diagonal matrix scaling of the loadings and the scores: normalize loadings ("normload"), normalize scores ("normscores"), symmetric weighting ("symmetric"), user-specified (*arg* = *number*).
cpnorm	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).
eigval = *vec_name*	Specify name of vector to hold the saved the eigenvalues in workfile.
eigvec = *mat_name*	Specify name of matrix to hold the save the eigenvectors in workfile.
prompt	Force the dialog to appear from within a program.

Covariance Options

cov = *arg* *(default* = "corr"*)*	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.
wgt = *name* (optional)	Name of series containing weights.
wgtmethod = *arg* (default = "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.
pairwise	Compute using pairwise deletion of observations with missing cases (pairwise samples).
partial = *arg*	Compute partial covariances conditioning on the list of series specified in *arg*.

df	Compute covariances with a degree-of-freedom correction accounting for the mean (for centered specifications) and any partial conditioning variables. 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.makepcomp 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 696 of *User's Guide I* for further discussion.

See `Group::pcomp` (p. 486) for tools to display the principal components results for the series in the group.

makesystem	Group Procs

Create system from a group.

Syntax

group_name.**makesystem***(options) [x1 x2 x3 ...] [***@**eqreg** *w1 w2 ...] [***@**inst** *z1 z2 ...] [***@**eqinst** *z3 z4 ...]*

Creates a system of equations out of the variables in the group. Each series in the group will be used as the dependent variable in an equation. The *[x1 x2 x3 ...]* list consists of regressors with common coefficients in the system. The **@eqreg** list consists of regressors with different coefficients in each equation. The list of variables that follow **@inst** are the common instruments. The list of variables that follow **@eqinst** are the equation specific instruments.

Options

name = *name*	Specify name for the system object.

ytrans = *arg*	Dependent variable transformation: none *(default)*, log ("log"), difference ("d"), difference of logs ("dlog"), one percentage change in decimal ("pch"), one-period percentage change—annualized, in percent ("pcha"), one-year percentage change in decimal ("pchy").
prompt	Force the dialog to appear from within a program.

Examples

```
grp1.makesystem(name=sys1) c x1 x2 @inst z1 z2 z3
```

creates a system named SYS1 with the series in GRP1 as the dependent variables and a common intercept and coefficients on X1 and X2, with common instruments Z1, Z2, and Z3.

```
grp1.makesystem(name=sys2) x1 @eqreg c x2 @inst z1 z2 @eqinst z3
```

creates a system named SYS2 with a common coefficient for X1 and a different intercept and coefficient for X2 for each equation. There are common intercepts Z1 and Z2, and an equation specific instrument Z3.

Cross-references

See Chapter 43. "System Estimation," on page 897 of *User's Guide II* for a discussion of system objects in EViews.

makewhiten

Group Procs

Whiten the series in the group.

Estimate a VAR(p) for the series in the group, compute the residuals, and save the results into whitened series.

Syntax

Group View: group_name.**makewhiten**(*options*) *out_specification*

where *out_specification* is either a list of names for the output series, one per series in the original group, or is a wildcard expression. Note that wildcards may not be used if the original group contains series expressions.

Options

grp = *arg*	Name of group to hold output series (optional).
lag = *arg* *(default* = 1*)*	Lag specification: *integer* (user-specified number of lags), "a" (automatic selection).
noc	Do not remove means (center data) prior to whitening.

infosel = *arg* *(default* = "aic")	Information criterion for automatic selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn).
maxlag = *integer*	Maximum lag-length for automatic selection *(optional)*. The default is an observation-based maximum of the integer portion of $T^{1/3}$.
prompt	Force the dialog to appear from within a program.

Examples

```
grp1.makewhiten(grp=wht, lag=a, infosel=sic, maxlag=10) *a
```

whitens the series in GRP1 using a VAR with auto-selected number of lags based on the SIC information criterion and a maximum of 10 lags. The resulting series are named using the wildcard expression "*a" in the named group WHT.

```
grp2.makewhiten(noc, lag=5) *a
```

whitens the series in GRP2 using a no-constant VAR and 5 lags.

Cross-references

See "Make Whitened" on page 715 of *User's Guide I* for detail.

members

Group Views

Display the members of the group.

Syntax

Group View: group_name.**members**

Examples

```
grp1.members
```

Cross-references

See "Group Members" on page 641 of *User's Guide I* for additional detail.

nnfit

Group Views

Scatterplot with bivariate nearest neighbor fit.

The `nnfit` command is no longer supported. See `scat` (p. 1310).

pcomp	Group Views

Principal components analysis.

Syntax

group_name.**pcomp***(options) [indices]*

where the elements to display in loadings, scores, and biplot graph form ("out = loadings", "out = scores" or "out = biplot") are given by the optional *indices*, (*e.g.*, "1 2 3" or "2 3"). If *indices* is not provided, the first two elements will be displayed.

Basic Options

out = *arg* *(default* = "table"*)*	Output type: eigenvector/eigenvalue table ("table"), eigenvalues graph ("graph"), loadings graph ("loadings"), scores graph ("scores"), biplot ("biplot").
eigval = *vec_name*	Specify name of vector to hold the saved the eigenvalues in workfile.
eigvec = *mat_name*	Specify name of matrix to hold the save the eigenvectors in workfile.
prompt	Force the dialog to appear from within a program.
p	Print results.

Number of Component Options

fsmethod = *arg* *(default* = "simple"*)*	Component retention method: "bn" (Bai and Ng (2002)), "ah" (Ahn and Horenstein (2013)), "simple" (simple eigenvalue methods), "user" (user-specified value). Note the following: (1) If using simple methods, the minimum eigenvalue and cumulative proportions may be specified using "minigen = " and "cproport = ". (2) If setting "fsmethod = user" to provide a user-specified value, you must specify the value with "r = ".
r = *arg (default* = 1*)*	User-specified number of components to retain (for use when "fsmethod = user").
mineigen = *arg* *(default* = 0*)*	Minimum eigenvalue to retain component (when "fsmethod = simple").
cproport = *arg* *(default* = 1.0*)*	Cumulative proportion of eigenvalue total to attain (when "fsmethod = simple").

mfmethod = *arg* *(default = "user")*	Maximum number of components used by selection methods: "schwert" (Schwert's rule, *default*), "ah" (Ahn and Horenstein's (2013) suggestion), "rootsize" ($\min(\sqrt{N}, \sqrt{T})$), "size" ($\min(N, T)$), "user" (user specified value), where N is the number of series and T is the number of observations. (1) For use with all components retention methods apart from user-specified ("fsmethod = user"). (2) If setting "mfmethod = user", you may specify the maximum number of components using "rmax = ". (3) Schwert's rule sets the maximum number of components using the rule: let $$L(k) = k(T/100)^{1/4}$$ for $k \in \{2, 4, 6, 8, 10, 12\}$ and let $T^* = \min(N, T)$; then the default maximum lag is given by $$L^* = \max_k\{L(k) \mid L(k) < T^*\}$$
n = *arg* or rmax = *arg* *(default* = all*)*	User-specified maximum number of factors to retain (for use when "mfmethod = user").
fsic = *arg* *(default* = avg*)*	Component selection criterion (when "fsmethod = bn"): "icp1" (ICP1), "icp2" (ICP2), "icp3" (ICP3), "pcp1" (PCP1), "pcp2" (PCP1), "pcp3" (ICP3), "avg" (average of all criteria ICP1 through PCP3). Component selection criterion (when "fsmethod = ah"): "er" (eigenvalue ratio), "gr" (growth ratio), "avg" (average of eigenvalue ratio and growth ratio). Component selection (when "fsmethod = simple"): "min" (minimum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "max" (maximum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "avg" (average the optimal number of factors as specified by the min and max rule, then round to the nearest integer).
demeantime	Demeans observations across time prior to component selection procedures, when "n = bn" or "n = ah".
sdizetime	Standardizes observations across time prior to component selection procedures, when "n = bn" or "n = ah".
demeancross	Demeans observations across cross-sections prior to component selection procedures, when "n = bn" or "n = ah".
sdizecross	Standardizes observations across cross-sections prior to component selection procedures, when "n = bn" or "n = ah".

Eigenvalues Plot Options

The default eigenvalue graph shows a scree plot of the ordered eigenvalues. You may use the "scree", "cproport", and "diff" option keywords to display any combination of the scree plot, cumulative eigenvalue proportions plot, or eigenvalue difference plot.

scree	Display a scree plot of the eigenvalues (if "output = graph).
diff	Display a graph of the eigenvalue differences (if "output = graph).
cproport	Display a graph of the cumulative proportions (if "output = graph).

Loadings, Scores, Biplot Graph Options

scale = *arg, (default* = "normload"*)*	Diagonal matrix scaling of the loadings and the scores: normalize loadings ("normload"), normalize scores ("normscores"), symmetric weighting ("symmetric"), user-specified (*arg* = *number*).
cpnorm	Compute the normalization for the scores so that cross-products match the target (by default, EViews chooses a normalization scale so that the moments of the scores match the target).
nocenter	Do not center the elements in the graph.
mult = *arg* *(default* = "first"*)*	Multiple graph options: first versus remainder ("first"), pairwise ("pair"), all pairs arrayed in lower triangle ("lt")
labels = *arg* *(default* = "outlier"*)*	Scores label options: identify outliers only ("outlier"), all points ("all"), none ("none").
labelprob = *arg* *(default* = 0.1*)*	Outlier label probability (if "labels = outlier").
autoscale = *arg* *(default* = 1.0*)*	Rescaling factor for auto-scaling.
userscale = *arg*	User-specified scaling.

Covariance Options

cov = *arg* *(default* = "corr"*)*	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.
wgt = *name (optional)*	Name of series containing weights.

wgtmethod = *arg* *(default = "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.
pairwise	Compute using pairwise deletion of observations with missing cases (pairwise samples).
partial = *arg*	Compute partial covariances conditioning on the list of series specified in *arg*.
df	Compute covariances with a degree-of-freedom correction accounting for the mean (for centered specifications) and any partial conditioning variables. The default behavior in these cases is to perform no adjustment (*e.g.* – compute sample covariance dividing by n rather than $n - k$).

Examples

```
group g1 x1 x2 x3 x4
freeze(tab1) g1.pcomp(eigval=v1, eigvec=m1)
```

The first line creates a group named G1 containing the four series X1, X2, X3, X4. The second line produces a view of the basic results for the principal components. 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.

```
g1.pcomp(out=graph)
g1.pcomp(out=graph, scree, cproport)
```

displays a screen plot of the eigenvalues, and a graph containing both a screen plot and a plot of the cumulative eigenvalue proportions.

```
g1.pcomp(out=loading)
```

displays a loadings plot, and

```
g1.pcomp(out=biplot, scale=symmetric, mult=lt) 1 2 3
```

displays a symmetric biplot for all three pairwise comparisons.

Cross-references

See "Principal Components" on page 685 of *User's Guide I* for further discussion.

To save principal components scores in series in the workfile, see Group::makepcomp (p. 481).

reorder	Group Procs

Reorder the members of the group.

Syntax

group_name.**reorder**(*options*) *arg*

The members of the group will be re-ordered based on the ordering given by *arg*. *arg* is either a list of variable names denoting the new ordering, or a keyword specifying a metric to use to compute the re-ordering:

@first	Order the group members based upon their first value (ignoring NAs).
@firstna	Order the group members based upon their first value (treating NAs as a value).
@last	Order the group members based upon their last value (ignoring NAs).
@lastna	Order the group members based upon their last value (treating NAs as a value).
@max	Order the group members based upon their maximum value.
@mean	Order the group members based upon their mean.
@median	Order the group members based upon their median value.
@min	Order the group members based upon their minimum value.
@obs	Order the group members based upon the number of non-NA observations.
@rand	Randomize the ordering.
@reverse	Reverse the current ordering.

Options

r	Reverse the sort order.

Examples

```
group g1 v w x y z
g1.reorder x y z w v
```

creates a group containing V W X Y Z, then reorders the group members to X Y Z W V.

```
g1.reorder @mean
```

reorders the group members based upon their mean (highest to lowest).

```
g1.reorder(r) @mean
```

reorders the group members based upon their mean (lowest to highest).

resample	Group Procs

Resample from observations in a group.

Syntax

group_name.**resample***(options) [output_spec]*

You should follow the `resample` keyword with options, and if desired, an *output_spec* containing a list of names or a wildcard expression identifying the series to hold the output.

- If a *output_spec* is a list of target series, the number of names must match the number of target series.
- If you provide a wildcard *output_spec,* the names of the original series will be used along with the wildcard to construct the output series names. You may not use a wildcard if the series in the groups are expressions.

  For example, resampling from a group containing the series X(–1) or LOG(X) will produce an error with a wildcard spec since EViews will attempt to append a suffix to the original name, producing an invalid object name.

By default, EViews uses the wildcard spec "*_b" as the *output_spec.*

Options

outsmpl = *smpl_spec*	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.
name = *group_name*	Name of group to hold created series.
permute	Draw from rows without replacement. Default is to draw with replacement.
weight = *series_name*	Name of series to be used containing values proportional to the desired row probabilities (importance sampling). The weight series must have non-missing non-negative values in the input sample, but the weights need not add up to 1, as EViews will normalize the weights. If no weights are provided, rows will be drawn with equal probability weights.

block = *integer*	Block length for each draw. Must be a positive integer. The default block length is 1.
withna *(default)*	[Draw / Do not draw] from all rows in the current sample, including those with NAs.
dropna	Do not draw from rows that contain missing values in the current workfile sample.
fixna	Excludes NAs from draws but copies rows containing missing values to the output series.
nopanel	Ignore panel structure when resampling. By default, EViews will resample within each cross-section and assign to the corresponding cross-section.
prompt	Force the dialog to appear from within a program.

- If the group name specified using "*name* = " 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 contiguous output sample. 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 overwritten if they are series objects, otherwise EViews will error. Note that only values of X_B and Y_B in the output sample (in this case the current workfile sample) will be overwritten.

```
g1.resample(weight=wt, name=G2) *_2
```

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 502 of *User's Guide I* for a discussion of the resampling procedure. For additional discussion of wildcards, see Appendix A. "Wildcards," on page 1227 of *User's Guide I*.

See also `@resample` (p. 1071) and `@permute` (p. 1037) in the *Command and Programming Reference* for sampling from matrices.

setattr	Group Procs

Set the object attribute.

Syntax

group_name.**setattr**(*attr*) *attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the "revised" attribute in the object A to the string "never", and extracts the attribute into the string object S.

Cross-references

See "Adding Custom Attributes in the Label View" on page 123 and "Adding Your Own Label Attributes" on page 70 of *User's Guide I*.

setfillcolor	Group Procs

Set the fill (background) color used in the group spreadsheet using values in the spreadsheet or in a different series.

Syntax

group_name.**setfillcolor**(*spec*) *fill_color_args*

where the required *spec* is one of the following:

type = *arg*	Type of fill coloring for spreadsheet cells: “single” (single color), “posneg” (positive-negative single threshold), “range” (single range coloring), “hilo” (high-low-median), “custom” (custom coloring).
s = arg	Colormap source: “none” (do not use a colormap and therefore do not color) or “series” (use the same colormap used by the individual series.

The first form specifies a colormap for all of the series in the group. The second form, uses individual colormaps obtained from the individual series.

General Arguments

To specify the series or expression whose values will determine the background color:

- **mapser**(*spec*)

where *spec* is a series name or expression.

To specify the minimum and maximum values where the coloring begins and ends:

- **min**(*color_arg*)
- **max**(*color_arg*)

To set the missing value (NA) background color:

- **naclr**(*color_arg*)

where *color_arg* is described below in “Color definitions” on page 497. If omitted, the color defaults to “white”.

Type-specific Arguments

There are optional type-specific arguments that correspond to each of the type choices:

Single color

To set the single background color:

clr(*color_arg*)

where *color_arg* is described below in “Color definitions” on page 497. If omitted, the color defaults to “white”.

Positive-negative single threshold

You may set the color for both the non-negative (**posclr**) and the negative (**negclr**) values

posclr(*color_arg*)
negclr(*color_arg*)

where *color_arg* is described below in "Color definitions" on page 497. If omitted, the non-negative color defaults to "white" and the negative color defaults to light-red.

Single range

To specify the range, you must specify the range endpoints:

range(*lower_val, upper_val[, range_def)*

where *range_def* specifies the range endpoints:

cright	closed on the right only
cboth	closed on both sides
cleft	closed on the left only
oboth	open on both sides

By default, the range will be open on the lower and closed on the upper threshold limits.

You should provide a color specification for the inside range color (**inclr**) and outside range color (**outclr**):

inclr(*color_arg*)
outclr(*color_arg*)

where *color_arg* is described below in "Color definitions" on page 497. If omitted, the interior color defaults to light-red, and the exterior defaults to white.

High-Low-Median

When "type = hilo" you may specify the high, low, and median coloring values:

highclr(*color_arg*)
lowclr(*color_arg*)
medianclr(*color_arg*)

where *color_arg* is described below in "Color definitions" on page 497. If omitted, the colors default to light-red.

Custom

When "type = custom" you may specify custom coloring options.

You may optionally set a base background color, and then add one or more custom threshold or range color specifications. Multiple threshold and range specifications will layer, with the first applied first, followed by the second, and so on.

Custom Base Color

To set the base color (optional):

clr(*color_arg*)

as described below in "Color definitions" on page 497. If omitted, the color defaults to "white".

Custom Threshold

To add a threshold specification:

thresh(**limit***(threshold_value, threshold_spec)*, **lowclr***(below_arg)*, **highclr***(above_arg)*, *threshold_name]*)

where *threshold_spec* is one of

cright	closed on the right
cleft	closed on the left

and the *below_arg* and *above_arg* are one of

color_arg	solid color specification
@grad(*color_arg*)	gradient using color specification
@trans	transparent

and *color_arg* are as described below in "Color definitions" on page 497. If omitted, the color defaults to "white".

The optional *threshold_name* argument may be used to attach a name to the corresponding definition.

Custom Range

To add a range specification:

range(**limit***(low_value, high_value, range_spec)*, **inclr***(inside_arg)*, **outclr***(outside_arg)[*, *range_name]*)

where *range_spec* is one of

cright	closed on the right only
cboth	closed on both sides
cleft	closed on the left only
oboth	open on both sides

inside_arg is one of

color_arg	solid color specification
@grad(*color_arg1*, *color_arg2*)	gradient using color specification, where *color_arg1 and color_arg2* are the low and high colors, res*pectively*.
@trans	transparent

outside_arg is one of

color_arg	solid color specification
@grad(*color_arg*)	gradient using color specification
@trans	transparent

color_arg1 and *color_arg2* are as described below in "Color definitions" on page 497.

The optional *range_name* argument may be used to attach a name to the corresponding definition.

Color definitions

color_arg specifies the color to be employed in the arguments above. The color may be specified using predefined color names, by specifying the individual red-green-blue (RGB) components using the special "@RGB" function, or by specifying the individual red-green-blue (RGB) components in hexadecimal using the special "@HEX" function.

The predefined colors are given by the keywords (with their RGB and HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
green	@rgb(0, 128, 0)	@hex(ffa8a8)
black	@rgb(0, 0, 0)	@hex(008000)
white	@rgb(255, 255, 255)	@hex(000000)
purple	@rgb(128, 0, 128)	@hex(ffffff)
orange	@rgb(255, 128, 0)	@hex(800080)
yellow	@rgb(255, 255, 0)	@hex(ff8000)
gray	@rgb(128, 128, 128)	@hex(ffff00)
ltgray	@rgb(192, 192, 192)	@hex(808080)

Examples

To set a gray background color for all cells in the spreadsheet, you may use:

```
grp1.setfillcolor(type=single) clr(gray)
```

To set a background color for negative values, you may use

```
grp1.setfillcolor(type=posneg) mapser(ser1)
```

which sets the background sheet fill color to white for non-negative values and light red for negative values of SER1.

Similarly,

```
grp1.setfillcolor(type=posneg) mapser(ser1) posclr(@rgb(10, 20,
    30)) negclr(purple)
```

sets the background sheet fill color to @rgb(10, 20, 30) for non-negative values and purple for negative values of SER1.

Range coloring may be specified using the "type = range" option. The command

```
grp1.setfillcolor(type=range) mapser(ser1) clr(ltgray) range(10,
    20, cleft) inclr(@rgb(128, 0, 128)) outclr(ltred) naclr(green)
```

sets the background fill to @rgb(128, 0, 128) for values between 10 and 20, light-red to values outside of the range 10 to 20, and green, for missing values.

Custom coloring allows you to construct more complex background filling:

```
grp1.setfillcolor(type=custom) mapser(ser1) clr(@rgb(10, 0, 0))
    range(limit(-10, 10, oboth), inclr(green), outclr(white)))
    thresh(limit(-1, oleft), highclr(grey), lowclr(@trans))
```

Cross-references

See "Value-Based Text and Fill Coloring" on page 186 of *User's Guide I*.

See also `Group::settextcolor` (p. 503).

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[:]*last_col*	Colon delimited range of columns (from low to high, *e.g.*, "3:5"). Apply to all series corresponding to the column range.
first_series[:]*last_series*	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
c*[.precision]*	fixed characters
e*[.precision]*	scientific/float
p*[.precision]*	percentage
r*[.precision]*	fraction

To specify a format that groups digits into thousands using a comma separator, place a "t" after the format character. For example, to obtain a fixed number of decimal places with commas used to separate thousands, use "ft*[.precision]*".

To use the period character to separate thousands and commas to denote decimal places, use ".." (two periods) when specifying the precision. For example, 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)"), 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 106 in the *Command and Programming Reference*). 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 period 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. 508), `Group::setindent` (p. 502) and `Group::setjust` (p. 502) 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 indentation setttings are taken from the Global Defaults for spreadsheet views ("Spreadsheet Data Display" on page 1019 of *User's Guide I*) at the time the spreadsheet was created.

The *col_range* option is used to describe the columns to be updated. See `Group::setformat` (p. 498) 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. 508) and `Group::setjust` (p. 502) for details on setting spreadsheet widths and justification.

setjust	Group Procs

Set the justification for cells in the spreadsheet view of the group object.

Syntax

group_name.**setjust**(*col_range*) *format_arg*

where *format_arg* may consist of the following:

top / middle / bottom	Vertical justification setting.
auto / left / center / right	Horizontal justification setting. Strings are left-justified and numbers are right-justified under “auto”.

The *col_range* option is used to specify the columns to be justified in the group. See `Group::setformat` (p. 498) for syntax.

Default display settings can be set in General Options; see “Spreadsheet Data Display” on page 1019 of *User’s Guide I*.

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 GROUP1.

```
group2.setjust(@all) right
```

sets the format for all series in the group GROUP2.

Cross-references

See `Group::setwidth` (p. 508) and `Group::setindent` (p. 502) for details on setting spreadsheet widths and indentation.

settextcolor

Group Procs

Set the text color used in the group spreadsheet using values in the spreadsheet or in a different series.

Syntax

group_name.**settextcolor**(*spec*) *fill_color_args*

where the required *spec* is one of the following:

type = *arg*	Type of fill coloring for spreadsheet cells: “single” (single color), “posneg” (positive-negative single threshold), “range” (single range coloring), “hilo” (high-low-median), “custom” (custom coloring).
s = arg	Colormap source: “none” (do not use a colormap and therefore do not color) or “series” (use the same colormap used by the individual series.

The first form specifies a colormap for all of the series in the group. The second form, uses individual colormaps obtained from the individual series.

General Arguments

To specify the series or expression whose values will determine the background color:

- **mapser**(*spec*)

where *spec* is a series name or expression.

To specify the minimum and maximum values where the coloring begins and ends:

- **min**(*color_arg*)
- **max**(*color_arg*)

To set the missing value (NA) background color:

- **naclr**(*color_arg*)

where *color_arg* is described below in "Color definitions" on page 507. If omitted, the color defaults to "black".

Type-specific Arguments

There are optional type-specific arguments that correspond to each of the type choices:

Single color

To set the single text color:

clr(*color_arg*)

where *color_arg* is described below in "Color definitions" on page 507. If omitted, the color defaults to "black".

Positive-negative single threshold

You may set the color for both the non-negative (**posclr**) and the negative (**negclr**) values

posclr(*color_arg*)
negclr(*color_arg*)

where *color_arg* is described below in "Color definitions" on page 507. If omitted, the non-negative color defaults to "black" and the negative color defaults to "red".

Single range

To specify the range, you must specify the range endpoints:

range(*lower_val, upper_val[, range_def)*

where *range_def* specifies the range endpoints:

cright	closed on the right only
cboth	closed on both sides
cleft	closed on the left only
oboth	open on both sides

By default, the range will be open on the lower and closed on the upper threshold limits.

You should provide a color specification for the inside range color (**inclr**) and outside range color (**outclr**):

inclr(*color_arg*)
outclr(*color_arg*)

where *color_arg* is described below in "Color definitions" on page 507. If omitted, the interior color defaults to light-red, and the exterior defaults to white.

High-Low-Median

When "type = hilo" you may specify the high, low, and median coloring values:

highclr(*color_arg*)
lowclr(*color_arg*)
medianclr(*color_arg*)

where *color_arg* is described below in "Color definitions" on page 507. If omitted, the colors default to light-red.

Custom

When "type = custom" you may specify custom coloring options.

You may optionally set a base text color, and then add one or more custom threshold or range color specifications. Multiple threshold and range specifications will layer, with the first applied first, followed by the second, and so on.

Custom Base Color

To set the base color (optional):

clr(*color_arg*)

as described below in "Color definitions" on page 507. If omitted, the color defaults to "white".

Custom Threshold

To add a threshold specification:

thresh(**limit***(threshold_value, threshold_spec)*, **lowclr***(below_arg),* **highclr***(above_arg), threshold_name]*)

where *threshold_spec* is one of

cright	closed on the right
cleft	closed on the left

and the *below_arg* and *above_arg* are one of

color_arg	solid color specification
@grad(*color_arg*)	gradient using color specification
@trans	transparent

and *color_arg* are as described below in "Color definitions" on page 507. If omitted, the color defaults to "white".

The optional *threshold_name* argument may be used to attach a name to the corresponding definition.

Custom Range

To add a range specification:

range(**limit***(low_value, high_value, range_spec),* **inclr***(inside_arg),* **outclr***(outside_arg)[, range_name]*)

where *range_spec* is one of

cright	closed on the right only
cboth	closed on both sides
cleft	closed on the left only
oboth	open on both sides

inside_arg is one of

color_arg	solid color specification
@grad(*color_arg1, color_arg2*)	gradient using color specification, where *color_arg1 and color_arg2* are the low and high colors, respectively.
@trans	transparent

outside_arg is one of

color_arg	solid color specification
@grad(*color_arg*)	gradient using color specification
@trans	transparent

color_arg1 and *color_arg2* are as described below in "Color definitions" on page 507.

The optional *range_name* argument may be used to attach a name to the corresponding definition.

Color definitions

color_arg specifies the color to be employed in the arguments above. The color may be specified using predefined color names, by specifying the individual red-green-blue (RGB) components using the special "@RGB" function, or by specifying the individual red-green-blue (RGB) components in hexadecimal using the special "@HEX" function.

The predefined colors are given by the keywords (with their RGB and HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
ltred	@rgb(255, 168, 168)	@hex(ffa8a8)
green	@rgb(0, 128, 0)	@hex(008000)
black	@rgb(0, 0, 0)	@hex(000000)
white	@rgb(255, 255, 255)	@hex(ffffff)
purple	@rgb(128, 0, 128)	@hex(800080)
orange	@rgb(255, 128, 0)	@hex(ff8000)
yellow	@rgb(255, 255, 0)	@hex(ffff00)
gray	@rgb(128, 128, 128)	@hex(808080)
ltgray	@rgb(192, 192, 192)	@hex(c0c0c0)

Examples

To set a gray text color for all cells in the spreadsheet, you may use:

```
grp1.settextcolor(type=single) clr(gray)
```

To set a text color for negative values, you may use

```
grp1.settextcolor(type=posneg) mapser(ser1)
```

which sets the text color to black for non-negative values and red for negative values of SER1.

Similarly,

```
grp1.settextcolor(type=posneg) mapser(ser1) posclr(@rgb(10, 20,
    30)) negclr(purple)
```

sets the text color to @rgb(10, 20, 30) for non-negative values and purple for negative values of SER1.

Range coloring may be specified using the "type = range" option. The command

```
grp1.settextcolor(type=range) mapser(ser1) clr(ltgray) range(10,
    20, cleft) inclr(@rgb(128, 0, 128)) outclr(ltred) naclr(green)
```

sets the text to @rgb(128, 0, 128) for values between 10 and 20, light-red to values outside of the range 10 to 20, and green, for missing values.

Custom coloring allows you to construct more complex text coloring:

```
grp1.settextcolor(type=custom) mapser(ser1) clr(@rgb(10, 0, 0))
    range(limit(-10, 10, oboth), inclr(green), outclr(white)))
    thresh(limit(-1, oleft), highclr(grey), lowclr(@trans))
```

Cross-reference

See "Value-Based Text and Fill Coloring" on page 186 of *User's Guide I*.

See also `Group::settextcolor` (p. 503).

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. 502) and `Group::setjust` (p. 502) for details on setting spreadsheet indentation and justification.

sheet	Group Views

Spreadsheet view of a group object.

Syntax

group_name.**sheet***(options)*

Options

w	Wide. In a panel this will switch to the unstacked form of the panel (dates along the side, cross-sections along the top).
t	Transpose.
a	All observations (ignore sample)
nl	Do not display labels.
tform = *arg* (*default* = "level"	Display transformed data: raw data ("level"), one period difference ("dif" or "d"), annual difference ("dify" or "dy"), one period percentage change ("pch" or "pc"), annualized one period percentage change ("pcha" or "pca"), annual percentage change ("pchy" or "pcy"), natural logarithm ("log"), one period difference of logged values ("dlog").
c	Compare view. Display the compare view of the group.
p	Print the spreadsheet view.

Examples

```
g1.sheet(p)
```

displays and prints the spreadsheet view of the group G1.

```
g1.sheet(t, tform=log)
```

shows log values of the series in G1 using the current sample in a wide spreadsheet.

```
g1.sheet(nl, tform=diff)
```

displays differenced values of the series in the group using the current sample with no labels.

```
g1.sheet(a, tform=pc)
```

displays the one period percent changes for all observations in the workfile.

Cross-references

See Chapter 5. "Basic Data Handling," on page 129 of *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 workfile 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 642 of *User's Guide II* for additional discussion.

stats	Group Views

Descriptive statistics for series in a group.

Computes and displays a table of means, medians, maximum and minimum values, standard deviations, and other descriptive statistics for each series in the group.

Syntax

group_name.**stats***(options)*

Options

i	Use individual sample for each series after removing missing values for the individual series. By default, EViews computes the statistics using a common sample after removing observations with missing values for any series in the group.
p	Print the stats table.

Examples

The commands

```
group group1 wage hrs edu
group01.stats
```

computes the descriptive statistics for each series in GROUP01 for the balanced sample obtained after deletion of observation in the current sample with missing values for any of the series Alternately,

```
group01.stats(i)
```

displays the descriptive statistics view of GROUP01 showing the statistics for each series computed using individual samples.

Cross-references

See "Descriptive Statistics" on page 667 of *User's Guide I* for a discussion of the descriptive statistics views of a group.

See also `boxplot` (p. 1279).

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.
prompt	Force the dialog to appear from within a program.
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 684 of *User's Guide I* for further discussion of these tests.

See also `Series::testby` (p. 857), `Series::teststat` (p. 858).

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.

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 *(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}.

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.

Sample Option

balance	Use balanced (across cross-sections or series) data when performing test.

Lag Difference Options

Specifies the number of lag difference terms to be included in the test equation. Applicable in "Summary", LLC, Breitung, IPS, and Fisher-ADF tests. The default setting depends on whether you choose to balance the samples across cross-sections.

If you do not include the "balance" option, the default is to perform automatic lag selection using the Schwarz criteria ("lagmethod = sic").

Alternately, if you include the "balance" option, the default setting is a common, observation-based fixed lag ("lag = *default*") where:

$$default = \begin{cases} 1 \text{ if } (T_{\min} \leq 60) \\ 2 \text{ if } (60 < T_{\min} \leq 100) \\ 4 \text{ if } (T_{\min} > 100) \end{cases} \tag{1.1}$$

lagmethod = *arg* (*default* = "sic")	Method for selecting lag lengths (number of first difference terms) to be included in the Dickey-Fuller test regressions: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn), "tstat" (Ng-Perron first backward significant *t*-statistic).
lag = *arg*	Specified lag length (number of first difference terms) to be included in the regression: *integer* (user-specified common lag length), *vector_name* (user-specific individual lag length, one row per cross-section).
maxlag = *arg*	Maximum lag length to consider when performing automatic lag length selection: *integer* (common maximum lag length), or *vector_name* (individual maximum lag length, one row per cross-section). The default setting produces individual maximum lags of, $default = \text{int}(\min(12,\ T_i/3) \cdot (T_i/100)^{1/4})$ where T_i is the length of the cross-section.
lagpval = *arg* (*default* = 0.1)	Probability value for use in the *t*-statistic automatic lag selection method (when "lagmethod = tstat").

Kernel Options

Specifies options for computing kernel estimates of the zero-frequency spectrum (long-run covariance). Applicable to "Summary", LLC, Fisher-PP, and Hadri tests.

hac = *arg* (*default* = "bt")	Method of estimating the frequency zero spectrum: "bt" (Bartlett kernel), "pr" (Parzen kernel), "qs" (Quadratic Spectral kernel),
band = *arg*, b = *arg* (*default* = "nw")	Method of selecting the bandwidth: "nw" (Newey-West automatic variable bandwidth selection), "a" (Andrews automatic selection), *number* (user-specified common bandwidth), *vector_name* (user-specified individual bandwidths, one row for each cross-section).

Other options

prompt	Force the dialog to appear from within a program.

p	Print output from the test.

Examples

The command:

```
Grp1.uroot(llc,exog=trend)
```

performs the LLC panel unit root test with exogenous individual trends and individual effects on series in GRP1.

```
Grp2.uroot(is,exog=const,maxlag=4,lagmethod=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.

```
Grp3.uroot(sum,exog=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 Testing" on page 773 of *User's Guide II* for discussion of standard unit root tests performed on a single series, and "Cross-sectionally Independent Panel Unit Root Testing" on page 811 and "Cross-sectionally Dependent Panel Unit Root Tests" on page 822 of *User's Guide II* for discussion of unit roots tests performed on panel structured workfiles, groups of series, or pooled data.

See also `Group::uroot2` (p. 515), `Series::uroot` (p. 863), `Series::uroot2` (p. 868), `Pool::uroot` (p. 689), `Pool::uroot2` (p. 692).

uroot2	Group Views

Compute dependent (second generation) panel unit root tests on a group of series.

Syntax

group_name.**uroot2***(options)*

where *group_name* is the name of a group object.

Options

General Options

type = *arg* *(default* = "panic"*)*	Type of unit root test: PANIC - Bai and Ng (2004) ("panic"), CIPS - Pesaran (2007) ("cips"). Note: (1) when performing PANIC testing, factor selection, MQ, ADF lag selection, VAR lag selection (possibly), long-run variance (possibly), and p-value simulation options are relevant. (2) when perform CIPS testing, ADF lag selection options are relevant.
exog = *arg* *(default* = "constant"*)*	Exogenous deterministic variables to include for each cross-section: "none" (no deterministic variables), "constant" (only a constant), "trend" (both a constant and trend).
prompt	Force the dialog to appear from within a program.
p	Print results.

ADF Lag Selection Options

adflagmethod = *arg* *(default* = "sic"*)*	Method for selecting lag length (number of first difference terms) to be included in the Dickey-Fuller test regression or number of lags in the AR spectral density estimator: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn), "msaic" (Modified Akaike), "msic" (Modified Schwarz), "mhqc" (Modified Hannan-Quinn), "tstat" (Ng-Perron first backward significant t-statistic).
adflag = *integer*	Use-specified fixed lag.
adfmaxlag = *integer*	Maximum lag length to consider when performing automatic lag length selection. Note: default is Schwert's rule: let $$L(k) = k(T/100)^{1/4}$$ for $k \in \{2, 4, 6, 8, 10, 12\}$ and let $T^* = \min(N, T)$; then the default maximum lag is given by $$L^* = \max_k \{L(k) \mid L(k) < T^*\}$$
adflagpval = *arg* *(default* = 0.1*)*	Probability value for use in the t-statistic automatic lag selection method ("lagmethod = tstat").

PANIC Number of Factor Selection Options

fsmethod = *arg* *(default* = "bn"*)*	Factor retention selection method: "bn" (Bai and Ng (2002)), "ah" (Ahn and Horenstein (2013)), "simple" (simple eigenvalue methods), "user" (user-specified value). Note the following: (1) If using simple methods, the minimum eigenvalue and cumulative proportions may be specified using "minigen = " and "cproport = ". (2) If setting "fsmethod = user" to provide a user-specified value, you must specify the value with "r = ".
r = *arg* *(default* = 1*)*	User-specified number of factors to retain (for use when "fsmethod = user").
mineigen = *arg* *(default* = 0*)*	Minimum eigenvalue to retain factor (when "fsmethod = simple").
cproport = *arg* *(default* = 1.0*)*	Cumulative proportion of eigenvalue total to attain (when "fsmethod = simple").
mfmethod = *arg*	Maximum number of factors used by selection methods: "schwert" (Schwert's rule, *default*), "ah" (Ahn and Horenstein's (2013) suggestion), "rootsize" ($\min(\sqrt{N}, \sqrt{T})$), "size" ($\min(N, T)$), "user" (user specified value). (1) For use with all factor retention methods apart from user-specified ("fsmethod = user"). (2) If setting "mfmethod = user", you may specify the maximum number of factors using "rmax = ". (3) Schwert's rule sets the maximum number of factors using the rule: let $$L(k) = k(T/100)^{1/4}$$ for $k \in \{2, 4, 6, 8, 10, 12\}$ and let $T^* = \min(N, T)$; then the default maximum lag is given by $$L^* = \max_k\{L(k) \mid L(k) < T^*\}$$
rmax = *arg* *(default* = all*)*	User-specified maximum number of factors to retain (for use when "mfmethod = user").

fsic = *arg (default* = avg)	Factor selection criterion (when "fsmethod = bn"): "icp1" (ICP1), "icp2" (ICP2), "icp3" (ICP3), "pcp1" (PCP1), "pcp2" (PCP1), "pcp3" (ICP3), "avg" (average of all criteria ICP1 through PCP3). Factor selection criterion (when "fsmethod = ah"): "er" (eigenvalue ratio), "gr" (growth ratio), "avg" (average of eigenvalue ratio and growth ratio). Factor selection criterion (when "fsmethod = simple"): "min" (minimum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "max" (maximum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "avg" (average the optimal number of factors as specified by the min and max rule, then round to the nearest integer).
demeantime	Demeans observations across time prior to component selection procedures, when "n = bn" or "n = ah".
sdizetime	Standardizes observations across time prior to component selection procedures, when "n = bn" or "n = ah".
demeancross	Demeans observations across cross-sections prior to component selection procedures, when "n = bn" or "n = ah".
sdizecross	Standardizes observations across cross-sections prior to component selection procedures, when "n = bn" or "n = ah".

PANIC VAR Lag Selection Options

For use when computing a PANIC test with MQ_f statistic.

varlagmethod = *arg* *(default* = "sic"*)*	Method for selecting lag length (number of first difference terms) to be included in the test statistic VAR: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn), "msaic" (Modified Akaike), "msic" (Modified Schwarz), "mhqc" (Modified Hannan-Quinn), "tstat" (Ng-Perron first backward significant *t*-statistic).
varlag = *integer*	Use-specified fixed lag.
varmaxlag = *integer*	Maximum lag length to consider when performing automatic lag length selection. Note: default is Schwert's rule: let $$L(k) = k(T/100)^{1/4}$$ for $k \in \{2, 4, 6, 8, 10, 12\}$ and let $T^* = \min(N, T)$; then the default maximum lag is given by $$L^* = \max_k \{L(k) \mid L(k) < T^*\}$$

PANIC Long-run Variance Options

For use when computing a PANIC test using the MQ_c statistic.

Whitening Options

lag = *arg*	Lag specification: *integer* (user-specified number of lags), "a" (automatic selection).
infosel = *arg* *(default* = "aic"*)*	Information criterion for automatic selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn) (if "lag = a").
maxlag = *integer*	Maximum lag-length for automatic selection *(optional)* (if "lag = a"). The default is an observation-based maximum of $T^{1/3}$.

Kernel Options

kern = *arg* *(default = "bart")*	Kernel shape: "none" (no kernel), "bart" (Bartlett), "bohman" (Bohman), "daniell" (Daniel), "parzen" (Parzen), "parzriesz" (Parzen-Riesz), "parzgeo" (Parzen-Geometric), "parzcauchy" (Parzen-Cauchy), "quadspec" (Quadratic Spectral), "trunc" (Truncated), "thamm" (Tukey-Hamming), "thann" (Tukey-Hanning), "tparz" (Tukey-Parzen), "user" (User-specified; see "kernwgt = " below).
kernwgt = *vector*	User-specified kernel weight vector (if "kern = user").
bw = *arg* *(default = "nwfixed")*	Bandwidth: "fixednw" (Newey-West fixed), "andrews" (Andrews automatic), "neweywest" (Newey-West automatic), *number* (User-specified bandwidth).
nwlag = *integer*	Newey-West lag-selection parameter for use in nonparametric bandwidth selection (if "bw = neweywest").
bwoffset = *integer* *(default = 0)*	Apply integer offset to bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").
bwint	Use integer portion of bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").

PANIC p-value Options

mcreps = *integer*	Number of Monte Carlo replications.
asymplen = *integer*	Asymptotic length of series.
seed = *number*	Specifies the random number generator seed
rng = *arg*	Specifies the type of random number generator. The key can be; 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").

Examples

```
grp.uroot2
```

The line above performs a PANIC unit root test on the series in the group GRP.

```
grp.uroot2(fsmethod=AH, mq=mqf, varlag=3)
```

The line above performs a PANIC unit root test using Ahn and Horenstein (2013) for factor selection determination and the MQ_f test for the number of common trends using a VAR(3) model.

```
grp.uroot2(test=cips, exog=trend, adnfosel=sic)
```

The line above performs a CIPS unit root test on the series in group GRP, with ADF testing performed on each cross-section with a constant and trend, and ADF lag selection using the Schwarz criterion.

Cross-references

See "Unit Root Testing" on page 773 of *User's Guide II* for discussion of standard unit root tests performed on a single series, and "Cross-sectionally Independent Panel Unit Root Testing" on page 811 and "Cross-sectionally Dependent Panel Unit Root Tests" on page 822 of *User's Guide II* for discussion of unit roots tests performed on panel structured workfiles, groups of series, or pooled data.

See also `Group::uroot` (p. 512), `Series::uroot` (p. 863), `Series::uroot2` (p. 868), `Pool::uroot` (p. 689), `Pool::uroot2` (p. 692).

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 the corresponding "Series" (p. 755) or "Alpha" (p. 6) objects.

Link Declaration

link.....link object declaration (p. 528).

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

display.....display table, graph, or spool in object window (p. 526).
label.....label information for the link (p. 527).

Link Procs

clearhist.....clear the contents of the history attribute (p. 524).
clearremarks.....clear the contents of the remarks attribute (p. 525).
copy.....creates a copy of the link (p. 525).
displayname.....set display name (p. 526).
linkto.....specify link object definition (p. 528).
olepush.....push updates to OLE linked objects in open applications (p. 534).
setattr.....set the value of an object attribute (p. 534).

Link Data Members

String values

@attr("*arg*").....string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@description.....string containing the description (if available).
@detailedtype.....string with the object type: "LINK".
@displayname.....string containing display name. If the Link object has no display name set, the name is returned.
@first.....string containing the date or observation number of the first non-missing observation of the Link. In a panel workfile, the first date at which any cross-section has a non-missing observation is returned.

@firstall ..... returns the same as `@first`, however in a panel workfile, the first date at which all cross-sections have a non-missing observation is returned.

@last ..... string containing the date or observation number of the last non-blank observation of the alpha. In a panel workfile, the last date at which any cross-section has a non-missing observation is returned.

@lastall..... returns the same as `@last`, however in a panel workfile, the last date at which all cross-sections have a non-missing observation is returned.

@name ..... string containing the Link's name.

@remarks ..... string containing the Link's remarks (if available).

@type ..... string with the series object type: "SERIES" or "ALPHA".

@updatetime..... string representation of the time and date at which the Link was last updated.

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.

clearhist	Link Procs

Clear the contents of the history attribute.

Removes the rowvector's history attribute, as shown in the label view of the rowvector.

Syntax

rowvector_name.**clearhist**

Examples

```
r1.clearhist
r1.label
```

The first line removes the history from the rowvector R1, and the second line displays the label view of R1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Link::label` (p. 527).

clearremarks	Link Procs

Clear the contents of the remarks attribute.

Removes the link's remarks attribute, as shown in the label view of the link.

Syntax

link_name.**clearremarks**

Examples

```
l1.clearremarks
l1.label
```

The first line removes the remarks from the link L1, and the second line displays the label view of L1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Link::label` (p. 527).

copy	Link Procs

Creates a copy of the link.

Creates either a named or unnamed copy of the link.

Syntax

link_name.**copy**

link_name.**copy** *dest_name*

Examples

```
L1.copy
```

creates an unnamed copy of the link L1.

```
L1.copy L2
```

creates L2, a copy of the link L1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

display	Link Views

Display table, graph, or spool output in the link object window.

Display the contents of a table, graph, or spool in the window of the object.

Syntax

link_name.**display** *object_name*

Examples

```
link1.display tab1
```

Display the contents of the table TAB1 in the window of the object LINK1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

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 123 of *User's Guide I* for a discussion of labels and display names.

See also `Link::label` (p. 527) and `Graph::legend` (p. 402).

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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User's Guide I* for a discussion of labels.

See also Link::displayname (p. 526).

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. 528) 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 249 of *User's Guide I*.

See also Link::linkto (p. 528) and unlink (p. 618).

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.

When performing frequency conversion (where ID series are not provided) where either of the pages are undated, EViews will perform a raw copy link, in which the first observation in the source workfile page is copied into the first observation in the destination page, the second observation in the source into the second observation in the destination, and so forth.

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.

Note that frequency conversion linking with panel structured pages offers special handling:

- If both pages are dated panel pages that are structured with a single identifier, EViews will perform frequency conversion cross-section by cross-section.
- Conversion from a dated panel page to a dated, non-panel page will first perform a mean contraction across cross-sections to obtain a single time series (by computing the means for each period), and then a frequency conversion of the resulting time series to the new frequency.
- Conversion from a dated, non-panel page to a dated panel page will first involve a frequency conversion of the single time series to the new frequency. The converted time series will be used for each cross-section in the panel page.

In all three of these cases, all of the high-to-low conversion methods are supported, but low-to-high frequency conversion only offers **Constant-match average** (repeating of the low frequency observations).

- Lastly, frequency conversion involving a panel page with more than one dimension or an undated page will default to raw data copy unless general match merge options are provided.

Options

General Match Merge Link Options

The following options are available when linking with general match merging:

smpl = *smpl_spec*	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".

c = *arg*	Set the match merge contraction or the frequency conversion method. If you are linking a numeric source series by general match merge, the argument can be one of: "mean", "med" (median), "max", "min", "sum", "sumsq" (sum-of-squares), "var" (variance), "sd" (standard deviation), "skew" (skewness), "kurt" (kurtosis), "quant" (quantile, used with "quant = " option), "obs" (number of observations), "nas" (number of NA values), "first" (first observation in group), "last" (last observation in group), "unique" (single unique group value, if present), "none" (disallow contractions). If linking an alpha series, only the non-summary methods "max", "min", "obs", "nas", first", "last", "unique" and "none" are supported. For numeric links, the default contraction method is "c = mean"; for alpha links, the default is "c = unique". If you are linking by frequency conversion, you may use this argument to specify the up- or down-conversion method using the options found in `fetch` (p. 449) in the *Command and Programming Reference*. The default frequency conversion methods are taken from the series defaults.
quant = *number*	Quantile value to be used when contracting using the "c = quant" option (*e.g,* "quant = .3").
nacat	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:

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 creating a frequency conversion link, converting from *high* to *low* frequency:

c = *arg*	*High to low conversion methods removing NAs*: "a" (average of the nonmissing observations), "s" (sum of the nonmissing observations), "f" (first nonmissing observation), "l" (last nonmissing observation), "x" (maximum nonmissing observation), "m" (minimum nonmissing observation). *High to low conversion methods propagating NAs:* "an" or "na" (average, propagating missings), "sn" or "ns" (sum, propagating missings), "fn" or "nf" (first, propagating missings), "ln" or "nl" (last, propagating missings), "xn" or "nx" (maximum, propagating missings), "mn" or "nm" (minimum, propagating missings).

Note that if no 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 (BIRTHSTATE).

Now suppose that we wish to find the median income of males in our data for each possible state of birth, and then to match merge that value into our 50 observation state page.

The following commands:

```
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 249 of *User's Guide I*.

See `Link::link` (p. 528). See also `unlink` (p. 618), and `copy` (p. 411) in the *Command and Programming Reference*.

olepush	Link Procs

Push updates to OLE linked objects in open applications.

Syntax

link_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

setattr	Link Procs

Set the object attribute.

Syntax

link_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the "revised" attribute in the object A to the string "never", and extracts the attribute into the string object S.

Cross-references

See "Adding Custom Attributes in the Label View" on page 123 and "Adding Your Own Label Attributes" on page 70 of *User's Guide I*.

Logl

Likelihood object. Used for performing maximum likelihood estimation of user-specified likelihood functions.

Logl Declaration

logl .....likelihood object declaration (p. 545).

To declare a logl object, use the `logl` keyword, followed by a name to be given to the object.

Logl Method

ml.....maximum likelihood estimation (p. 547).

Logl Views

append.....add line to the specification (p. 537).
cellipse .....confidence ellipses for coefficient restrictions (p. 538).
checkderivs.....compare user supplied and numeric derivatives (p. 539).
coefcov .....coefficient covariance matrix (p. 541).
display.....display table, graph, or spool in object window (p. 542).
grads.....examine the gradients of the log likelihood (p. 543).
label .....label view of likelihood object (p. 544).
output.....table of estimation results (p. 548).
results.....estimation results (p. 549).
spec .....likelihood specification (p. 550).
wald .....Wald coefficient restriction test (p. 551).

Logl Procs

clearhist.....clear the contents of the history attribute (p. 540).
clearremarks .....clear the contents of the remarks attribute (p. 540).
copy.....creates a copy of the logl (p. 541).
displayname.....set display name (p. 542).
makegrads .....make group containing gradients of the log likelihood (p. 546).
makemodel .....make model (p. 546).
olepush.....push updates to OLE linked objects in open applications (p. 548).
setattr .....set the value of an object attribute (p. 549).
updatecoefs.....update coefficient vector(s) from likelihood (p. 550).

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 41. "The Log Likelihood (LogL) Object," on page 731 of *User's Guide II* for further discussion.

@byeqn .....evaluate specification by equation.

@byobs..... evaluate specification by observation (default).
@deriv..... specify an analytic derivative series.
@derivstep..... set parameters to control step size.
@logl..... specify the likelihood contribution series.
@param..... set starting values.
@temp..... remove temporary working series.

Logl Data Members

Scalar Values (system data)

@aic..... Akaike information criterion.
@coefcov(i,j)..... covariance of coefficients i and j.
@coefs(i)..... coefficient i.
@hq..... Hannan-Quinn information criterion.
@linecount..... scalar containing the number of lines in the Logl object.
@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.

Vectors and Matrices

@coefcov..... covariance matrix of estimated parameters.
@coefs..... coefficient vector.
@pvals..... vector containing the coefficient probability values.
@stderrs..... vector of standard errors for coefficients.
@tstats..... vector of z-statistic values for coefficients.

String values

@attr("*arg***")**..... string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@command..... full command line form of the estimation command.
@description..... string containing the Logl object's description (if available).
@detailedtype..... returns a string with the object type: "LOGL".
@displayname..... returns the Logl's display name. If the Logl has no display name set, the name is returned.
@line(i)..... returns a string containing the i-th line of the Logl object.
@method..... command line form of the estimation method.
@name..... returns the Logl's name.
@remarks..... string containing the logl object's remarks (if available).

@smpl.....sample used for Logl estimation.
@svector.....returns an Svector where each element is a line of the Logl object.
@svectornb.....same as `@svector`, with blank lines removed.
@type.....returns a string with the object type: "LOGL".
@updatetime.....returns a string representation of the time and date at which the Logl was last updated.

Logl Examples

To declare a likelihood named LL1:

```
logl ll1
```

To define a likelihood function for OLS (not a recommended way to do OLS!):

```
ll1.append @logl logl1
ll1.append res1 = y-c(1)-c(2)*x
ll1.append logl1 = log(@dnorm(res1/@sqrt(c(3))))-log(c(3))/2
```

To estimate LL1 by maximum likelihood (the "showstart" option displays the starting values):

```
ll1.ml(showstart)
```

To save the estimated covariance matrix of the parameters from LL1 as a named matrix COV1:

```
matrix cov1=ll1.@coefcov
```

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.

Cross-references

See "Specification," on page 733 of *User's Guide II* for discussion of logl specifications.

See also `Logl::spec` (p. 550).

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

ind = *arg*	Specifies whether and how to draw the individual coefficient intervals. The default is "ind = line" which plots the individual coefficient intervals as dashed lines. "ind = none" does not plot the individual intervals, while "ind = shade" plots the individual intervals as a shaded rectangle.
size = *number (default* = 0.95*)*	Set the size (level) of the confidence ellipse. You may specify more than one size by specifying a space separated list enclosed in double quotes.

dist = *arg*	Select the distribution to use for the critical value associated with the ellipse size. The default depends on estimation object and method. If the parameter estimates are least-squares based, the $F(2, n-2)$ distribution is used; if the parameter estimates are likelihood based, the $\chi^2(2)$ distribution will be employed. "dist = f" forces use of the F-distribution, while "dist = c" uses the χ^2 distribution.
prompt	Force the dialog to appear from within a program.
p	Print the graph.

Examples

The two commands:

```
log1.cellipse c(1), c(2), c(3)
log1.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).

```
log1.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 Intervals and Confidence Ellipses" on page 203 of *User's Guide II* for discussion.

See also `Logl::wald` (p. 551).

checkderivs	Logl 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 41. "The Log Likelihood (LogL) Object," on page 731 of *User's Guide II* for a general discussion of the likelihood object and the **@deriv** statement for specifying analytic derivatives.

See also `Logl::grads` (p. 543) and `Logl::makegrads` (p. 546).

clearhist	Logl Procs

Clear the contents of the history attribute for logl objects.

Removes the logl's history attribute, as shown in the label view of the logl.

Syntax

logl_name.**clearhist**

Examples

```
L1.clearhist
L1.label
```

The first line removes the history from the logl L1, and the second line displays the label view of L1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Logl::label` (p. 544).

clearremarks	Logl Procs

Clear the contents of the remarks attribute.

Removes the logl's remarks attribute, as shown in the label view of the logl.

Syntax

logl_name.**clearremarks**

Examples

```
l1.clearremarks
l1.label
```

The first line removes the remarks from the logl L1, and the second line displays the label view of L1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Logl::label` (p. 544).

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 the `@coefcov` object data member:

```
sym eqcov = ll2.@coefcov
```

Cross-references

See Chapter 41. "The Log Likelihood (LogL) Object," on page 738 of *User's Guide II* for a general discussion of the likelihood object.

See also `Coef::coef` (p. 26) and `Logl::spec` (p. 550).

copy	Logl Procs

Creates a copy of the logl.

Creates either a named or unnamed copy of the logl.

Syntax

logl_name.**copy**

logl_name.**copy** *dest_name*

Examples

```
L1.copy
```

creates an unnamed copy of the logl L1.

```
L1.copy L2
```

creates L2, a copy of the logl L1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

display	Logl Views

Display table, graph, or spool output in the logl object window.

Display the contents of a table, graph, or spool in the window of the logl object.

Syntax

logl_name.**display** *object_name*

Examples

```
logl1.display tab1
```

Display the contents of the table TAB1 in the window of the object LOGL1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

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

```
lg1.displayname Hours Worked
lg1.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 123 of *User's Guide I* for a discussion of labels and display names.

See also `Logl::label` (p. 544).

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

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:

```
ll2.grads
```

To display and print the table view:

```
ll2.grads(t, p)
```

Cross-references

See also `Logl::makegrads` (p. 546).

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.

c	Clears all text fields in the label.
d	Sets the description field to *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User's Guide I* for a discussion of labels.

See also `Logl::displayname` (p. 542).

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 41. "The Log Likelihood (LogL) Object," on page 731 of *User's Guide II* for further examples of the use of the likelihood object.

See also `Logl::append` (p. 537) for adding specification lines to an existing likelihood object, and `Logl::ml` (p. 547) 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

n = *arg*	Name of group object to contain the series.

Examples

```
ll2.grads(n=out)
```

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

```
ll2.grads(n=out) g1 g2 g3
```

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

Cross-references

See "Check Derivatives," on page 743 of *User's Guide II* for further discussion.

See also `Logl::grads` (p. 543).

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

```
ll3.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 52. "Models," on page 1177 of *User's Guide II* for a discussion of specifying and solving models in EViews.

See also `Logl::append` (p. 537), `Model::merge` (p. 628) and `Model::solve` (p. 640).

ml	Logl Method

Maximum likelihood estimation of logl models.

Syntax

logl_name.**ml***(options)*

Options

optmethod = *arg*	Optimization method: "bfgs" (BFGS); "newton" (Newton-Raphson), "opg" or "bhhh" (OPG or BHHH), "legacy" (EViews legacy). Newton-Raphson is the default method.
optstep = *arg*	Step method: "marquardt" (Marquardt); "dogleg" (Dogleg); "linesearch" (Line search). Marquardt is the default method.
cov = *arg*	Covariance method: "ordinary" (default method based on inverse of the estimated information matrix), "huber" or "white" (Huber-White sandwich methods).,
covinfo = *arg*	Information matrix method: "opg" (OPG); "hessian" (observed Hessian). (Applicable when non-legacy "optmethod = ".)
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.
prompt	Force the dialog to appear from within a program.
p	Print basic estimation results.

Examples

```
bvar.ml
```

estimates the logl object BVAR by maximum likelihood.

Cross-references

See Chapter 41. "The Log Likelihood (LogL) Object," on page 731 of *User's Guide II* for a discussion of user specified likelihood models.

olepush — Logl Procs

Push updates to OLE linked objects in open applications.

Syntax

logl_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

output — Logl Views

Display estimation output.

`output` changes the default object view to display the estimation output (equivalent to using `Logl::results` (p. 549)).

Syntax

logl_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:

```
log2.output
```

displays the estimation output for likelihood object LOG2.

Cross-references

See `Logl::results` (p. 549).

results	Logl Views

Displays the results view of an estimated likelihood object.

Syntax

logl_name.**results***(options)*

Options

p	Print the view.

Examples

```
ll1.results(p)
```

prints the estimation results from the estimated logl, LL1.

setattr	Logl Procs

Set the object attribute.

Syntax

logl_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the “revised” attribute in the object A to the string “never”, and extracts the attribute into the string object S.

Cross-references

See "Adding Custom Attributes in the Label View" on page 123 and "Adding Your Own Label Attributes" on page 70 of *User's Guide I*.

spec	Logl Views

Display the text specification view for logl objects.

Syntax

logl_name.**spec***(options)*

Options

p	Print the specification text.

Examples

```
lg1.spec
```

displays the specification of the logl object LG1.

Cross-references

See also `Logl::append` (p. 537).

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. 26).

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 210 of *User's Guide II* for a discussion of Wald tests.

See also `Logl::cellipse` (p. 538), `testdrop` (p. 607), `testadd` (p. 606).

Matrix

Matrix (two-dimensional array).

Matrix Declaration

matrix ..... declare matrix object (p. 583).

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. 558).
cov ..... covariance matrix by columns (p. 562).
display ..... display table, graph, or spool in object window (p. 565).
freq ..... n-way contingency table (p. 572).
label ..... label information for the matrix (p. 581).
pcomp ..... principal components analysis of the columns in a matrix (p. 584).
sheet ..... spreadsheet view of the matrix (p. 597).
stats ..... descriptive statistics for each column of the matrix (p. 598).
testbtw ..... tests of equality for mean, median, or variance between series in group (p. 600).

Matrix Procs

clearcollabels ..... clear the column labels in a matrix object (p. 556).
clearhist ..... clear the contents of the history attribute (p. 556).
clearremarks ..... clear the contents of the remarks attribute (p. 557).
clearrowlabels ..... clear the row labels in a matrix object (p. 557).
copy ..... creates a copy of the matrix (p. 558).
displayname ..... set display name (p. 565).
distdata ..... save a matrix containing distribution plot data computed from the matrix (p. 566).
export ..... save matrix as Excel 2007 XLSX, CSV, tab-delimited ASCII text, RTF, HTML, Enhanced Metafile, PDF, TEX, or MD file on disk (p. 568).
fill ..... fill the elements of the matrix (p. 571).
import ..... imports data from a foreign file into the matrix object (p. 575).

Matrix Graph Views

Graph creation views are discussed in detail in "Graph Creation Command Summary" on page 1267.

xybar ..... XY bar graph (p. 1330).
xyerrbar ..... XY error bar graph (p. 1332).
xyline..... XY line graph (p. 1333).
xypair ..... XY pairs graph (p. 1337).

Matrix Data Members

String values

@attr("*arg*") ..... string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@collabels..... string containing the column labels of the matrix.
@description ..... string containing the Matrix object's description (if available).
@detailedtype ..... string with the object type: "MATRIX".
@displayname..... string containing the Matrix object's display name. If the Matrix has no display name set, the name is returned.
@name ..... string containing the Matrix object's name.
@remarks ..... string containing the Matrix object's remarks (if available).
@rowlabels ..... string containing the row labels of the matrix.
@type ..... string with the object type: "MATRIX".
@updatetime..... string representation of the time and date at which the Matrix was last updated.

Scalar values

(i,j) ..... (i,j)-th element of the matrix. Simply append "(i, j)" to the matrix name (without a ".").
@cols..... number of columns in the matrix.
@rows ..... number of rows in the matrix.

Matrix values

@col(*arg*) ..... Returns the columns defined by *arg*. *arg* may be an integer, vector of integers, string, or svector of strings. Integers correspond to column numbers so that, for example, *arg* = 2 specifies the second column. Strings correspond to column labels so that *arg* = "2" specifies the first column labeled "2".
@diag ..... vector containing the diagonal elements of the matrix.
@dropboth(*arg1, arg2*)Returns the matrix with the rows defined by *arg1* and columns defined by *arg2* removed. The *args* may be integers, vectors of integers, strings, or svectors of strings. Integers correspond to row or column numbers so that, for example, *arg1* = 2 specifies the second row. Strings correspond to row or column labels so that *arg2* = "2" specifies the first column labeled "2".

@dropcol(*arg*).....Returns the matrix with the columns defined by *arg* removed. *arg* may be an integer, vector of integers, string, or svector of strings. Integers correspond to column numbers so that, for example, *arg* = 2 specifies the second column. Strings correspond to column labels so that *arg* = "2" specifies the first column labeled "2".

@droprow(*arg*) .....Returns the matrix with rows defined by *arg* removed. *arg* may be an integer, vector of integers, string, or svector of strings. Integers correspond to row numbers so that, for example, *arg* = 2 specifies the second row. Strings correspond to row labels so that *arg* = "2" specifies the first row labeled "2".

@icol(*arg*) .....Returns the indices for the columns defined by *arg* where *arg* is a string or svector of strings. The strings correspond to column labels so that *arg* = "2" specifies the first column labeled "2".

@irow(*arg*) .....Returns the indices for the rows defined by *arg* where *arg* is a string or svector of strings. The strings correspond to row labels so that *arg* = "2" specifies the first row labeled "2".

@row(*arg*) .....Returns the rows defined by *arg*. *arg* may be an integer, vector of integers, string, or svector of strings. Integers correspond to row numbers so that, for example, *arg* = 2 specifies the second row. Strings correspond to row labels so that *arg* = "2" specifies the first row labeled "2".

@sub(*arg1*, *arg2*) ..Returns the matrix with rows defined by *arg1* and columns with defined by *arg2*. The *args* may be integers, vectors of integers, strings, or svectors of strings. Integers correspond to row or column numbers so that, for example, *arg1* = 2 specifies the second row. Strings correspond to row or column labels so that *arg2* = "2" specifies the first column labeled "2".

@t .....transpose of the matrix.

Matrix Examples

The following assignment statements create and initialize matrix objects,

```
matrix copymat=results
matrix covmat1=eq1.@coefcov
matrix(5,2) count
count.fill 1,2,3,4,5,6,7,8,9,10
```

as does the equation procedure:

```
eq1.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.

clearcollabels	Matrix Procs

Clear the column labels in a matrix object.

Syntax

matrix_name.**clearcollabels**

Examples

```
mat1.clearcollabels
```

clears the custom column labels from the matrix MAT1.

Cross-references

See also `Matrix::clearrowlabels` (p. 557).

clearhist	Matrix Procs

Clear the contents of the history attribute for matrix objects.

Removes the matrix's history attribute, as shown in the label view of the matrix.

Syntax

matrix_name.**clearhist**

Examples

```
m1.clearhist
m1.label
```

The first line removes the history from the matrix M1, and the second line displays the label view of M1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Matrix::label` (p. 581).

clearremarks	Matrix Procs

Clear the contents of the remarks attribute.

Removes the matrix's remarks attribute, as shown in the label view of the matrix.

Syntax

matrix_name.**clearremarks**

Examples

```
m1.clearremarks
m1.label
```

The first line removes the remarks from the matrix M1, and the second line displays the label view of M1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Matrix::label` (p. 581).

clearrowlabels	Matrix Procs

Clear the row labels in a matrix object.

Syntax

matrix_name.**clearrowlabels**

Examples

```
mat1.clearrowlabels
```

clears the custom row labels from the matrix MAT1.

Cross-references

See also Matrix::clearcollabels (p. 556).

copy	Matrix Procs

Creates a copy of the matrix.

Creates either a named or unnamed copy of the matrix.

Syntax

matrix_name.**copy**

matrix_name.**copy** *dest_name*

Examples

```
m1.copy
```

creates an unnamed copy of the matrix M1.

```
m1.copy m2
```

creates M2, a copy of the matrix M1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

cor	Matrix Views

Compute covariances, correlations, and other measures of association for the columns in a matrix.

You may compute measures related to Pearson product-moment (ordinary) covariances and correlations, Spearman rank covariances, or Kendall's tau along with test statistics for evaluating whether the correlations are equal to zero.

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 the name of a conditioning matrix. The

columns should contain the conditioning variables, 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 `Matrix::cor` is equivalent to the `Matrix::cov` (p. 562) command with a different default setting.

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.
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

wgt = *name* (optional)	Name of vector containing weights. The number of rows of the weight vector should match the number of rows in the original matrix.
wgtmethod = *arg* (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.
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.

multi = *arg* (default = "none")	Adjustment to p-values for multiple comparisons: none ("none"), Bonferroni ("bonferroni"), Dunn-Sidak ("dunn").
outfmt = *arg* (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").
prompt	Force the dialog to appear from within a program.
p	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 `Matrix::cov` (p. 562). For simple forms of the calculation, see `@cor` (p. 766), and `@cov` (p. 767) in the *Command and Programming Reference*.

cov	Matrix Views

Compute covariances, correlations, and other measures of association for the columns in a matrix.

You may compute measures related to Pearson product-moment (ordinary) covariances and correlations, Spearman rank covariances, or Kendall's tau along with test statistics for evaluating whether the correlations are equal to zero.

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 the name of a conditioning matrix. The columns should contain the conditioning variables, 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 `Matrix::cov` is equivalent to the `Matrix::cor` (p. 558) command with a different default setting.

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.
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

wgt = *name (optional)*	Name of vector containing weights. The number of rows of the weight vector should match the number of rows in the original matrix.
wgtmethod = *arg (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.
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.
multi = *arg (default* = "none"*)*	Adjustment to p-values for multiple comparisons: none ("none"), Bonferroni ("bonferroni"), Dunn-Sidak ("dunn").
outfmt = *arg (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").
prompt	Force the dialog to appear from within a program.
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 `Matrix::cor` (p. 558). For simple forms of the calculation, see `@cor` (p. 766), and `@cov` (p. 767) in the *Command and Programming Reference*.

display	Matrix Views

Display table, graph, or spool output in the matrix object window.

Display the contents of a table, graph, or spool in the window of the matrix object.

Syntax

matrix_name.**display** *object_name*

Examples

```
matrix1.display tab1
```

Display the contents of the table TAB1 in the window of the object MATRIX1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

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 123 of *User's Guide I* for a discussion of labels and display names.

See also `Matrix::label` (p. 581).

distdata	Matrix Procs

Save a matrix containing distribution plot data computed from the matrix.

Saves the data used to display the kernel regression, nearest neighbor regression, or empirical quantile-quantile plot to the workfile.

Syntax

matrix_name.**distdata**(**dtype** = *dist_type, dist_options*) *matrix_name_pattern*

saves the distribution plot data specified by *dist_type* where *dist_type* must be one of the following keywords:

kernfit	Kernel regression *(default)*.
nnfit	Nearest neighbor (local) regression.
empqq	Empirical quantile-quantile plot.

The *matrix_name_pattern* is used to define a naming pattern for the output matrices; if the pattern is "NAME", the resulting matrices will be named "NAME01", "NAME02", … and so on, using the next available name.

Options

For the first two types ("kernfit" and "nnfit"), *dist_options* are any of the distribution type-specific options described in "Kernfit Options" on page 1345 and "Nnfit Options" on page 1345, respectively. The empirical quantile-quantile plot type ("empqq") takes the options described in `qqplot` (p. 1306) under "Empirical Options" on page 1309.

Note that the graph display specific options such as "fill," "nofill," "leg," and "noline" are not relevant for this procedure.

In addition, you may use the "mult" option to specify multiple series handling

mult = *mat_type*	Multiple series or column handling: where *mat_type* may be: “pairs” or “p” - pairs, “mat” or “m” - scatterplot matrix, “lower” or “l” - lower triangular matrix.

and the “prompt” option to force the dialog display

prompt	Force the dialog to appear from within a program.

Examples

If MAT1 is a matrix with four columns,

```
mat1.distdata(mult=first, dtype=kernel, k=e, ngrid=100) m
```

then creates three matrices, M01, M02 and M03, where the M01 contains the kernel fit (with an Epanechnikov kernel and 100 grid points) of the second column of MAT1 on the first column, M02 contains the fit of the first column on the third column, and M03 contains the kernel fit of column 1 on column 4.

```
mat1.distdata(mult=pairs, dtype=local, b=0.3, d=1, neval=100, s) n
```

creates two matrices, N1 and N2, where N1 contains the nearest neighbor fit of column 1 on column 2 computed using a bandwidth of 0.3 and polynomial degree of 1, 100 evaluation points and symmetric neighbors, and N2 contains the data for the nearest neighbor fit of column 3 on column 4.

If we extract a new matrix MAT2 containing first three columns of MAT1, the commands

```
matrix mat2 = mat1.@col(@range(1, 3))
mat2.distdata(mult=all, dtype=empqq, q=r) mat
```

creates 3 matrices; MAT01, MAT02, and MAT03, where MAT01 contains the empirical quantile-quantile for columns 1 and 2, computed using the rankit quantile method, MAT02 contains the qq-plot data for columns 1 and 3, and MAT03 contains the qq-plot data for columns 2 and 3.

Cross-references

For a description of distribution graphs and quantile-quantile graphs, see “Auxiliary Graph Types,” on page 855 of *User’s Guide I*.

See also `qqplot` (p. 1306) and “Auxiliary Spec” on page 1343.

export	Matrix Procs

Export matrix to disk as an Excel 2007 XLSX, CSV, tab-delimited ASCII text, RTF, HTML, Enhanced Metafile, LaTeX, PDF, or Markdown file.

Syntax

matrix_name.**export***(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.

The base syntax for writing Excel 2007 files is:

matrix_name.**export**(*options*) *[path\]file_name [table_description]*

where the *table_description* may contain:

- "range = *arg*", where arg is top left cell of the destination 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.

Options

t = *file_type* *(default* = "csv"*)*	Specifies the file type, where *file_type* may be one of: "excelxml" (Excel 2007 (xml)),"csv" (CSV - comma-separated), "rtf" (Rich-text format), "txt" (tab-delimited text), "html" (HTML - Hypertext Markup Language), "emf" (Enhanced Metafile), "pdf" (PDF - Portable Document Format), "tex" (LaTeX), or "md" (Markdown). Files will be saved with the ".xlsx", ".csv", ".rtf", ".txt", ".htm", ".emf", ".pdf", ".tex", or ".md" extensions, respectively.
s = *arg*	Scale size, where *arg* is from 5 to 200, representing the percentage of the original table size (only valid for HTML or RTF files).
n = *string*	Replace all cells that contain NA values with the specified string. "NA" is the default.
h / -h	Include(/do not include) column and row headers. The default is to not include the headers
prompt	Force the dialog to appear from within a program.

PDF Options

landscape	Save in landscape mode (the default is to save in portrait mode).
size = *arg* (*default* = "letter")	Page size: "letter", "legal", "a4", and "custom".
width = *number* (*default* = 8.5)	Page width in inches if "size = custom".
height = *number* (*default* = 11)	Page height in inches if "size = custom".
leftmargin = *number* *(default* = 0.5*)*	Left margin width in inches.
rightmargin = *number* *(default* = 0.5*)*	Right margin width in inches.
topmargin = *number* *(default* = 1*)*	Top margin width in inches.
bottommargin = *number (default* = 1*)*	Bottom margin width in inches.

LaTeX Options

texspec / -texspec	[Include / Do not include] the full LaTeX documentation specification in the LaTeX output. The default behavior is taken from the global default settings.

Excel Options

mode = *arg*	Specify whether to create a new file, overwrite an existing file, or update an existing file. arg may be "create" (create new file only; error on attempt to overwrite) or "update" (update an existing file, only overwriting the area specified by the range = *table_description*). If the "mode = " option is not used, EViews will create a new file, unless the file already exists in which case it will overwrite it. Note that the "mode = update" option is only available for Excel in 1) Excel versions through 2003, if Excel is installed, and 2) Excel 2007 (xml). Note: Excel does not need to be installed for Excel 2007 writing.

Excel 2007 Options

mode = *arg*	Specify whether to create a new file, overwrite an existing file, or update an existing file. arg may be "create" (create new file only; error on attempt to overwrite) or "update" (update an existing file, only overwriting the area specified by the range = *table_description*). If the "mode = " option is not used, EViews will create a new file, unless the file already exists in which case it will overwrite it. Note that the "mode = update" option is only available for Excel in 1) Excel versions through 2003, if Excel is installed, and 2) Excel 2007 (xml). Note: Excel does not need to be installed for Excel 2007 writing.
cellfmt = *arg*	Specify whether to use EViews, pre-existing, or remove cell formatting (colors, font, number formatting when possible, column widths and row heights) for the written range. *arg* may be "eviews" (replace current formatting in the file with the same cell formatting in EViews), "preserve" (leave current cell formatting already in the Excel file), or "clear" (remove current formatting and do not replace).
strlen = *arg* (default = 256)	Specify the maximum the number of characters written for cells containing text. Strings in cells which are longer the max, will be truncated.

Examples

The command:

```
matrix1.export mymatrix
```

exports the data in MATRIX1 to a CSV file named "mymatrix.CSV" in the default directory.

```
matrix1.export(h, t=csv, n="NaN") mymatrix
```

saves the contents of MATRIX1 along with the column and row headers to a CSV (comma separated value) file named "mymatrix.CSV" and writes all NA values as "NaN".

```
matrix1.export(h, t=html, s=50) mymatrix
```

exports the data in MATRIX1 along with the column and row headers to a HTML file named "mymatrix.HTM" at half of the original size.

```
matrix1.export(n=".", r=B) mymatrix
```

saves the data in the second column to a CSV file named "mymatrix.CSV", and writes all NA values as ".".

```
matrix1.export(t=excelxml, cellfmt=clear, mode=update) mymatrix
    range=Country!b5
```

writes the data in MATRIX1 to the preexisting "mymatrix.XLSX" Excel file to the "Country" sheet at cell B5, where all cell formatting is cleared.

Cross-references

See also Matrix::import (p. 575).

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 starting 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:

$$m1 = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}, \qquad m2 = \begin{bmatrix} 1 & 0 \\ 1 & 2 \end{bmatrix} \tag{1.2}$$

Cross-references

See Chapter 11. "Matrix Language," on page 279 of the *Command and Programming Reference* for a detailed discussion of vector and matrix manipulation in EViews.

freq — Matrix Views

Compute frequency tables for columns of a matrix.

The `freq` command performs a one-way or *N*-way frequency tabulation.

- When used with a matrix containing a single column, `freq` performs a one-way frequency tabulation.
- When used with a matrix containing multiple columns, `freq` produces an *N*-way frequency tabulation for all of the columns in the matrix.

Frequencies are computed for all of the rows of the matrix. Rows with NAs are dropped unless included by option. You may use options to control automatic binning (grouping) of values and the order of the entries of the table.

Syntax

matrix_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* = 1000*)*	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*	(optional) Make bins if average count per distinct value is less than the specified number.
b = *integer* *(default* = 50*)*	Maximum number of categories to bin into if performing automatic binning.
n, obs, count *(default)*	Display frequency counts.
nocount	Do not display frequency counts.
nolimit	Remove prompt warning for continuing when the total number of cells is very large.
sort = *arg* *(default* = "lohi"*)*	Sort order for entries in the frequency table: high data value to low ("hilo"), low data value to high ("lohi" – *default*), high frequency to low ("freqhilo"), low frequency to high ("freqlohi").
prompt	Force the dialog to appear from within a program.
p	Print the table.

Options for one-way tables

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.

Options for N-way tables

table *(default)*	Display in table mode.
list	Display in list mode.
rowm *(default)* / norowm	[Display / Do not display] row marginals.

colm *(default)* / nocolm	[Display / Do not display] column marginals.
tabm *(default)* / notabm	[Display / Do not display] table marginals—only for more than two series.
subm *(default)* / nosubm	[Display / Do not display] sub marginals—only for "*l*" option with more than two series.
full *(default)* / sparse	(Full/Sparse) tabulation in list display.
totpct / nototpct *(default)*	[Display / Do not display] percentages of total observations.
tabpct / notabpct *(default)*	[Display / Do not display] percentages of table observations—only for more than two series.
rowpct / norowpct *(default)*	[Display / Do not display] percentages of row total.
colpct / nocolpct *(default)*	[Display / Do not display] percentages of column total.
exp / noexp *(default)*	[Display / Do not display] expected counts under full independence.
tabexp / notabexp *(default)*	[Display / Do not display] expected counts under table independence—only for more than two series.
test *(default)* / notest	[Display / Do not display] tests of independence.

Examples

```
matrix x(50, 4)
rndint(x, 10)
x.freq(nov, noa)
```

tabulates each value (no binning) of HRS in ascending order with counts, percentages, and cumulatives.

```
x.freq(v=200, b=100, keepna, noa)
```

tabulates X including NAs. The values will be binned if INC has more than 200 distinct values; EViews will create at most 100 equal value-width bins. The number of bins may be less than 100.

```
x.freq(v=10, norowm, nocolm)
```

displays tables of binned pairs of the first two columns of X for each bin/value of the remaining columns. The table will not contain row and column marginals.

```
x.freq(v=10, norowm, nocolm, sort=freqhilo)
```

displays the same table with the table rows and columns ordered from values with highest frequency to lowest.

Cross-references

See "One-Way Tabulation" on page 467 and "N-Way Tabulation" on page 680 of *User's Guide I* for a discussion of frequency tables.

import	Matrix Procs

Imports data from a foreign file into the matrix object.

Syntax

matrix_name.**import***([type =]) source_description import_specification*

- *source_description* should contain a description of the file from which the data is to be imported. The specification of the description is usually just the path and file name of the file, however you can also specify more precise information. See `wfopen` (p. 640) of the *Command and Programming Reference* for more details on the specification of *source_description*.
- The optional "type = " option may be used to specify a source type. For the most part, you should not need to specify a "type = " option as EViews will automatically determine the type from the filename. The following table summaries the various source formats with the corresponding "type = " keywords:

	Option Keywords
Excel (through 2003)	"excel"
Excel 2007 (xml)	"excelxml"
HTML	"html"
Text / ASCII	"text"

- *import_specification* can be used to provide additional information about the file to be read. The details of import_specification will depend upon the type of file being imported.

Excel Files

The syntax for reading Excel files is:

matrix_name.**import(type = excel***[xml]*) *source_description [table_description] [variables_description]*

The following *table_description* elements may be used when reading Excel data:

- "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.

- "byrow", transpose the incoming data. This option allows you to read files where the series are contained in rows (one row per series) rather than columns.

The optional *variables_description* may be formed using the elements:

- "colhead = *int*", number of table rows to be treated as 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 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 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.

Excel Examples

```
matrix_name.import "c:\data files\data.xls"
```

loads the active sheet of DATA.XLSX into the MATRIX_NAME matrix object.

```
matrix_name.import "c:\data files\data.xls" range="GDP data"
```

reads the data contained in the "GDP data" sheet of "Data.XLS" into the MATRIX_NAME object.

HTML Files

The syntax for reading HTML pages is:

matrix_name.**import(type = html)** *source_description [table_description] [variables_description]*

The following *table_description* elements may be used when reading an HTML file or page:

- "table = *arg*", where *arg* specifies which HTML 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.

- "byrow", transpose the incoming data. This option allows you to import files where the series are contained in rows (one row per series) rather than columns.

The optional *variables_description* may be formed using the elements:

- "colhead = *int*", number of table rows to be treated as 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.

HTML Examples

```
mat1.import "c:\data.html"
```

loads into the MAT1 matrix object the data located in the HTML file "Data.HTML" located on the C:\ drive

```
mat1.import(type=html) "http://www.tradingroom.com.au/apps/mkt/
    forex.ac" colhead=3
```

loads into a matrix object MAT1 the data with the given URL located on the website site "http://www.tradingroom.com.au". The column header is set to three rows.

Text and Binary Files

The syntax for reading text or binary files is:

matrix_name.**import(type** = *arg***)** *source_description [table_description]*
[variables_description]

If a *table_description* is not provided, EViews will attempt to read the file as a free-format text file. The following *table_description* elements may be used when reading a text or binary file:

- "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.
- "byrow", transpose the incoming data. This option allows you to import files where the series are contained in rows (one row per series) rather than columns.
- "lastcol", include implied last column. For lines that end with a delimiter, this option adds an additional column. When importing a CSV file, lines which have the delimiter as the last character (for example: "name, description, date,"), EViews normally determines the line to have 3 columns. With the above option, EViews will determine the line to have 4 columns. Note this is not the same as a line containing "name, description, date". In this case, EViews will always determine the line to have 3 columns regardless if the option is set.

A central component of the *table_description* element is the format statement. You may specify the data format using the following table descriptors:

- Fortran Format:

  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 Format:

  rformat = "*[n1]Type[Width][.Precision]*, *[n2]Type[Width][.Precision]*, ...)"

  where optional type is "$" for string or "#" for number, and *n1*, *n2*, *n3*, *n4*, *etc.* are the range of columns containing the data.

- C printf/scanf Format:

  cformat = "*fmt*"

  where *fmt* follows standard C language (printf/scanf) format rules.

The optional *variables_description* may be formed using the elements:

- "colhead = *int*", number of table rows to be treated as 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.

Text and Binary File Examples (.txt, .csv, etc.)

```
mat2.import c:\data.csv skip=5
```

reads "Data.CSV" into a MAT2, skipping the first 5 rows.

```
mat2.import(type=text) c:\date.txt delim=comma
```

loads the comma delimited data "Date.TXT" into the MAT2 matrix object.

Cross-references

See also `Matrix::export` (p. 568).

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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User’s Guide I* for a discussion of labels.

See also `Matrix::displayname` (p. 565).

makepcomp	Matrix Procs

Save the scores from a principal components analysis of the series in a matrix.

Syntax

matrix_name.**makepcomp***(options) output_list*

where the *output_list* is a list of names identifying the saved components. 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

scale = *arg* *(default* = "normload"*)*	Diagonal matrix scaling of the loadings and the scores: normalize loadings ("normload"), normalize scores ("normscores"), symmetric weighting ("symmetric"), user-specified (*arg* = *number*).
cpnorm	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).
eigval = *vec_name*	Specify name of vector to hold the saved the eigenvalues in workfile.
eigvec = *mat_name*	Specify name of matrix to hold the save the eigenvectors in workfile.
prompt	Force the dialog to appear from within a program.

Covariance Options

cov = *arg* *(default* = "corr"*)*	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.
wgt = *name* (optional)	Name of vector containing weights. The number of rows of the weight vector should match the number of rows in the original matrix.

wgtmethod = *arg* (default = "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.
pairwise	Compute using pairwise deletion of observations with missing cases (pairwise samples).
df	Compute covariances with a degree-of-freedom correction accounting for the mean (for centered specifications) and any partial conditioning variables. The default behavior in these cases is to perform no adjustment (*e.g.* – compute sample covariance dividing by n rather than $n - k$).

Examples

```
mat1.makepcomp 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.

```
mat1.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:

```
mat1.makepcomp(scale=normscore, cpnorm) comp1 comp2 comp3
```

Cross-references

See "Saving Component Scores," beginning on page 696 of *User's Guide I* for further discussion. See `Matrix::pcomp` (p. 584) for tools to display the principal components results for the matrix.

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 11. "Matrix Language," beginning on page 279 of the *Command and Programming Reference* for further discussion.

See "Rowvector" (p. 701) and "Vector" (p. 1221) and "Sym" (p. 989) for full descriptions of the various matrix objects.

olepush	Matrix Procs

Push updates to OLE linked objects in open applications.

Syntax

matrix_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

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

out = *arg* *(default* = "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.
eigval = *vec_name*	Specify name of vector to hold the saved the eigenvalues in workfile.
eigvec = *mat_name*	Specify name of matrix to hold the save the eigenvectors in workfile.
prompt	Force the dialog to appear from within a program.
p	Print results.

Number of Component Options

fsmethod = *arg* *(default* = "simple"*)*	Component retention method: "bn" (Bai and Ng (2002)), "ah" (Ahn and Horenstein (2013)), "simple" (simple eigenvalue methods), "user" (user-specified value). Note the following: (1) If using simple methods, the minimum eigenvalue and cumulative proportions may be specified using "minigen = " and "cproport = ". (2) If setting "fsmethod = user" to provide a user-specified value, you must specify the value with "r = ".
r = *arg (default* = 1*)*	User-specified number of components to retain (for use when "fsmethod = user").
mineigen = *arg* *(default* = 0*)*	Minimum eigenvalue to retain component (when "fsmethod = simple").
cproport = *arg* *(default* = 1.0*)*	Cumulative proportion of eigenvalue total to attain (when "fsmethod = simple").
mfmethod = *arg* *(default = "user")*	Maximum number of components used by selection methods: "schwert" (Schwert's rule, *default*), "ah" (Ahn and Horenstein's (2013) suggestion), "rootsize" ($\min(\sqrt{N}, \sqrt{T})$), "size" ($\min(N, T)$), "user" (user specified value), where N is the number of columns of the matrix and T is the number of rows. (1) For use with all components retention methods apart from user-specified ("fsmethod = user"). (2) If setting "mfmethod = user", you may specify the maximum number of components using "rmax = ". (3) Schwert's rule sets the maximum number of components using the rule: let $$L(k) = k(T/100)^{1/4}$$ for $k \in \{2, 4, 6, 8, 10, 12\}$ and let $T^* = \min(N, T)$; then the default maximum lag is given by $$L^* = \max_k\{L(k) \mid L(k) < T^*\}$$
n = *arg* or rmax = *arg* *(default* = all*)*	User-specified maximum number of factors to retain (for use when "mfmethod = user").

fsic = *arg (default* = avg*)*	Component selection criterion when "fsmethod = bn": "icp1" (ICP1), "icp2" (ICP2), "icp3" (ICP3), "pcp1" (PCP1), "pcp2" (PCP1), "pcp3" (ICP3), "avg" (average of all criteria ICP1 through PCP3). Component selection criterion when "fsmethod = ah": "er" (eigenvalue ratio), "gr" (growth ratio), "avg" (average of eigenvalue ratio and growth ratio). Component selection when "fsmethod = simple": "min" (minimum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "max" (maximum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "avg" (average the optimal number of factors as specified by the min and max rule, then round to the nearest integer).
demeantime	Demeans observations across time prior to component selection procedures.
sdizetime	Standardizes observations across time prior to component selection procedures.
demeancross	Demeans observations across cross-sections prior to component selection procedures.
sdizecross	Standardizes observations across cross-sections prior to component selection procedures.

Covariance Options

cov = *arg* *(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").
wgt = *name* (optional)	Name of vector containing weights. The number of rows of the weight vector should match the number of rows in the original matrix.
wgtmethod = *arg* (default = "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.
pairwise	Compute using pairwise deletion of observations with missing cases (pairwise samples).

df	Compute covariances with a degree-of-freedom correction accounting for the mean (for centered specifications) and any partial conditioning variables. The default behavior in these cases is to perform no adjustment (*e.g.* – compute sample covariance dividing by n rather than $n-k$).

Graph Options

scale = *arg,* *(default* = "normload"*)*	Diagonal matrix scaling of the loadings and the scores: normalize loadings ("normload"), normalize scores ("normscores"), symmetric weighting ("symmetric"), user-specified (*arg = number)*.
mult = *arg* *(default* = "first")	Multiple series handling: plot first against remainder ("first"), plot as x-y pairs ("pair"), lower-triangular plot ("lt").
nocenter	Do not center graphs around the origin. By *default*, EViews centers biplots around (0, 0).
labels = *arg,* *(default* = "outlier")	Observation labels for the scores: outliers only ("outlier"), all points ("all"), none ("none").
labelprob = *number*	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).
autoscale = *arg*	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 "user-scale = " to scale the original loadings.
userscale = *arg*	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 "autoscale = " to scale the automatically scaled loadings.
cpnorm	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 685 of *User's Guide I* for further discussion. See also "Covariance Analysis," beginning on page 668 of *User's Guide I* 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 `Sym::eigen` (p. 1002) to examine the eigenvalues of a symmetric matrix.

read	Matrix Procs

Import data from a foreign disk file into a matrix.

(This is a deprecated method of importing into a matrix. See `Matrix::import` (p. 575) for the currently supported method.)

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 *default* 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.
skipcol = *integer*	Number of columns to skip. Must be used with the "*rect*" option.
skiprow = *integer*	Number of rows to skip. Must be used with the "rect" option.
comment = *symbol*	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.
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

t	Read data organized by column (transposed). Default is to read by row.
letter_number (default = "b2"*)*	Coordinate of the upper-left cell containing data.
s = *sheet_name*	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 152 of *User's Guide I* for a discussion and examples of importing data from external files.

See also `Matrix::export` (p. 568).

resample	Matrix Procs

Resample from rows in a group.

Syntax

group_name.**resample***(options) output_name*

You should follow the `resample` keyword and options with an *output_name* containing a name of the matrix to hold the output.

By default, EViews uses the *output_name* "*_b" so that "_b" will be appended to the name of the original matrix and used as the output name.

Options

rows = *n*	(Optional) Number of rows of the output matrix. Default is to create an output matrix with the same the number of rows as the source matrix.
permute	Draw from rows without replacement. Default is to draw with replacement.

weight = *vector_name*	Name of vector to be used for weighted sampling, containing values proportional to the desired row probabilities (importance sampling). The weight vector must have the same number of rows as the matrix, with non-missing, non-negative values. The weight values need not add up to 1, as EViews will normalize the weights. If no weights are provided, rows will be drawn with equal probability weights.
block = *integer*	Block length for each draw. Must be a positive integer. The default block length is 1.
withna *(default)*	[Draw / Do not draw] from all rows in the matrix, including those with NAs.
dropna	Do not draw from rows that contain missing values in the matrix.
fixna	Excludes NAs from draws but copies rows containing missing values to the output matrix.
prompt	Force the dialog to appear from within a program.

- Block bootstrap ("*block* = " length larger than 1) requires a contiguous output. Therefore a block length larger than 1 cannot be used together with the "fixna" option.
- The "*fixna*" option will have an effect only if there are missing values in the rows of the source matrix.
- If you specify "*fixna*", we first copy rows with any missing values in the matrix sample to the output matrix (if relevant). Then the remaining rows of the output matrix will be resampled from the remaining rows of the input matrix.
- If you choose "*dropna*" and the block length is larger than 1, the rows of the matrix used for simulation may adjust 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

```
matrix(100, 5) f
nrnd(f)
f.resample(n = 1000) f_resample
```

creates a new matrix F_RESAMPLE obtained by drawing 1000 rows with replacement from F.

```
f.resample(weight=wt) f_weighted
```

will compute a weighted resample and save the results to the same sized matrix F_WEIGHTED. The rows in the source will be drawn with probabilities proportional to the corresponding values in the series WT. WT must have the same number of rows as F and must contain non-missing, non-negative values.

Cross-references

See "Resample" on page 502 of *User's Guide I* for a discussion of the resampling procedure. For additional discussion of wildcards, see Appendix A. "Wildcards," on page 1227 of *User's Guide I*.

See also `@resample` (p. 1071) and `@permute` (p. 1037) in the *Command and Programming Reference* for sampling from matrices.

resize	Matrix Procs

Resize the matrix object.

Syntax

matrix_name.**resize** *rows cols*

Examples

```
mat1.resize 3 5
```

resizes the matrix MAT1 to 3 rows and 5 columns, retaining the contents of any existing elements and initializing new elements to 0.

setattr	Matrix Procs

Set the object attribute.

Syntax

alpha_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the "revised" attribute in the object A to the string "never", and extracts the attribute into the string object S.

Cross-references

See "Adding Custom Attributes in the Label View" on page 123 and "Adding Your Own Label Attributes" on page 70 of *User's Guide I*.

setcollabels | Matrix Procs

Set the column labels in a matrix object.

Syntax

matrix_name.**setcollabels** *label1 label2 label3....*

Follow the `setcollabels` command with a space delimited list of column labels. Note that each column label should not contain spaces unless it is enclosed in quotes. If you provide fewer labels than there are columns, EViews will keep the corresponding default column names ("C11", "C12", etc...).

Examples

```
mat1.setcollabels USA UK FRANCE
```

sets the column label for the first column in matrix MAT1 to USA, the second to UK, and the third to FRANCE.

Cross-references

See `Matrix::setrowlabels` (p. 596).

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:

g*[.precision]*	significant digits
f*[.precision]*	fixed decimal places
c*[.precision]*	fixed characters

e*[.precision]*	scientific/float
p*[.precision]*	percentage
r*[.precision]*	fraction

To specify a format that groups digits into thousands using a comma separator, place a "t" after the format character. For example, to obtain a fixed number of decimal places with commas used to separate thousands, use "ft*[.precision]*".

To use the period character to separate thousands and commas to denote decimal places, use ".." (two periods) when specifying the precision. For example, 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)"), 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. 597), `Matrix::setindent` (p. 595) and `Matrix::setjust` (p. 596) 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. 597) and `Matrix::setjust` (p. 596) for details on setting spreadsheet widths and justification.

setjust | Matrix Procs

Set the horizontal justification for all cells in the spreadsheet view of the matrix object.

Syntax

matrix_name.**setjust** *format_arg*

where *format_arg* may be set to left, center, right, or auto (strings are left-justified and numbers are right-justified). Default display settings can be set in General Options; see "Spreadsheet Data Display" on page 1019 of *User's Guide I*.

Examples

```
mat1.setjust left
```

left-justifies the cells in the spreadsheet view of the matrix MAT1.

Cross-references

See `Matrix::setwidth` (p. 597) and `Matrix::setindent` (p. 595) for details on setting spreadsheet widths and indentation.

setrowlabels | Matrix Procs

Set the row labels in a matrix object.

Syntax

matrix_name.**setrowlabels** *label1 label2 label3....*

Follow the `setrowlabels` command with a space delimited list of row labels. Note that each row label should not contain spaces unless it is enclosed in quotes. If you provide fewer labels than there are rows, EViews will use the corresponding default row names ("R11", "R12", etc...).

Examples

```
mat1.setrowlabels USA UK FRANCE
```

sets the row label for the first row in matrix MAT1 to USA, the second to UK, and the third to FRANCE.

Cross-references

See `Matrix::setcollabels` (p. 594).

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. 595) and `Matrix::setjust` (p. 596) 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.

showlabels

Matrix Procs

Displays the custom row and column labels of a matrix spreadsheet.

Syntax

matrix_name.**showlabels** *mode*

where *mode* is either 0 or 1 where 0 displays the default row and column labels and 1 displays the custom row and column labels (if present).

Examples

```
m1.showlabels 1
```

displays the custom row and column labels for the M1 spreadsheet. If custom labels have not been set the default labels will be displayed.

```
m1.showlabels 0
```

displays the default row and column labels for the M1 spreadsheet.

Cross-references

See also `Matrix::setcollabels` (p. 594) and `Matrix::setrowlabels` (p. 596).

stats

Matrix Views

Descriptive statistics for columns of a matrix.

Computes and displays a table of means, medians, maximum and minimum values, standard deviations, and other descriptive statistics for each column of the matrix.

Syntax

matrix_name.**stats***(options)*

Options

i	Individual sample for each column after removing missing values for the column. By default, EViews computes the statistics using a common sample after removing rows with missing values.
p	Print the stats table.

Examples

The command

```
matrix1.stats
```

computes the descriptive statistics for each column of MATRIX1 after listwise deletion of rows with missing values. Alternately,

```
matrix1.stats(i)
```

displays the descriptive statistics view of MATRIX1 showing the statistics for each column of the matrix computed using individual samples.

Cross-references

See "Descriptive Statistics" on page 667 of *User's Guide I* for a discussion of the descriptive statistics views of a group.

See also `boxplot` (p. 1279).

testbtw	Matrix Views

Test equality of the mean, median or variance of the series in the group.

Syntax

group_name.**testbtw***(options)*

By default, `testbtw` will test for equality of means, but you may specify instead tests of medians or variances as an option, and choose whether to use balanced or unbalanced samples.

Options

mean *(default)*	Test equality of mean.
med	Test equality of median.
var	Test equality of variance.
c	Use common (balanced) sample for computing statistics after removing observations with missing values for any series in the group. By default, EViews computes the statistics for each series using an individual (potentially unbalanced) sample after removing observations with missing values for the series.
prompt	Force the dialog to appear from within a program.
p	Print the test results.

Examples

The commands

```
group g1 wage_m wage_f
g1.testbtw
```

uses both parametric ANOVA and nonparametric approaches to test the equality of means of the two series WAGE_M and WAGE_F using individual samples to compute the means. Alternately, the command,

```
g1.testbtw(var, c)
```

tests for equality of variances using a balanced sample after listwise deletion of rows with missing values.

Cross-references

See "Tests of Equality" on page 684 of *User's Guide I* for further discussion of these tests.

See also `Series::testby` (p. 857) and `Series::teststat` (p. 858).

testbtw	Matrix Views

Test equality of the mean, median or variances of the columns in a matrix.

Syntax

matrix_name.**testbtw***(options)*

By default, `testbtw` will test for equality of means, but you may specify instead tests of medians or variances as an option, and choose whether to use balanced or unbalanced samples.

Options

mean *(default)*	Test equality of means.
med	Test equality of medians.
var	Test equality of variances.
c	Use common (balanced) sample for computing statistics after removing rows with missing values for any element in the row. By default, EViews computes the statistics for each column using a individual (potentially unbalanced) sample after removing observations with missing values for the column.
prompt	Force the dialog to appear from within a program.
p	Print the test results.

Examples

The command

```
mat1.testbtw
```

uses both parametric ANOVA and nonparametric approaches to test the equality of the means of the columns of MAT1 using individual samples. Alternately, the command,

```
mat1.testbtw(var, c)
```

tests for equality of the column variances using a balanced sample after listwise deletion of rows with missing values.

Cross-references

See "Tests of Equality" on page 684 of *User's Guide I* for further discussion of these tests.

See also `Series::testby` (p. 857) and `Series::teststat` (p. 858).

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.

This routine should realistically only be used in the oft-hand chance that you wish to write into a Lotus file. Improved Excel, text, and other format writing is available in `Matrix::export` (p. 568).

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

na = *string*	Specify text string for NAs. Default is "NA".
d = *arg*	Specify delimiter *(default* is tab*)*: "s" (space), "c" (comma).
t	Write by column (transpose the data). Default is to write by row.

Spreadsheet (Lotus, Excel) files

letter_number	Coordinate of the upper-left cell containing data.
t	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 171 of *User's Guide I* for a discussion.

See also `Matrix::export` (p. 568), `Matrix::read` (p. 589).

Model

Set of simultaneous equations used for forecasting and simulation.

Model Declaration

model..... declare model object (p. 629).

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 1203 of *User's Guide II*.

Model Views

block..... display model block structure (p. 612).
boundscheck ..... show a display of any series that have crossed their set boundaries during a solve (p. 613).
compare ..... show the differences between scenarios for the specified series (p. 614).
display ..... display table, graph, or spool in object window (p. 618).
eqs ..... view of model organized by equation (p. 620).
digraph ..... display the model dependency graph (p. 622).
label..... view or set label information for the model (p. 625).
msg..... display model solution messages (p. 630).
printview ..... show enhanced display of the mode specification (p. 631).
scenlist..... display list description of the model scenarios (p. 638).
text ..... show text showing equations in the model (p. 644).
trace..... view of trace output from model solution (p. 644).
vars..... view of model organized by variable (p. 647).

Model Procs

addassign ..... assign add factors to equations (p. 608).
addinit ..... initialize add factors (p. 609).
adjust..... prepare a variable for editing in the current scenario and/or update its values using an array expression (p. 611).
addover ..... set the active scenario add factor overrides (p. 610).
append ..... append a line of text to a model (p. 612).
clearhist ..... clear the contents of the history attribute (p. 613).
clearremarks ..... clear the contents of the remarks attribute (p. 614).

unlink ..... break links in specification (p. 645).
update ..... update model specification (p. 646).

Model Data Members

Scalar values

@bounds..... integer containing the number of variables that crossed their boundaries during the previous solve.

String values

@attr("*arg*" ..... string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@description ..... string containing the Model object's description (if available).
@detailedtype ..... string with the object type: "MODEL".
@displayname..... string containing the Model object's display name. If the Model has no display name set, the name is returned.
@droppedaf..... string containing a space delimited list of the add factor series dropped or converted to exogenous series, if any, as part of the last successful fliptype procedure. The order of series in this list matches the order of series in the related data member @dropedaf-dep.
@droppedafdep .... string containing a space delimited list of dependent series that had add factors dropped or converted to exogenous series, if any, as part of the last successful fliptype procedure. The order of series in this list matches the order of series in the related data member @dropedaf.
@name ..... string containing the Model object's name.
@remarks ..... string containing the Model object's remarks (if available).
@scenarios..... string containing a list of scenarios in the Model.
@type ..... string with the object type: "MODEL".
@updatetime..... string representation of the time and date at which the Model was last updated.

String values for Model variables

@addfactors[("scenario")] or @aflist[("scenario")] string containing a space delimited list of the model's addfactor variables in the specified scenario (default is Actuals).
@depends("variable")string containing the variables that "variable" depends on.
@endoglist[("scenario")] string containing a space delimited list of the model's endogenous variables in the specified scenario (default is Actuals).
@excludelist[("scenario")] string containing a space delimited list of the model's excluded variables in the specified scenario (default is Actuals).

@exoglist[("scenario")] string containing a space delimited list of the model's exogenous variables in the specified scenario (default is Actuals).

@identity.....string containing a space delimited list of the model's endogenous variables determined by identities.

@overridelist[("scenario")] or **@olist**[("scenario")] string containing a space delimited list of the model's variables set as overrides in the specified scenario (default is Actuals).

@linklist.....string containing space delimited list of all linked objects in the model

@spec("variable") .string containing the estimation object name or text specification of the equation determining the specified endogenous variable, or an empty string if "variable" is an invalid name.

@stochastic.....string containing a space delimited list of stochastic endogenous variables.

@upends("variable")string containing the variables that depend on "variable".

@varlist[("scenario")] string containing a space delimited list of all the model's variables for the specified scenario (default is Actuals).

In addition to a scenario name, you may specify "@active" (in quotes) to specify the current active scenario or "@alternate" to specify the current alternative scenario.

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 shorthand keywords:

@all	All equations.
@stochastic	All stochastic equations (no identities).
@identity	All identities.

The options identify the type of add factor to be used, and control the assignment behavior for equations where you have previously assigned add factors. `addassign` may be called multiple times to add different types of add factors to different equations. `addassign` may also be called to remove existing add factors.

Options

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 1212 of *User's Guide II*. See also Chapter 52. "Models," beginning on page 1177 of *User's Guide II* for a general discussion of models.

See `Model::addinit` (p. 609).

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

v = *arg* (*default* = "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; override = actual), "a" (set add factor values so that the equation has no residual when evalulated at actives).
s = *arg* (*default* = "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 1212 of *User's Guide II*. See also Chapter 52. "Models," on page 1177 of *User's Guide II* for a general discussion of models.

See also `Model::addassign` (p. 608).

addover — Model Procs

Add override.

Set the active scenario add factor overrides.

Syntax

model.**addover***(options)*

Examples

o	Use scenario add factors (default is to use the baseline factors)
nc	Do not create active scenario add factor series if they do not already exist (default is to create the series if necessary).

Examples

```
mod1.addover(o)
```

overrides the active scenario's add factors.

Cross-references

See Chapter 52. "Models," on page 1177 of *User's Guide II* for a discussion of specifying and solving models in EViews. See also `Model::append` (p. 612), `Model::merge` (p. 628) and `Model::solve` (p. 640).

adjust	Model Procs

Prepare a variable for editing in the current scenario and/or update its values using an array expression.

Syntax

model_name.**adjust***(options) ser [array expression]...*

The adjust proc allows you to adjust the values of the series *ser* in the current scenario. If the series is an exogenous variable, it will be added to the override list. If the series is an endogenous variable it will be added to both the excluded list and the override list.

If an array expression is provided, the overridden series will be modified according to the expression specification. Note that a transform may optionally be provided as part of the variable name using the syntax: `transform(`*varname*`)`.

If you use an array expression which applies an operator to existing series values the overriden series must already exist in the workfile, unless the *init* option is used.

Options

init[=scenario]	Initialize the overridden variable with values from the specified scenario before applying any adjustment. If no scenario name is provided, the variable is initialized with values from the base scenario.

Examples

```
mod1.adjust gdp
```

simply sets the variable GDP as an overriden variable in the current scenario. If GDP is endogenous, it is also added to the exclude list.

```
mod1.scenario(a=_1) "sim1"
mod1.adjust gdp =+10
```

sets the current scenario as “SIM1”, with an alias of _1, and then overrides the variable GDP, setting the override series, GDP_1, equal to the previous values in GDP_1 plus 10.

```
mod1.scenario(a=_1) "sim1"
mod1.adjust(init="sim2") gdp =+10
```

performs the same operation, but rather than using the previous values in GDP_1 for the array expression, the values in GDP_2 (corresponding to the scenario SIM2) are used.

Cross-references

See Chapter 52. "Models," on page 1177 of *User's Guide II* for a discussion of specifying and solving models in EViews. See also the discussion in "Specifying Scenarios" on page 1208 of *User's Guide II*.

See `Model::scenario` (p. 636) and `Model::compare` (p. 614).

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 1177 of *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.**checkbounds***(options)*

Options

p	Print the block structure view.

Cross-references

See "Block Structure View" on page 1202 of *User's Guide II* for details. Chapter 52. "Models," on page 1177 of *User's Guide II* provides a general discussion of models.

See also `Model::eqs` (p. 620), `Model::text` (p. 644) and `Model::vars` (p. 647) for alternative representations of the model.

boundscheck — Model Views

Display the model check boundaries view.

Show a display of any series that have crossed their set boundaries during a solve.

Syntax

model_name.**boundscheck***(options)*

Options

p	Print the bounds check view.

Cross-references

See "Block Structure View" on page 1202 of *User's Guide II* for details. Chapter 52. "Models," on page 1177 provides a general discussion of models.

See also `Model::setbounds` (p. 639)l.

clearhist — Model Procs

Clear the contents of the history attribute for model objects.

Removes the model's history attribute, as shown in the label view of the model.

Syntax

model_name.**clearhist**

Examples

```
m1.clearhist
m1.label
```

The first line removes the history from the model M1, and the second line displays the label view of M1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Model::label` (p. 625).

clearremarks	Model Procs

Clear the contents of the remarks attribute.

Removes the model's remarks attribute, as shown in the label view of the model.

Syntax

model_name.**clearremarks**

Examples

```
m1.clearremarks
m1.label
```

The first line removes the remarks from the model M1, and the second line displays the label view of M1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Model::label` (p. 625).

compare	Model Views

Produce a table showing the differences between scenarios for the specified series.

Syntax

model_name.**compare***(options) model_vars*

The compare view allows you to quickly compare the results from different scenarios (or the actual values) following a model solve. By default the output table will show any of the series specified in *model_vars* whose difference between the current active and comparison scenarios exceeds a specified tolerance. You may optionally use the "patt = " option to specify a separate set of comparison series from those in the current comparison scenario.

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.
@overides	All currently overridden exogenous variables
@excludes	All currently overridden endogenous variables

Options

tol = *num*	Set the tolerance level for comparing the series. Any differences below the tolerance will not be reported. Default value is 0.001.
patt = *"pattern"*	Set the comparison set of series. Without this option, EViews will build the comparison set based upon the current comparison scenario. This option allows you to select a different set of series using pattern matching. *pattern* should contain an * to represent the variable names given in *model_vars*.

Examples

```
mod1.scenario(a="_0") "scenario0"
mod1.scenario(c, a="_1") "scenario1"
mod1.solve(a=t)
mod1.compare gdp unemp infl
```

The first two lines of this example set the current active scenario "Scenario0", and set the comparison scenario to "Scenario1", with a name alias of "_1". The model is then solved for both scenarios. The `compare` command is used to produce a table detailing the differences between the two scenarios for the three variables GDP, UNEMP and INFL. Any differences between the solved series GDP_0 and GDP_1, UNEMP_0 and UNEMP_1 or INFL_0 and INFL_1 greater than 0.001 will be shown in the table.

```
mod1.compare(tol=0.00001) gdp unemp infl
```

produces the same table, but uses a lower tolerance rate (of 0.00001).

```
mod1.compare @endog
```

produces a table comparing all endogenous variables in the model, not just GDP, UNEMP and INFL.

```
mod1.compare(patt="*_2") gdp unemp infl
```

produces a table that compares GDP_0 with GDP_2, UNEMP_0 with UNEMP_2 and INFL_0 with INFL_2, even though the current comparison scenario is still "Scenario1".

Cross-references

See Chapter 52. "Models," on page 1177 of *User's Guide II* for a discussion of specifying and solving models in EViews. See also, the discussion in "Specifying Scenarios" on page 1208 of *User's Guide II*.

See `Model::scenario` (p. 636) and `Model::adjust` (p. 611).

control	Model Procs

Solve for values of one or more control variables so that an equal number of target series follow specified trajectories over the current workfile sample.

Syntax

model_name.**control***(options) controls targets trajectories*

The controls, targets, and trajectories parameters are all lists of variable/series names, which may comprise any combination of individual series or group object names. The controls should be exogenous variables and the targets endogenous variables, with the trajectories being arbitrary series. An equal number of control, target, and trajectory series must be specified. These three lists do not need to be ordered strictly as above; they may be interleaved or kept together in blocks. For example, any of the following arrangements of the six arguments for a two-variable control would be acceptable: Examples

```
control1 control2 target1 target2 trajectory1 trajectory2
control1 target1 trajectory1 control2 target2 trajectory2
control1 control2 target1 trajectory1 target2 trajectory2
```

The solved values overwrite the original control series unless the 'create' or 'csuffix' option is specified. When the names for target and trajectory series share a common base, the 'tsuffix' option may be used to abridge the parameter list.

Options

create	Store the solved values in series with a given suffix rather than overwriting the original control series. The destination series names are the control names appended with either the active scenario alias suffix (the default) or the suffix specified by the csuffix option.
csuffix = *suff*	Specifies the series name suffix for storing modified control values. The presence of this option implies the create option above.
tsuffix = *suff*	Specifies the series name suffix for generating trajectory names from target names. When this option is present, the trajectories argument should be omitted.

Examples

```
m1.control myvar targetvar trajvar
```

will alter the values of exogenous variable MYVAR such that the model solution will produce values for endogenous variable TARGETVAR to match series TRAJVAR over the current workfile sample.

```
m1.control(create) isav irtn ftot ftot_tr gtot gtot_tr
```

will determine the values of exogenous variables ISAV and IRTN such that the model solution will produce values for endogenous variables FTOT and GTOT that match series FTOT_TR and GTOT_TR, respectively. Assuming the current model scenario suffix is _0, the control values will be stored in series named ISAV_0 and IRTN_0.

```
m1.control(create, tsuffix=_tr) isav irtn ftot gtot
```

similar to the previous example, but the trajectory series names are implicitly defined as the target names appended with the _tr suffix, i.e., FTOT_TR and GTOT_TR.

Cross-references

See "Solve Control for Target" on page 1232 of *User's Guide II*. See Chapter 52. "Models," on page 1177 of *User's Guide II* for a general discussion of models.

copy	Model Procs

Creates a copy of the model.

Creates either a named or unnamed copy of the model.

Syntax

model_name.**copy**

model_name.**copy** *dest_name*

Examples

```
m1.copy
```

creates an unnamed copy of the model M1.

```
m1.copy m2
```

creates M2, a copy of the model M1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

display	Model Views

Display table, graph, or spool output in the model object window.

Display the contents of a table, graph, or spool in the window of the model object.

Syntax

model_name.**display** *object_name*

Examples

```
model1.display tab1
```

Display the contents of the table TAB1 in the window of the object MODEL1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

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 123 of *User's Guide I* for a discussion of labels and display names.

See also `Model::label` (p. 625).

drop	Model Procs

Drop equations for one or more endogenous variables in the model.

Syntax

model_name.**drop**(*options*) *var_list*

Where *var_list* is a space delimited list of variables whose equations will be dropped from the model. By default if a variable is contained in a multi-equation object, such as a system, VAR or model, the entire object will be dropped, which will also drop the specification for the other variables defined in that object.

Options

nomult	Do not drop multi-equation objects
noerr	Suppress variable not found errors

Examples

```
m1.drop gdp
```

will drop the equation/object which has GDP as a dependent variable from the model M1.

```
m2.drop(nomult) gdp unemp
```

will drop the equations which have GDP or UNEMP as dependent variables. Systems, VARs, models, etc… will not be dropped.

Cross-references

droplink	Model Procs

Drop linked objects from the model.

Syntax

model_name.**droplink**(*options*) obj_list

Where obj_list is a space delimited list of objects to be dropped from the model.

Options

noerr	Suppress object link not found errors

Examples

```
m1.droplink eq1 mod1
```

will drop the equation EQ1 and the model MOD1 from the model M1.

Cross-references

endog	Model Views

Note that `endog` and `makeendog` are no longer supported for model objects. See instead, `Model::makegroup` (p. 627).

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 1199 of *User's Guide II* for details. See Chapter 52. "Models," on page 1177 of *User's Guide II* for a general discussion of models.

See also `Model::block` (p. 612), `Model::text` (p. 644), and `Model::vars` (p. 647) 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.)
r	Re-include a variable (drop from the exclude list).

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 govexp
```

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 1208 of *User's Guide II*.

See also Model::override (p. 630), Model::reinclude (p. 632), and Model::solveopt (p. 641).

digraph	Model Views

Display the model dependency graph.

Syntax

model_name.**digraph**

Examples

```
model1.digraph
```

visualizes the dependencies among the model's variables as a directed graph.

Cross-references

See "Dependency Graph View" in *User's Guide II* for further details on the model dependency graph.

fliptype	Model Procs

Symbolically alter the model's equations to specify a new set of endogenous variables.

Syntax

model_name.**fliptype**(*options*) *variables* [@lock *fixed-variables*] [@lockafs] [@lockinnovs]

The *variables* parameter is a list of endogenous and exogenous variables in any order. Group objects may be included in the list. An equal number of endogenous and exogenous variables must be specified. Each listed endogenous variable will be made exogenous and each listed exogenous variable will be made endogenous by rewriting a number of the model's equations. The number of equations modified will be at least the number of endogenous/exogenous variables specified, e.g., flipping a set of three endogenous and three exogenous variables will require at least three equations to be modified. All modified equations will have any associated add factor or innovation variance dropped from their specification (see the af2exog option below). The optional @lock clause can be used to prevent the modification of any equation whose endogenous variable is listed in *fixed-variables*. Inclusion of the optional @lockafs or @lockinnovs tags indicates that all equations with either add factors or innovations variances, respectively, should not be modified.

Options

dest = *name*	Instead of altering the invoking model object, a copy of the model named *name* will be created and the flip applied to this new model.
noshift	Rewriting an equation may involve a shift in its lag structure (because endogenous variables must be specified at lag zero). For example, flipping the two variables in the single equation "y = x(-1)" produces "x = y(1)". Such lag shifting is allowed by default, but can be prevented by including this option. Without lag shifting, EViews may have fewer potential ways to rewrite each equation, possibly resulting in a greater number of altered equations or even an inability to find any satisfactory set of rewrites.
af2exog [= *suffix*]	Add factor series that are dropped are instead converted to exogenous series and explicitly appear in the model's equations. This allows the resulting model to remain numerically consistent with the original model. With the optional *suffix* specification, the new exogenous series are named as the add factor series followed by *suffix*, otherwise the original add factor series names are used.

Examples

```
mod1.fliptype var1 var2
```

will flip the endogenous/exogenous role of variables VAR1 and VAR2 in the model by rewriting the model's equations. For example, if VAR1 is initially endogenous and VAR2 is initially exogenous, the procedure will rewrite the equations of the model so that VAR2 becomes endogenous (the first left-hand side variable of some equation) and VAR1 becomes exogenous (not the first left-hand side variable of any equation). The role of all other variables in the model is unchanged.

Cross-references

See "Solve Control for Target" on page 1232 of *User's Guide II*. See Chapter 52. "Models," on page 1177 of *User's Guide II* for a general discussion of models.

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

"i" or "identity"	Specifies that the equation is an identity in stochastic solution.
"s" or "stochastic"	Specifies that the equation is stochastic with unknown innovation variance in stochastic solution. Note: if a value has been previously specified in the *positive_num* option, it will be kept.
positive_num	Specifies that the equation is stochastic with an equation innovation standard error equal to the positive number *positive_num*. Note: the innovation standard error is only relevant when used with the `Model::stochastic` command, with the "v=t" option set.

Options for exogenous variables

number	*number* 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 1222 of *User's Guide II*.

See also `Model::model` (p. 629), `Model::stochastic` (p. 643), and `Model::solve` (p. 640).

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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User’s Guide I* for a discussion of labels.

See also `Model::displayname` (p. 618).

makeendog	Model Procs

Note that in `endog` and `makeendog` are no longer supported for model objects. See instead, `Model::makegroup` (p. 627).

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.
@overides	All currently overridden exogenous variables
@excludes	All currently overridden endogenous variables

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 = *trans_type* (*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 = *sol_type* (*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 = *grouping (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 1237 of *User's Guide II* for details. See Chapter 52. "Models," on page 1177 of *User's Guide II* for a general discussion of models.

See `Model::makegroup` (p. 627).

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.
@overides	All currently overridden exogenous variables
@excludes	All currently overridden endogenous variables

Options

a	Include actuals.

c	Include comparison scenarios.
d	Include deviations.
r	Include percentage deviations.
n	Do not include active scenario (by default the active scenario is included).
t = *arg* *(default* = "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).
s = *arg* *(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 = *arg* *(default* = "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 1237 of *User's Guide II* for details. See Chapter 52. "Models," on page 1177 of *User's Guide II* for a general discussion of models.

See also `Model::makegraph` (p. 626).

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

t	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. 612) or `Model::merge` (p. 628).

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 52. "Models," on page 1177 of *User's Guide II* for a discussion of specifying and solving models in EViews.

See also `Model::append` (p. 612), `Model::merge` (p. 628) and `Model::solve` (p. 640).

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 52. "Models," on page 1177 of *User's Guide II* for a discussion of specifying and solving models in EViews.

See also `Model::solve` (p. 640) and `Model::solveopt` (p. 641).

olepush	Model Procs

Push updates to OLE linked objects in open applications.

Syntax

model_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

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.
r	Remove the variable from the 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:

```
modl.override(m) add1
```

The override list now contains both series.

Cross-references

See the discussion in "Specifying Scenarios" on page 1208 of *User's Guide II*. See also Chapter 52. "Models," on page 1177 of *User's Guide II* for a general discussion of models.

See `Model::exclude` (p. 621). `Model::scenario` (p. 636) and `Model::revert` (p. 635).

printview	Model Views

Display print view of the model.

Show enhanced display of the model specification.

Syntax

model_name.**printview***(options)*

Options

keeplink	Do not display the underlying equations in linked estimation objects (default is to break links in the display).
idents	Display identities.
innov	Display @innov statements.
addfact	Display add factors.
comments	Display comments.

dispnames	Use display names
decimals = *integer*	Display numbers using the specified number of decimal digits.
signif = *integer*	Display numbers using the specified number of significant digits.
p	Print the model solution messages.

Options

```
mod1.printview(idents, innov)
```

displays the model with broken links, identities, and @innov statements.

```
mod1.printview(idents, innov, signif=3)
```

displays numeric values with three significant digits.

Cross-references

See "Print View" on page 1204 of *User's Guide II* for a discussion

See also `Model::text` (p. 644).

reinclude

Model Procs

Removes one or more variables from the excluded variable list.

Syntax

Model_name.**reinclude**(*options*) ser1 ser2

The specified variables are removed from the current active scenario's exclude list, and generates an add factor for each variable so that the solution for the current scenario remains unchanged.

Options

v	Create variable shift add factors in cases where no add factor Is currently associated with the endogenous variable. (Default is to create intercept shifts).
skipidents	Ignore endogenous variables whose associated equation is tagged as an identity.

Examples

```
M1.reinclude x z
```

removes both X and Z from the exclude list in the current scenario, and creates add factors for each.

Cross-references

See Chapter 52. "Models," on page 1177 of *User's Guide II* for a discussion of specifying and solving models in EViews.

See also `Model::exclude` (p. 621).

replace	Model Procs

Replace the text specification for an endogenous variable in the model with a new specification.

Syntax

model_name.**replace** *new_specification*

The replace command will only replace the specification for variables that currently have a text specification in the model. Variables defined by a linked object cannot have their specification replace. New_specification should be the new text specification for the variable. Note EViews will automatically detect the endogenous variable in the new specification, even if it is contained in an implicit expression, and will replace the existing specification for that variable.

Examples

```
model m1
m1.append x = 3*y
m1.replace x = 4*y
```

this trivial example first creates a model, M1, then adds a text specification for the variable X, setting it equal to 3*Y. It then changes the specification for X to set it equal to 4*Y.

```
model m1
m1.append x = 3*y
m1.replace log(x) = 4*y
```

this example replaces the specification for X by setting the log of X equal to 4*Y.

```
model m1
m1.append log(x/w) = 3*y
m1.replace x^2 = 4*y
```

this example first defines using an expression, log(X/W), The replace command is used to change that definition to be in terms of X^2. EViews automatically detects that the new specification, even though it is based in terms of X^2, should be used to replace the cur-

rent line involving log(X/W).

Examples

See also Model::replacelink (p. 634) and Model::replacevar (p. 634).

replacelink

Model Procs

Replace a linked object with a different linked object.

Syntax

model_name.**replacelink** *old_obj new_obj*

old_obj should be the name of an object currently linked inside the model. That object will be removed from the model and replaced with the new object, *new_obj*.

Examples

```
M1.replacelink eq1 eq1_new
```

replaces the linked equation, EQ1, with a different equation, EQ1_NEW.

Cross-references

See also Model::append (p. 612), Model::replace (p. 633) and Model::replacevar (p. 634).

replacevar

Model Procs

Replace all instances of a variable in the text specification of a model with a different variable.

Syntax

model_name.**replacevar** *oldvar newvar*

`replacevar` can only replace variables defined by a text specification inside the model. It will not replace variables contained inside linked objects. Note that `replacevar` does not do a simple text substitution, and is capable of determining full variable names from other pieces of text.

Examples

```
Model m1
M1.append y = 3*x
M1.replacevar x z
```

this example creates a model, M1, and adds a text specification for the variable Y, setting it equal to 3*X. It then replaces all occurrences of X with Z, changing the specification of Y to be equal to 3*Z.

```
M1.append y=3*log(x(-2))
m1.append w = 4*x1
M1.replacevar x z
```

this example generates a specification for Y, setting it equal to 3 times the log of the twice lagged value of X, and a specification for W, setting it equal to 4 times X1. It then replaces all instances of the variable X with the variable Z, changing the specification of Y to be equal to 3 times the log of twice lagged Z. Note that the specification of W does not change, since X1 is a different variable from X.

Cross-references

See also `Model::append` (p. 612), `Model::replacelink` (p. 634) and `Model::replacevar` (p. 634).

revert	Model Procs

Reverts one or more overridden variables in the active model scenario back to their baseline values.

Syntax

model_name.**revert** *ser1* [*ser2* ...]

The specified variables will be removed from the override and exclude list of this scenario, and the associated overridden series in the workfile will be deleted.

If an asterisk is provided for the variable name, all overridden series in the active model scenario will be reverted.

Examples

```
M1.revert x z
```

removes X and Z from the override list in the current scenario.

Cross-references

See also `Model::override` (p. 630), `Model::exclude` (p. 621), and `Model::scenario` (p. 636).

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 = "*name*"	Copy the Excludes and Overrides from the named scenario.
d	Delete the named scenario.
a = *string*	Set the scenario alias string to be used when creating aliased variables (*string* is a 1 to 3 alphanumeric string to be used in creating aliased variables). If an underscore is not specified, one will be added to the beginning of the string. Examples: "_5", "_T", "S2". The string "A" may not be used since it may conflict with add factor specifications.
desc = *string*	Description of the scenario.
usedesc	Export the description specified in "desc = " to all solution series.
v	Copy the values of any overridden series in the scenario specified in the "i = " option into the overridden series for this scenario, creating new series if necessary.

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:

```
mod1.scenario(a=_ps) "Peace Scenario"
```

changes the alias for "Peace Scenario" to "_PS" and makes this scenario the active scenario.

The command:

```
mod1.scenario(n, a=w, i="Peace Scenario", c) "War Scenario"
```

creates a scenario called "War Scenario", initializes it with the Excludes and Overrides contained in "Peace Scenario", associates it with the alias "_W", and makes this scenario the comparison scenario.

```
mod1.scenario(i="Scenario 1") "Scenario 2"
```

copies the Excludes and Overrides in "Scenario 1" to "Scenario 2" and makes "Scenario 2" the active scenario.

Compatibility Notes

For backward compatibility with EViews 4, 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

```
mod1.scenario Scenario 1
```

is currently valid, but may not be in future versions of EViews.

Cross-references

Scenarios are described in detail in "Specifying Scenarios" on page 1208 of *User's Guide II*. Chapter 52. "Models," on page 1177 of *User's Guide II* documents EViews models.

See also `Model::solve` (p. 640).

scenlist	Model Views

Display list description of the model scenarios.

Syntax

model_name.**scenlist***(options)*

Options

p	Print the model scenario view.

Examples

```
model1.scenlist
```

displays the list of scenarios view of the object MODEL1.

Cross-references

See "Scenario List" on page 1211 of *User's Guide II* for further details on the scenario list view.

See also `Model::scenario` (p. 636).

setattr	Model Procs

Set the object attribute.

Syntax

model_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the "revised" attribute in the object A to the string "never", and extracts the attribute into the string object S.

Cross-references

See "Adding Custom Attributes in the Label View" on page 123 and "Adding Your Own Label Attributes" on page 70 of *User's Guide I*.

setbounds	Model Procs

Set upper and lower boundaries for endogenous variables during model solution.

EViews will warn if any variable's solved for value is higher than the upper bound or less than the lower bound for any observation during the solve.

Syntax

model_name.**setbounds** variable1(upper, lower) *[variable2(upper, lower)…]*

Examples

```
model1.setbounds gdp(0,1000) unemp(un_lower, un_higher)
```

specifies a lower bound of 0 and 1,000 for GDP and the series bounds UN_LOWER and UN_HIGHER for UNEMP in model MODEL01.

Cross-references

See "Boundaries" on page 1227 of *User's Guide II* for details.

See also `Model::boundscheck` (p. 613).

settrace	Model Procs

Specify the endogenous variables to be traced when solving the model

Specifies the endogenous variables for which you wish to keep intermediate calculations at the next deterministic simulation. The intermediate results of all traced variables will be part of the model solution output. Tracing intermediate values may give you some idea of where to look for problems when a model is generating errors or failing to converge.

Syntax

model_name.**settrace** *[endogenous_list]*

If the *endogenous_list* of variables is omitted, `settrace` clears out the existing trace specification.

Examples

```
model1.trace gdp cons interest cpi
```

specifies that GDP, CONS, INTEREST, and CPI should be traced at the next simulation.

If you then issue the command:

```
mod1.settrace
```

EViews will clear the trace list.

Cross-references

See the discussion in "Diagnostics" on page 1226 of *User's Guide II*.

See also Model::trace (p. 644) and Model::track (p. 645).

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. 641). Stochastic solution options should be set using Model::stochastic (p. 643).

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 52. "Models," on page 1177 of *User's Guide II* for a discussion of models.

See also Model::model (p. 629), Model::msg (p. 630), Model::solveopt (p. 641), and Model::stochastic (p. 643).

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

s = *arg* *(default* = "d"*)*	Solution type: "*d*" (deterministic), "*m*" (stochastic – collect means only), "*s*" (stochastic – collect means and s.d.), "*b*" (stochastic – collect means and confidence bounds), "*a*" (stochastic – collect all; means, s.d. and confidence bounds).
d = *arg* *(default* = "d"*)*	Model solution dynamics: "d" (dynamic solution), "s" (static solution), "f" (fitted values – single equation solution).
struct = t	Ignore ARMA terms and use only the structural part of the equations when solving the model.
m = *integer* *(default* = 5000*)*	Maximum number of iterations for solution (maximum 100,000).
c = *number* *(default* = 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.
a = *arg* *(default* = "f"*)*	Alternate scenario solution: "t" (true - solve both active and alternate scenario and collect deviations for stochastic), "f" (false - solve only the active scenario).
o = *arg* *(default* = "g"*)*	Solution method: "g" (Gauss-Seidel), "n" (Newton), "b" (Broyden), "w" (E-Newton), "q" (E-QNewton).
i = *arg* *(default* = "a"*)*	Set initial (starting) solution values: "a" (actuals), "p" (values in period prior to start of solution period).
n = *arg* *(default* = "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.
e = *arg* *(default* = "t"*)*	Excluded variables initialized from actuals: "t" (true), "f" (false).

t = *arg* *(default = "u")*	Terminal condition for forward solution: "u" (user supplied - actuals), "l" (constant level), "d" (constant difference), "g" (constant growth rate).
w = *arg*	Solve direction: "t" (two-directional), "f" (forwards only).
g = *arg* *(default = = 7)*	Number of digits to round solution: an integer value (number of digits), "n" (do not roundoff).
z = *arg* *(default = = 1e-7)*	Zero value: a positive number below which the solution (absolute value) is set to zero, "n" (do not set to zero).
f = *arg* *(default = = "t")*	Order simultaneous blocks for minimum feedback: "t" (true), "f" (false).
v = *arg* *(default = = "f")*	Display verbose diagnostic messages: "t" (true), "f" (false).
j = *arg* *(default = = "a")*	Use analytic or numeric Jacobians: "a" (analytic), "n" (numeric only).

Cross-references

See Chapter 52. "Models," on page 1177 of *User's Guide II* for a discussion of models.

See also `Model::model` (p. 629), `Model::msg` (p. 630), and `Model::solve` (p. 640). Stochastic solution options should be set using `Model::stochastic` (p. 643).

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. 612), `Model::merge` (p. 628), `Model::printview` (p. 631).

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.

i = *arg* *(default* = "n"*)*	Innovation generation: "n" (normal random number) or "b" (bootstrap).
d = *arg* *(default* = "f"*)*	Diagonal covariance matrix (for bootstrap: draw resids 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 *pos_number*.
s = *quoted_sample*	Covariance estimation sample (Bootstrap residual draw sample). For example, s = "1970.1 2003.4"
r = *integer* *(default* = 1000*)*	Number of stochastic repetitions.
k	Calculate confidence interval from entire sample.
f = *number* *(default* = .02*)*	Fraction of failed repetitions before stopping.
b = *number* *(default* = .95*)*	Size of stochastic confidence intervals.
c = *arg* *(default* = "f"*)*	Include coefficient uncertainty: "t" (true), "f" (false).
p = *page_name*	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 52. “Models,” on page 1177 of *User’s Guide II* for a discussion of models. See `Model::innov` (p. 623) to set options on individual series in stochastic solution.

See also `Model::model` (p. 629), `Model::solve` (p. 640) and `Model::solveopt` (p. 641).

text	Model Views

Display text representation of the model specification.

Syntax

model_name.**text***(options)*

The `text` command is equivalent to `Model::spec` (p. 642).

Options

p	Print the model text specification.

Examples

```
model1.text
```

displays the text representation of the object MODEL1.

Cross-references

See Chapter 52. “Models,” on page 1177 of *User’s Guide II* for further details on models.

See also `Model::printview` (p. 631).

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 1226 of *User’s Guide II* for further details on tracing model solutions.

See also Model::msg (p. 630), Model::solve (p. 640) and Model::solveopt (p. 641).

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

```
model1.track gdp cons interest cpi
```

specifies that GDP, CONS, INTEREST, and CPI should be tracked at the next simulation.

```
model1.track @all
```

tracks all endogenous variables at the next simulation.

Cross-references

See also Model::model (p. 629) and Model::trace (p. 644).

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, the name of an estimation object, 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 52. "Models," on page 1177 of *User's Guide II* for a discussion of specifying and solving models in EViews. See also `Model::append` (p. 612), `Model::merge` (p. 628) and `Model::solve` (p. 640).

update	Model Procs

Update model specification.

Recompiles the model and updates links.

Syntax

model.**update** *[arg]*

Follow the name of the model object by a period the keyword `update`, and optionally a list of estimation objects to update. If no argument is provided, every object is updated.

Examples

```
mod1.update
```

recompiles and updates all of the links in MOD1.

```
mod1.update eq01 var01
```

updates the links to equation objectEQ01 and VAR object VAR01.

Cross-references

See Chapter 52. "Models," on page 1177 of *User's Guide II* for a discussion of specifying and solving models in EViews. See also `Model::append` (p. 612), `Model::merge` (p. 628) and `Model::solve` (p. 640).

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 1201 of *User's Guide II* for details. See Chapter 52. "Models," on page 1177 of *User's Guide II* for a general discussion of models.

See also `Model::block` (p. 612), `Model::text` (p. 644), and `Model::eqs` (p. 620) 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

pool ..... declare pool object (p. 675).

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

ls ..... estimate linear regression models including cross-section weighted least squares, and fixed and random effects models (p. 667).
tsls ..... linear two-stage least squares (TSLS) regression models (p. 685).

Pool Views

cellipse..... Confidence ellipses for coefficient restrictions (p. 652).
coefcov..... coefficient covariance matrix (p. 654).
coint ..... Johansen's cointegration test (p. 655).
describe..... calculate pool descriptive statistics (p. 659).
fixedtest ..... test significance of estimates of fixed effects (p. 664).
label..... label information for the pool object (p. 666).
output ..... table of estimation results (p. 674).
ranhaus..... Hausman test for correlation between random effects and regressors (p. 675).
representations ..... text showing equations in the model (p. 678).
residcor..... residual correlation matrix (p. 679).
residcov ..... residual covariance matrix (p. 679).
resids ..... table or graph of residuals for each pool member (p. 680).
results..... table of estimation results (p. 681).
sheet ..... spreadsheet view of series in pool (p. 682).
testadd ..... likelihood ratio test for adding variables to pool equation (p. 684).
testdrop..... likelihood ratio test for dropping variables from pool equation (p. 685).
uroot..... unit root test on a pool series (p. 689).
uroot2 ..... dependent (second generation panel) unit root tests on a pool series (p. 692).
wald..... Wald coefficient restriction test (p. 697).

Pool Procs

Pool Data Members

String Values

@attr("*arg*").....string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@command .....full command line form of the estimation command. Note this is a combination of `@method`, `@options` and `@spec`.
@crossids .....space delimited list of the Pool identifiers.
@crossidsest .....space delimited list of the Pool identifiers used in estimation.
@description.....string containing the Pool object's description (if available).
@detailedtype .....returns a string with the object type: "POOL".
@displayname .....returns the Pool's display name. If the Pool has no display name set, the name is returned.
@idname(i) .....*i*-th cross-section identifier.
@idnameest(i) .....*i*-th cross-section identifier for estimated equation.
@method.....command line form of estimation method ("LS", "TSLS", *etc.*...).

@name ..... returns the Pool's name.
@options..... command line form of pool estimation options.
@remarks ..... string containing the pool object's remarks (if available).
@smpl ..... description of sample used for estimation.
@spec..... original Pool estimation specification.
@type ..... returns a string with the object type: "POOL".
@updatetime..... returns a string representation of the time and date at which the Pool was last updated.

Scalar Values

@aic ..... Akaike information criterion.
@coefcov(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.
@npers ..... number of workfile periods used in estimation of the pool equation.
@r2 ..... R-squared statistic.
@rbar2..... adjusted R-squared statistic.
@regobs..... total number of observations in regression.
@schwarz..... Schwarz information criterion.
@sddep..... standard deviation of the dependent variable.
@se ..... standard error of the regression.
@ssr ..... sum of squared residuals.
@stderrs(i)..... standard error for coefficient i.
@totalobs..... total number of observations in the pool. For a balanced sample this is "@regobs*@ncrossest".
@tstats(i) ..... t-statistic value for coefficient i.
c(i)..... i-th element of default coefficient vector for the pool.

Vectors and Matrices

@coefcov ..... covariance matrix for coefficients of equation.
@coefs..... coefficient vector.

@effects.....vector of estimated fixed or random effects (only for fixed or random effects estimation).
@residcov.....(sym) covariance matrix of the residuals.
@stderrs.....vector of standard errors for coefficients.
@tstats.....vector of *t*-statistic values for coefficients.

Pool Examples

To read data using the pool object:

```
mypool1.read(b2) data.xls x? y? z?
```

To delete and store pool series you may enter:

```
mypool1.delete x? y?
mypool1.store z?
```

Descriptive statistics may be computed using the command:

```
mypool1.describe(m) z?
```

To estimate a pool equation using least squares and to access the *t*-statistics, enter:

```
mypool1.ls y? c z? @ w?
vector tstat1 = mypool1.@tstats
```

Pool Entries

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 1249 of *User's Guide II* for a discussion of pool identifiers.

See also `Pool::drop` (p. 661).

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

ind = *arg*	Specifies whether and how to draw the individual coefficient intervals. The default is "ind = line" which plots the individual coefficient intervals as dashed lines. "ind = none" does not plot the individual intervals, while "ind = shade" plots the individual intervals as a shaded rectangle.
size = *number* *(default* = 0.95*)*	Set the size (level) of the confidence ellipse. You may specify more than one size by specifying a space separated list enclosed in double quotes.
dist = *arg*	Select the distribution to use for the critical value associated with the ellipse size. The default depends on estimation object and method. If the parameter estimates are least-squares based, the $F(2, n-2)$ distribution is used; if the parameter estimates are likelihood based, the $\chi^2(2)$ distribution will be employed. "dist = f" forces use of the F-distribution, while "dist = c" uses the χ^2 distribution.
prompt	Force the dialog to appear from within a program.
p	Print the graph.

Examples

The two commands:

```
pool1.cellipse c(1), c(2), c(3)
pool1.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).

```
pool1.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 Intervals and Confidence Ellipses" on page 203 of *User's Guide II* for discussion.

See also `Pool::wald` (p. 697).

clearhist	Pool Procs

Clear the contents of the history attribute for pool objects.

Removes the pool's history attribute, as shown in the label view of the pool.

Syntax

pool_name.**clearhist**

Examples

```
p1.clearhist
p1.label
```

The first line removes the history from the pool P1, and the second line displays the label view of P1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Pool::label` (p. 666).

clearremarks	Pool Procs

Clear the contents of the remarks attribute.

Removes the pool's remarks attribute, as shown in the label view of the pool.

Syntax

pool_name.**clearremarks**

Examples

```
p1.clearremarks
p1.label
```

The first line removes the remarks from the pool P1, and the second line displays the label view of P1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Pool::label` (p. 666).

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. 26).

coint	Pool Views

Panel cointegration tests.

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:

ac = *arg* *(default* = "bt"*)*	Method of estimating the frequency zero spectrum: "bt" (Bartlett kernel), "pr" (Parzen kernel), "qs" (Quadratic Spectral kernel). Applicable to Pedroni and Kao tests.
band = *arg* *(default* = "nw"*)*	Method of selecting the bandwidth, where *arg* may be "nw" (Newey-West automatic variable bandwidth selection), or a number indicating a user-specified common bandwidth. Applicable to Pedroni and Kao tests.
lag = *arg*	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.
info = *arg* *(default* = "sic"*)*	Information criterion to use when computing automatic lag length selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn). Applicable to Pedroni and Kao tests.
maxlag = *int*	Maximum lag length to consider when performing automatic lag length selection, where *int* is an integer. The $$default = \text{int}(\min((T_i - k)/3, 12) \cdot (T_i/100)^{1/4})$$ where T_i is the length of the cross-section. Applicable to Pedroni and Kao tests.
disp = *arg* *(default* = 500*)*	Maximum number of individual results to be displayed.
prompt	Force the dialog to appear from within a program.
p	Print results.

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 "References" on page 1482 of *User's Guide II* for details on panel cointegration testing. See also `Pool::uroot` (p. 689).

copy	Pool Procs

Creates a copy of the pool.

Creates either a named or unnamed copy of the pool.

Syntax

pool_name.**copy**

pool_name.**copy** *dest_name*

Examples

```
p1.copy
```

creates an unnamed copy of the pool P1.

```
p1.copy p2
```

creates P2, a copy of the pool P1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

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 53. "Pooled Time Series, Cross-Section Data," on page 1247 of *User's Guide II* for a discussion of cross-section identifiers.

See also `Pool::add` (p. 651), `Pool::drop` (p. 661) and `Pool::pool` (p. 675).

delete	Pool Procs

Deletes series based upon identifiers in a pool.

Syntax

pool_name.**delete***(option) 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.

Options

prompt	Force the dialog to appear from within a program.

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 107 of *User's Guide I* for a discussion of working with objects, and Chapter 10. "EViews Databases," on page 333 of *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 *same observations*. If an observation is missing for any series, in any cross-section, it will be dropped for all cross-sections.

prompt	If no pool series are specified, force the dialog to appear from within a program.
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 53. "Pooled Time Series, Cross-Section Data," on page 1247 of *User's Guide II* for a discussion of the computation of these statistics, and a description of individual and balanced samples.

display

Pool Views

Display table, graph, or spool output in the pool object window.

Display the contents of a table, graph, or spool in the window of the pool object.

Syntax

pool_name.**display** *object_name*

Examples

```
pool1.display tab1
```

Display the contents of the table TAB1 in the window of the object POOL1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

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 123 of *User's Guide I* for a discussion of labels and display names.

See also `Pool::label` (p. 666).

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

```
crossc.drop jpn kor hk
```

drops the cross-section members JPN, KOR, and HK from the pool CROSSSC.

Cross-references

"Cross-section Identifiers" on page 1249 of *User's Guide II* discusses pool identifiers.

See also Pool::add (p. 651).

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.)
prompt	Force the dialog to appear from within a program.

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*	*High to low conversion methods removing NAs*: "a" (average of the nonmissing observations), "s" (sum of the nonmissing observations), "f" (first nonmissing observation), "l" (last nonmissing observation), "x" (maximum nonmissing observation), "m" (minimum nonmissing observation). *High to low conversion methods propagating NAs*: "an" or "na" (average, propagating missings), "sn" or "ns" (sum, propagating missings), "fn" or "nf" (first, propagating missings), "ln" or "nl" (last, propagating missings), "xn" or "nx" (maximum, propagating missings), "mn" or "nm" (minimum, propagating missings).

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 MYILLEG@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 333 of *User's Guide I* for a discussion of databases, databank files, and frequency conversion. Appendix A. "Wildcards," on page 1227 of the *Command and Programming Reference* describes the use of wildcard characters.

See also `Series::setconvert` (p. 834), `Pool::store` (p. 682), and `Pool::store` (p. 682).

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 "Fixed Effects Testing" on page 1354 of *User's Guide II* for discussion. See also `Pool::testadd` (p. 684), `Pool::testdrop` (p. 685), `Pool::ranhaus` (p. 675), and `Pool::wald` (p. 697).

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***(option)* ser_name = *expression*

You may use the cross section identifier "?" in the series name and/or in the expression on the right-hand side.

Options

prompt	Force the dialog to appear from within a program.

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 53. "Pooled Time Series, Cross-Section Data," on page 1247 of *User's Guide II* for a discussion of the computation of pools, and a description of individual and balanced samples.

See `Series::series` (p. 832) 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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User's Guide I* for a discussion of labels.

See also `Pool::displayname` (p. 661).

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. 564) of the *Command and Programming Reference*).
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 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".
numericderiv / -numericderiv	[Do / do not] use numeric derivatives only. If omitted, EViews will follow the global default.
fastderiv / -fastderiv	[Do / do not] use fast derivative computation. If omitted, EViews will follow the global default.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
cx = *arg*	Cross-section effects: (default) none, fixed effects ("cx = f"), random effects ("cx = r").
per = *arg*	Period effects: *(default)* none, fixed effects ("per = f"), random effects ("per = r").
wgt = *arg*	GLS weighting: *(default)* none, cross-section system weights ("wgt = cxsur"), period system weights ("wgt = persur"), cross-section diagonal weighs ("wgt = cxdiag"), period diagonal weights ("wgt = perdiag").
cov = *arg*	Coefficient covariance method: (default) ordinary, White cross-section system (period clustering) robust ("cov = cxwhite" or "cov = percluster"), White period system (cross-section clustering) robust ("cov = perwhite" or "cov = cxcluster"), White heteroskedasticity robust ("cov = stackedwhite"), White two-way cluster robust (cov = bothcluster"), 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").
keepwgts	Keep full set of GLS weights used in estimation with object, if applicable (by default, only small memory weights are saved).

rancalc = *arg* *(default* = "sa"*)*	Random component method: Swamy-Arora ("rancalc = sa"), Wansbeek-Kapteyn ("rancalc = wk"), Wallace-Hussain ("rancalc = wh").
nodf	Do not perform degree of freedom corrections in computing coefficient covariance matrix. The default is to use degree of freedom corrections.
b	Estimate using a balanced sample (pool estimation only).
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
iter = *arg* *(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.
unbalsur	Compute SUR factorization for unbalanced data using the subset of available observations in a cluster.
prompt	Force the dialog to appear from within a program.
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 20. "Basic Regression Analysis," on page 5 and Chapter 21. "Additional Regression Tools," on page 23 of *User's Guide II* discuss the various regression methods in greater depth.

See Chapter 53. "Pooled Time Series, Cross-Section Data," on page 1247 of *User's Guide II* for a discussion of pool estimation, and Chapter 55. "Panel Estimation," on page 1321 of *User's Guide II* for a discussion of panel equation estimation.

See Chapter 13. "Special Expression Reference," on page 323 of the *Command and Programming Reference* for special terms that may be used in `ls` specifications.

See also `Pool::tsls` (p. 685) 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, options) pool_series1 [pool_series2 pool_series3…]*

List the ordinary and pool series to be placed in the group. If specified, *group_name* should be the first option.

Options

prompt	Force the dialog to appear from within a program.

Examples

```
pool1.makegroup(g1) x? z y?
```

places the ordinary series Z, and all of the series represented by the pool series X? and Y?, in the group G1.

Cross-references

See "Making a Group of Pool Series" on page 1265 of *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

```
pool3.ls m1? gdp? tb3?
pool3.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 52. "Models," on page 1177 of *User's Guide II* for a discussion of specifying and solving models in EViews.

See also `Model::merge` (p. 628) and `Model::solve` (p. 640).

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

n = *arg*	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 "Residuals" on page 1283 of *User's Guide II*.

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.
prompt	Force the dialog to appear from within a program.

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

```
pool1.makestats gdp_? edu_? @ mean sd
```

computes the mean and standard deviation of the GDP_? and EDU_? series in each period (across the cross-section members) using the default common sample. The mean and standard deviation series will be named GDP_MEAN, EDU_MEAN, GDP_SD, and EDU_SD.

```
pool1.makestats(b) gdp_? @ max min
```

Computes the maximum and minimum values of the GDP_? series in each period using the balanced sample. The max and min series will be named GDP_MAX and GDP_MIN.

Cross-references

See Chapter 53. "Pooled Time Series, Cross-Section Data," on page 1247 of *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. 659).

makesystem	Pool Procs

Create system from a pool object.

Syntax

pool_name.**makesystem***(options)* y *[x1 x2 x3 ...] [***@cxeg** *w1 w2 ...] [***@inst** *z1 z2 ...] [***@cxinst** *z3 z4 ...]*

Creates a system out of the pool equation specification. Each cross-section in the pool will be used to form an equation. The pool variable y is the dependent variable. The *[x1 x2 x3 ...]* list consists of regressors with common coefficients in the system. The **@cxreg** list are regressors with different coefficients in each cross-section. The list of variables that follow **@inst** are the common instruments. The list of variables that follow **@cxinst** are the equation specific instruments.

Note that period specific coefficients and effects are not available in this routine.

Options

name = *name*	Specify name for the system object.
prompt	Force the dialog to appear from within a program.

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 43. "System Estimation," on page 897 of *User's Guide II* for a discussion of system objects in EViews.

olepush	Pool Procs

Push updates to OLE linked objects in open applications.

Syntax

pool_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

output	Pool Views

Display estimation output.

`output` changes the default object view to display the estimation output (equivalent to using `Pool::results` (p. 681)).

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. 681).

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. 651) and `Pool::drop` (p. 661).

Examples

```
pool zoo1 dog cat pig owl ant
```

Declares a pool object named ZOO1 with the listed cross-section identifiers.

Cross-references

See Chapter 53. "Pooled Time Series, Cross-Section Data," on page 1247 of *User's Guide II* for a discussion of working with pools in EViews.

See `Pool::add` (p. 651) and `Pool::drop` (p. 661). See also `Pool::ls` (p. 667) 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. 684), `Pool::testdrop` (p. 685), `Pool::fixedtest` (p. 664), and `Pool::wald` (p. 697).

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.

Note: we strongly recommend that you instead of using this proc, you use `wfopen` *or* `pageload` *to read the source data into a panel structured workfile and* `pageunstack` *if desired.*

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

prompt	Force the dialog to appear from within a program.

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 = *text*	Specify text for NAs. Default is "NA".
d = t	Treat tab as delimiter (note: you may specify multiple delimiter options). The *default* 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.
names	Series names provided in file.
label = *integer*	Number of lines between the header line and the data. Must be used with the "name" option.
rect *(default)* / norect	[Treat / Do not treat] file layout as rectangular.
skipcol = *integer*	Number of columns to skip. Must be used with the "*rect*" option.
skiprow = *integer*	Number of rows to skip. Must be used with the "rect" option.
comment = *symbol*	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.
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).
currency = *symbol*	Specify symbol/character for currency data.

Options for spreadsheet (Lotus, Excel) files

t	Read data organized by series. Default is to read by observation with series in columns.
letter_number (default = "b2"*)*	Coordinate of the upper-left cell containing data.
s = *sheet_name*	Sheet name for Excel 5–8 Workbooks.

Options for pool reading

bycross *(default)* / 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 "Creating a Workfile by Reading from a Foreign Data Source" on page 51 and "Importing Data" on page 152 of *User's Guide I* for a discussion and examples of importing data from external files.

Chapter 54. "Working with Panel Data," beginning on page 1297 of *User's Guide II* describes panel data alternatives to working with pooled data.

See also `pageload` (p. 546) and `wfopen` (p. 640) of the *Command and Programming Reference* and `Pool::write` (p. 698).

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.

Cross-references

See "Estimating a Pool Equation" on page 1268 of *User's Guide II* for a discussion of pool equations.

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. 679) and `Pool::makeresids` (p. 671).

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 "Estimating a Pool Equation" on page 1268 of *User's Guide II* for a discussion of pool equations.

See also `Pool::residcor` (p. 679) and `Pool::makeresids` (p. 671).

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 *(default)*	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. 671).

Cross-references

See "Estimating a Pool Equation" on page 1268 of *User's Guide II* for a discussion of pool equations.

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.

Cross-references

See "Estimating a Pool Equation" on page 1268 of *User's Guide II* for a discussion of pool equations.

setattr	Pool Procs

Set the object attribute.

Syntax

pool_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the "revised" attribute in the object A to the string "never", and extracts the attribute into the string object S.

Cross-references

See "Adding Custom Attributes in the Label View" on page 123 and "Adding Your Own Label Attributes" on page 70 of *User's Guide I*.

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

prompt	Force the dialog to appear from within a program.
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 53. "Pooled Time Series, Cross-Section Data," on page 1247 of *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

d = *db_name*	Store to the specified database.
i	Store to individual databank files.
1 / 2	Store series in [single / double] precision to save space.
o	Overwrite object in database (default is to merge data, where possible).
g = *arg*	Group store from workfile to database: "s" (copy group definition and series as separate objects), "t" (copy group definition and series as one object), "d" (copy series only as separate objects), "l" (copy group definition only).
prompt	Force the dialog to appear from within a program.

If you do not specify the precision option (1 or 2), the global option setting will be used. See "Database Storage Defaults" on page 1021 of *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 129 of *User's Guide I* discusses exporting data in other file formats. See Chapter 10. "EViews Databases," on page 333 of *User's Guide I* for a discussion of EViews databases and databank files.

For additional discussion of wildcards, see Appendix A. "Wildcards," on page 1227 of the *Command and Programming Reference*.

See also `Pool::fetch` (p. 662).

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

prompt	Force the dialog to appear from within a program.
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 Diagnostics" on page 203 of *User's Guide II* for further discussion.

See also `Pool::testdrop` (p. 685) and `Pool::wald` (p. 697).

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

prompt	Force the dialog to appear from within a program.
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 Diagnostics" on page 203 of *User's Guide II* for further discussion of testing coefficients.

See also `Pool::testadd` (p. 684) and `Pool::wald` (p. 697).

tsls	Pool Methods

Two-stage least squares.

Syntax

pool_name.ts**ls***(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.
numericderiv / -numericderiv	[Do / do not] use numeric derivatives only. If omitted, EViews will follow the global default.
fastderiv / -fastderiv	[Do / do not] use fast derivative computation. If omitted, EViews will follow the global default.
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. 564) of the *Command and Programming Reference*).
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".
cx = *arg*	Cross-section effects. For fixed effects estimation, use "cx = f"; for random effects estimation, use "cx = r".

per = *arg*	Period effects. For fixed effects estimation, use "cx = f"; for random effects estimation, use "cx = r".
wgt = *arg*	GLS weighting: *(default)* none, cross-section system weights ("wgt = cxsur"), period system weights ("wgt = persur"), cross-section diagonal weighs ("wgt = cxdiag"), period diagonal weights ("wgt = perdiag").
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").
keepwgts	Keep full set of GLS weights used in estimation with object, if applicable (by default, only small memory weights are saved).
rancalc = *arg* *(default* = "sa"*)*	Random component method: Swamy-Arora ("rancalc = sa"), Wansbeek-Kapteyn ("rancalc = wk"), Wallace-Hussain ("rancalc = wh").
nodf	Do not perform degree of freedom corrections in computing coefficient covariance matrix. The default is to use degree of freedom corrections.
coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default is to use the "C" coefficient vector.
iter = *arg* *(default* = "onec"*)*	Iteration control for GLS specifications: perform one weight iteration, then iterate coefficients to convergence ("iter = onec"), iterate weights and coefficients simultaneously to convergence ("iter = sim"), iterate weights and coefficients sequentially to convergence ("iter = seq"), perform one weight iteration, then one coefficient step ("iter = oneb"). Note that random effects models currently do not permit weight iteration to convergence.
s	Use the current coefficient values in "C" as starting values for equations with AR or MA terms (see also `param` (p. 564) of the *Command and Programming Reference*).

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".
unbalsur	Compute SUR factorization in unbalanced data using the subset of available observations for a cluster.
prompt	Force the dialog to appear from within a program.
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 91 and "Two-Stage Least Squares" on page 899 of *User's Guide II* for details on two-stage least squares estimation in single equations and systems, respectively.

"Instrumental Variables" on page 1291 of *User's Guide II* discusses estimation using pool objects, while "Instrumental Variables Estimation" on page 1324 of *User's Guide II* discusses estimation in panel structured workfiles.

See also `Pool::ls` (p. 667).

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. 26).

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:

exog = *arg* (*default* = "const")	Specification of exogenous trend variables in the test equation: "const" "trend" (include a constant and a linear time trend).
dif = *integer* (*default* = 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 *(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.

Sample Option

balance	Use balanced (across cross-sections or series) data when performing test.

Lag Difference Options

Specifies the number of lag difference terms to be included in the test equation. Applicable in "Summary", LLC, Breitung, IPS, and Fisher-ADF tests. The default setting depends on whether you choose to balance the samples across cross-sections.

If you do not include the "balance" option, the default is to perform automatic lag selection using the Schwarz criteria ("lagmethod = sic").

Alternately, if you include the "balance" option, the default setting is a common, observation-based fixed lag ("lag = *default*") where:

$$default = \begin{cases} 1 \text{ if } (T_{\min} \le 60) \\ 2 \text{ if } (60 < T_{\min} \le 100) \\ 4 \text{ if } (T_{\min} > 100) \end{cases} \tag{1.3}$$

lagmethod = *arg* (*default* = "sic")	Method for selecting lag lengths (number of first difference terms) to be included in the Dickey-Fuller test regressions: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn), "tstat" (Ng-Perron first backward significant *t*-statistic).
lag = *arg*	Specified lag length (number of first difference terms) to be included in the regression: *integer* (user-specified common lag length), *vector_name* (user-specific individual lag length, one row per cross-section).
maxlag = *arg*	Maximum lag length to consider when performing automatic lag length selection: *integer* (common maximum lag length), or *vector_name* (individual maximum lag length, one row per cross-section). The default setting produces individual maximum lags of, $default = \text{int}(\min(12,\ T_i/3) \cdot (T_i/100)^{1/4})$ where T_i is the length of the cross-section.
lagpval = *arg* (*default* = 0.1)	Probability value for use in the *t*-statistic automatic lag selection method (when "lagmethod = tstat").

Kernel Options

Specifies options for computing kernel estimates of the zero-frequency spectrum (long-run covariance). Applicable to "Summary", LLC, Fisher-PP, and Hadri tests.

hac = *arg* (*default* = "bt")	Method of estimating the frequency zero spectrum: "bt" (Bartlett kernel), "pr" (Parzen kernel), "qs" (Quadratic Spectral kernel),
band = *arg*, b = *arg* (*default* = "nw")	Method of selecting the bandwidth: "nw" (Newey-West automatic variable bandwidth selection), "a" (Andrews automatic selection), *number* (user-specified common bandwidth), *vector_name* (user-specified individual bandwidths, one row for each cross-section).

Other options

prompt	Force the dialog to appear from within a program.
p	Print output from the test.

Examples

```
Pool1.uroot(llc,exog=trend) gdp?
```

performs the LLC panel unit root test with exogenous individual trends and individual effects on pool series GDP?

```
Pool1.uroot(ips,exog=const,maxlag=4,lagmethod=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,exog=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 Testing" on page 773 of *User's Guide II* for discussion of standard unit root tests performed on a single series, and "Cross-sectionally Independent Panel Unit Root Testing" on page 811 and "Cross-sectionally Dependent Panel Unit Root Tests" on page 822 of *User's Guide II* for discussion of unit roots tests performed on panel structured workfiles, groups of series, or pooled data.

See also `Pool::uroot2` (p. 692), `Series::uroot` (p. 863), `Series::uroot2` (p. 868), `Group::uroot` (p. 512), `Group::uroot2` (p. 515).

uroot2	Pool Views

Compute dependent (second generation) panel unit root tests on a group of series.

Syntax

pool_name.**uroot2***(options) pool_series*

where *group_name* is the name of a pool object and *pool_series* is a pool series.

Options

General Options

type = *arg* (*default* = "panic")	Type of unit root test: PANIC - Bai and Ng (2004) ("panic"), CIPS - Pesaran (2007) ("cips"). Note: (1) when performing PANIC testing, factor selection, MQ, ADF lag selection, VAR lag selection (possibly), long-run variance (possibly), and *p*-value simulation options are relevant. (2) when perform CIPS testing, ADF lag selection options are relevant.
exog = *arg* (*default* = "constant")	Exogenous deterministic variables to include for each cross-section: "none" (no deterministic variables), "constant" (only a constant), "trend" (both a constant and trend).
prompt	Force the dialog to appear from within a program.
p	Print results.

ADF Lag Selection Options

adflagmethod = *arg* *(default* = "sic"*)*	Method for selecting lag length (number of first difference terms) to be included in the Dickey-Fuller test regression or number of lags in the AR spectral density estimator: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn), "msaic" (Modified Akaike), "msic" (Modified Schwarz), "mhqc" (Modified Hannan-Quinn), "tstat" (Ng-Perron first backward significant t-statistic).
adflag = *integer*	Use-specified fixed lag.
adfmaxlag = *integer*	Maximum lag length to consider when performing automatic lag length selection. Note: default is Schwert's rule: let $$L(k) = k(T/100)^{1/4}$$ for $k \in \{2, 4, 6, 8, 10, 12\}$ and let $T^* = \min(N, T)$; then the default maximum lag is given by $$L^* = \max_k\{L(k) \mid L(k) < T^*\}$$
adflagpval = *arg* *(default* = 0.1*)*	Probability value for use in the t-statistic automatic lag selection method ("lagmethod = tstat").

PANIC Number of Factor Selection Options

fsmethod = *arg* *(default* = "bn"*)*	Factor retention selection method: "bn" (Bai and Ng (2002)), "ah" (Ahn and Horenstein (2013)), "simple" (simple eigenvalue methods), "user" (user-specified value). Note the following: (1) If using simple methods, the minimum eigenvalue and cumulative proportions may be specified using "minigen = " and "cproport = ". (2) If setting "fsmethod = user" to provide a user-specified value, you must specify the value with "r = ".
r = *arg* *(default* = 1*)*	User-specified number of factors to retain (for use when "fsmethod = user").
mineigen = *arg* *(default* = 0*)*	Minimum eigenvalue to retain factor (when "fsmethod = simple").
cproport = *arg* *(default* = 1.0*)*	Cumulative proportion of eigenvalue total to attain (when "fsmethod = simple").

mfmethod = *arg*	Maximum number of factors used by selection methods: "schwert" (Schwert's rule, *default*), "ah" (Ahn and Horenstein's (2013) suggestion), "rootsize" ($\min(\sqrt{N}, \sqrt{T})$), "size" ($\min(N, T)$), "user" (user specified value). (1) For use with all factor retention methods apart from user-specified ("fsmethod = user"). (2) If setting "mfmethod = user", you may specify the maximum number of factors using "rmax = ". (3) Schwert's rule sets the maximum number of factors using the rule: let $$L(k) = k(T/100)^{1/4}$$ for $k \in \{2, 4, 6, 8, 10, 12\}$ and let $T^* = \min(N, T)$; then the default maximum lag is given by $$L^* = \max_k\{L(k) \mid L(k) < T^*\}$$
rmax = *arg* *(default* = all*)*	User-specified maximum number of factors to retain (for use when "mfmethod = user").
fsic = *arg (default* = avg*)*	Factor selection criterion when "fsmethod = bn": "icp1" (ICP1), "icp2" (ICP2), "icp3" (ICP3), "pcp1" (PCP1), "pcp2" (PCP1), "pcp3" (ICP3), "avg" (average of all criteria ICP1 through PCP3). Factor selection criterion when "fsmethod = ah": "er" (eigenvalue ratio), "gr" (growth ratio), "avg" (average of eigenvalue ratio and growth ratio). Factor selection when "fsmethod = simple": "min" (minimum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "max" (maximum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "avg" (average the optimal number of factors as specified by the min and max rule, then round to the nearest integer).
demeantime	Demeans observations across time prior to factor selection procedures.
sdizetime	Standardizes observations across time prior to factor selection procedures.
demeancross	Demeans observations across cross-sections prior to factor selection procedures.
sdizecross	Standardizes observations across cross-sections prior to factor selection procedures.

PANIC VAR Lag Selection Options

For use when computing a PANIC test with MQ_f statistic.

varlagmethod = *arg* *(default* = "sic"*)*	Method for selecting lag length (number of first difference terms) to be included in the test statistic VAR: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn), "msaic" (Modified Akaike), "msic" (Modified Schwarz), "mhqc" (Modified Hannan-Quinn), "tstat" (Ng-Perron first backward significant t-statistic).
varlag = *integer*	Use-specified fixed lag.
varmaxlag = *integer*	Maximum lag length to consider when performing automatic lag length selection. Note: default is Schwert's rule: let $$L(k) = k(T/100)^{1/4}$$ for $k \in \{2, 4, 6, 8, 10, 12\}$ and let $T^* = \min(N, T)$; then the default maximum lag is given by $$L^* = \max_k \{L(k) \mid L(k) < T^*\}$$

PANIC Long-run Variance Options

For use when computing a PANIC test using the MQ_c statistic.

Whitening Options

lag = *arg*	Lag specification: *integer* (user-specified number of lags), "a" (automatic selection).
infosel = *arg* *(default* = "aic"*)*	Information criterion for automatic selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn) (if "lag = a").
maxlag = *integer*	Maximum lag-length for automatic selection *(optional)* (if "lag = a"). The default is an observation-based maximum of $T^{1/3}$.

Kernel Options

kern = *arg* *(default* = "bart"*)*	Kernel shape: "none" (no kernel), "bart" (Bartlett), "bohman" (Bohman), "daniell" (Daniel), "parzen" (Parzen), "parzriesz" (Parzen-Riesz), "parzgeo" (Parzen-Geometric), "parzcauchy" (Parzen-Cauchy), "quadspec" (Quadratic Spectral), "trunc" (Truncated), "thamm" (Tukey-Hamming), "thann" (Tukey-Hanning), "tparz" (Tukey-Parzen), "user" (User-specified; see "kernwgt = " below).
kernwgt = *vector*	User-specified kernel weight vector (if "kern = user").
bw = *arg* *(default* = "nwfixed"*)*	Bandwidth: "fixednw" (Newey-West fixed), "andrews" (Andrews automatic), "neweywest" (Newey-West automatic), *number* (User-specified bandwidth).
nwlag = *integer*	Newey-West lag-selection parameter for use in nonparametric bandwidth selection (if "bw = neweywest").
bwoffset = *integer* *(default* = 0*)*	Apply integer offset to bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").
bwint	Use integer portion of bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").

PANIC p-value Options

mcreps = *integer*	Number of Monte Carlo replications.
asymplen = *integer*	Asymptotic length of series.
seed = *number*	Specifies the random number generator seed
rng = *arg*	Specifies the type of random number generator. The key can be; 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").

Examples

```
pool01.uroot2 ser?
```

The line above performs a PANIC unit root test on the series in the pool series SER?.

```
pool01.uroot2(fsmethod=AH, mq=mqf, varlag=3)
```

The line above performs a PANIC unit root test using Ahn and Horenstein (2013) for factor selection determination and the MQ_f test for the number of common trends using a VAR(3) model.

```
pool01.uroot2(test=cips, exog=trend, adnfosel=sic)
```

The line above performs a CIPS unit root test on the pool series SER?, with ADF testing performed on each cross-section with a constant and trend, and ADF lag selection using the Schwarz criterion.

Cross-references

See "Unit Root Testing" on page 773 of *User's Guide II* for discussion of standard unit root tests performed on a single series, and "Cross-sectionally Independent Panel Unit Root Testing" on page 811 and "Cross-sectionally Dependent Panel Unit Root Tests" on page 822 of *User's Guide II* for discussion of unit roots tests performed on panel structured workfiles, groups of series, or pooled data.

See also `Pool::uroot` (p. 689), `Series::uroot` (p. 863), `Series::uroot2` (p. 868), `Group::uroot` (p. 512), `Group::uroot2` (p. 515).

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

prompt	If no *restrictions* are specified, force the dialog to appear from within a program.
p	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 210 of *User's Guide II* for a discussion of Wald tests.

See also `Pool::cellipse` (p. 652), `Pool::testdrop` (p. 685), `Pool::testadd` (p. 684).

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.

Note: we strongly recommend that you instead of using this proc, you use `pagestack` (p. 555) *to create a panel structured workfile and then use* `wfsave` (p. 656) *or* `pagesave` (p. 548) *(*all in *Command and Programming Reference).*

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

prompt	Force the dialog to appear from within a program.

Other options are used to specify the format of the output file.

File type

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

If you omit the "*t* = " option, EViews will determine the type based on the file extension. Unrecognized extensions will be treated as ASCII files. For Lotus and Excel spreadsheet files specified without the "*t* = " option, EViews will automatically append the appropriate extension if it is not otherwise specified.

ASCII text files

na = *string*	Specify text string for NAs. Default is "NA".
names (*default)* / nonames	[Write / Do not write] series names.
id	Write dates/obs and cross-section identifiers.
d = *arg*	Specify delimiter (*default* is tab): "s" (space), "c" (comma).
t	Write by series. Default is to write by obs with series in columns.

Spreadsheet (Lotus, Excel) files

letter_number	Coordinate of the upper-left cell containing data.
names *(default)* / nonames	[Write / Do not write] series names.
id	Write dates/obs and cross-section identifiers.
dates = *arg*	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).
t	Write by series. Default is to write by obs with series in columns.

Pooled data writing

bycross *(default)* / byper	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 Data," beginning on page 169 of *User's Guide I* for a discussion. Pool writing is discussed in "Exporting Pooled Data" on page 1266 of *User's Guide II*.

See also `pagesave` (p. 548) of the *Command and Programming Reference* and `Pool::read` (p. 676).

Rowvector

Row vector. (One dimensional array of numbers).

Rowvector Declaration

rowvector.....declare rowvector object (p. 727).

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 rowvector will be sized and initialized accordingly:

```
rowvector(10) y=3
rowvector z=results
```

Rowvector Views

display.....display table, graph, or spool in object window (p. 706).
edftest.....empirical distribution function tests (p. 709).
freq.....one-way tabulation (p. 715).
hist.....descriptive statistics and histogram (p. 716).
label.....label information for the rowvector (p. 723).
sheet.....spreadsheet view of the rowvector (p. 731).
stats.....descriptive statistics of the elements of the rowvector (p. 735).
teststat.....simple hypothesis tests (p. 733).

Rowvector Procs

clearcollabels.....clear the column labels in a rowvector object (p. 704).
clearhist.....clear the contents of the history attribute (p. 704).
clearremarks.....clear the contents of the remarks attribute (p. 705).
clearrowlabels.....clear the row labels in a rowvector object (p. 705).
copy.....creates a copy of the rowvector (p. 705).
displayname.....set display name (p. 706).
distdata.....save a matrix containing distribution plot data computed from the rowvector (p. 707).
export.....export rowvector as Excel 2007 XLSX, CSV, tab-delimited ASCII text, RTF, HTML, Enhanced Metafile, PDF, TEX, or MD file on disk (p. 710).
fill.....fill elements of the rowvector (p. 714).

import ..... imports data from a foreign file into the rowvector object (p. 717).
olepush ..... push updates to OLE linked objects in open applications (p. 724).
read ..... (deprecated) import data from disk (p. 724).
resize ..... resize the rowvector object (p. 726).
setattr ..... set the value of an object attribute (p. 727).
setcollabels ..... set the column labels in a rowvector object (p. 728).
setformat ..... set the display format for the rowvector spreadsheet (p. 728).
setindent ..... set the indentation for the rowvector spreadsheet (p. 729).
setjust ..... set the horizontal justification for all cells in the spreadsheet view of the rowvector object (p. 730).
setrowlabels ..... set the row label in a rowvector object (p. 730).
setwidth ..... set the column width in the rowvector spreadsheet (p. 731).
showlabels ..... displays the custom row and column labels of a rowvector spreadsheet (p. 732).
write ..... export data to disk (p. 734).

Rowvector Graph Views

Graph creation views are discussed in detail in "Graph Creation Command Summary" on page 1267.

bar ..... bar graph of each column (element) of the data against the row index (p. 1275).
boxplot ..... boxplot graph (p. 1279).
distplot ..... distribution graph (p. 1283).
dot ..... dot plot graph (p. 1290).
errbar ..... error bar graph view (p. 1294).
mixed ..... mixed-type graph (p. 1301).
pie ..... pie chart view (p. 1304).
qqplot ..... quantile-quantile graph (p. 1306).
scat ..... scatter diagrams of the columns of the rowvector (p. 1310).
scatmat ..... matrix of all pairwise scatter plots (p. 1315).
scatpair ..... scatterplot pairs graph (p. 1318).
spike ..... spike graph (p. 1323).
xybar ..... XY bar graph (p. 1330).
xypair ..... XY pairs graph (p. 1337).

Rowvector Data Members

String values

@attr("*arg*") ..... string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@collabels ..... string containing the column labels of the rowvector.

@description.....string containing the Rowvector object's description (if available).
@detailedtype.....string with the object type: "ROWVECTOR".
@displayname.....string containing the Rowvector object's display name. If the Rowvector has no display name set, the name is returned.
@name.....string containing the Rowvector object's name.
@remarks.....string containing the Rowvector object's remarks (if available).
@rowlabels.....string containing the row label of the rowvector.
@type.....string with the object type: "ROWVECTOR".
@updatetime.....string representation of the time and date at which the Rowvector was last updated.

Scalar values

(i).....*i*-th element of the rowvector. Simply append "(i)" to the rowvector name (without a ".").
@cols.....number of columns in the rowvector.

Rowvector values

@col(*arg*).....Returns the rowvector defined by *arg*. *arg* may be an integer, vector of integers, string, or svector of strings. Integers correspond to column numbers so that, for example, *arg* = 2 specifies the second column. Strings correspond to column labels so that *arg* = "2" specifies the first column labeled "2".
@dropcol(*arg*).....Returns the rowvector with the columns defined by *arg* removed. *arg* may be an integer, vector of integers, string, or svector of strings. Integers correspond to column numbers so that, for example, *arg* = 2 specifies the second column. Strings correspond to column labels so that *arg* = "2" specifies the first column labeled "2".
@icol(*arg*).....Returns the indices for the columns defined by *arg* where *arg* is a string or svector of strings. The strings correspond to column labels so that *arg* = "2" specifies the first column labeled "2".
@t.....Returns transpose.

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 rowvector 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.

clearcollabels	Rowvector Procs

Clear the column labels in a rowvector object.

Syntax

rowvector_name.**clearcollabels**

Examples

```
rvec1.clearcollabels
```

clears the custom column labels from the rowvector RVEC1.

Cross-references

See also `Rowvector::clearrowlabels` (p. 705).

clearhist	Rowvector Procs

Clear the contents of the history attribute.

Removes the rowvector's history attribute, as shown in the label view of the rowvector.

Syntax

rowvector_name.**clearhist**

Examples

```
r1.clearhist
r1.label
```

The first line removes the history from the rowvector R1, and the second line displays the label view of R1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Rowvector::label` (p. 723).

clearremarks	Rowvector Procs

Clear the contents of the remarks attribute.

Removes the rowvector's remarks attribute, as shown in the label view of the rowvector.

Syntax

rowvector_name.**clearremarks**

Examples

```
r1.clearremarks
r1.label
```

The first line removes the remarks from the rowvector R1, and the second line displays the label view of R1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Rowvector::label` (p. 723).

clearrowlabels	Rowvector Procs

Clear the row label in a rowvector object.

Syntax

rowvector_name.**clearrowlabels**

Examples

```
rvec1.clearrowlabels
```

clears the custom row label from the rowvector RVEC1.

Cross-references

See also `Rowvector::clearcollabels` (p. 704).

copy	Rowvector Procs

Creates a copy of the rowvector.

Creates either a named or unnamed copy of the rowvector.

Syntax

rowvector_name.**copy**

rowvector_name.**copy** *dest_name*

Examples

```
r1.copy
```

creates an unnamed copy of the rowvector R1.

```
r1.copy r2
```

creates R2, a copy of the rowvector R1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

display	Rowvector Views

Display table, graph, or spool output in the rowvector object window.

Display the contents of a table, graph, or spool in the window of the rowvector object.

Syntax

rowvector_name.**display** *object_name*

Examples

```
rowvector1.display tab1
```

Display the contents of the table TAB1 in the window of the object ROWVECTOR1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

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 123 of *User's Guide I* for a discussion of labels and display names.

See also `Rowvector::label` (p. 723).

distdata	Rowvector Procs

Save a matrix containing distribution plot data computed from the rowvector.

Saves the data used to display a histogram, kernel density, theoretical distribution, empirical CDF or survivor plot, or quantile plot to the workfile.

Syntax

rowvector_name.**distdata**(**dtype** = *dist_type, dist_options*) *matrix_name*

saves the distribution plot data specified by *dist_type*, where *dist_type* must be one of the following keywords:

hist	Histogram *(default).*
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.
theoryqq	Theoretical quantile-quantile plot.

Options

The theoretical quantile-quantile plot type "theoryqq" takes the options described in `qqplot` (p. 1306) under "Theoretical Options" on page 1308.

For the remaining types, *dist_options* are any of the distribution type-specific options described in `distplot` (p. 1283).

Note that the graph display specific options such as "fill," "nofill," and "leg," and "noline" are not relevant for this procedure.

You may use the "prompt" option to force the dialog display

prompt	Force the dialog to appear from within a program.

Examples

```
rvec1.distdata(dtype=hist, anchor=0, scale=dens, rightclosed)
   matrix01
```

creates the data used to draw a histogram from the rowvector RVEC1 with the anchor at 0, density scaling, and right-closed intervals, and stores that data in a matrix called MATRIX01 in the workfile.

```
rvec1.distdata(dtype=kernel, k=b, ngrid=50, b=.5) matrix02
```

generates the kernel density data computed with a biweight kernel at 50 grid points, using a bandwidth of 0.5 and linear binning, and stores that data in MATRIX02.

```
rvec1.distdata(dtype=theoryqq, q=o, dist=logit, p1=.5) matrix03
```

creates theoretical quantile-quantile data from RVEC1 using the ordinary quantile method to calculate quantiles. The theoretical distribution is the logit distribution, with the location parameter set to 0.5. The data is saved into the matrix MATRIX03.

Cross-references

For a description of distribution graphs and quantile-quantile graphs, see "Analytical Graph Types," on page 836 of *User's Guide I*.

See also `distplot` (p. 1283) and `qqplot` (p. 1306).

edftest	Rowvector Views

Computes goodness-of-fit tests based on the empirical distribution function.

Compute Kolmogorov-Smirnov, Lilliefors, Cramer-von Mises, Anderson-Darling, and Watson empirical distribution function tests.

Syntax

rowvector_name.**edftest***(options)*

Options

General Options

dist = *arg* *(default* = "nomal"*)*	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).
p1 = *number*	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.
p2 = *number*	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.
p3 = *number*	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.
prompt	Force the dialog to appear from within a program.
p	Print test results.

Estimation Options

The following options apply if iterative estimation of parameters is required:

b	Use Berndt-Hall-Hall-Hausman (BHHH) algorithm. The default is Marquardt.
m = *integer*	Maximum number of iterations.

c = *number*	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
s	Take starting values from the C coefficient vector. By default, EViews uses distribution specific starting values that typically are based on the method of the moments.

Examples

The command

```
rvec1.edftest
```

uses the default settings to test whether the elements of the rowvector RVEC1 come from a normal distribution. Both the location and scale parameters are estimated from the data in RVEC1.

```
rvec1.edftest(type=chisq)
```

tests whether the data in RVEC1 follow a χ^2 distribution with degrees-of-freedom estimated from the data.

```
freeze(tab1) rvec1.edftest(type=chisq, p1=5)
```

tests whether the data in RVEC1 comes from a χ^2 distribution with 5 degrees-of-freedom. The output is stored in the table object TAB1.

Cross-references

See "Empirical Distribution Tests" on page 465 of *User's Guide I* for a description of the goodness-of-fit tests.

See also qqplot (p. 1306).

export	Rowvector Procs

Export rowvector to disk as an Excel 2007 XLSX, CSV, tab-delimited ASCII text, RTF, HTML, Enhanced Metafile, LaTeX, PDF, or Markdown file.

Syntax

vector_name.**export***(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.

The base syntax for writing Excel 2007 files is:

vector_name.**export**(*options*) *[path\]file_name [table_description]*

where the *table_description* may contain:

- "range = *arg*", where arg is top left cell of the destination 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.

Options

t = *file_type* *(default* = "csv"*)*	Specifies the file type, where *file_type* may be one of: "excelxml" (Excel 2007 (xml)),"csv" (CSV - comma-separated), "rtf" (Rich-text format), "txt" (tab-delimited text), "html" (HTML - Hypertext Markup Language), "emf" (Enhanced Metafile), "pdf" (PDF - Portable Document Format), "tex" (LaTeX), or "md" (Markdown). Files will be saved with the ".xlsx", ".csv", ".rtf", ".txt", ".htm", ".emf", ".pdf", ".tex", or ".md" extensions, respectively.
s = *arg*	Scale size, where *arg* is from 5 to 200, representing the percentage of the original table size (only valid for HTML or RTF files).
n = *string*	Replace all cells that contain NA values with the specified string. "NA" is the default.
h / -h	Include(/do not include) column and row headers. The default is to not include the headers
prompt	Force the dialog to appear from within a program.

PDF Options

landscape	Save in landscape mode (the default is to save in portrait mode).
size = *arg* (*default* = "letter")	Page size: "letter", "legal", "a4", and "custom".
width = *number* (*default* = 8.5)	Page width in inches if "size = custom".
height = *number* (*default* = 11)	Page height in inches if "size = custom".
leftmargin = *number (default* = 0.5*)*	Left margin width in inches.
rightmargin = *number (default* = 0.5*)*	Right margin width in inches.
topmargin = *number (default* = 1*)*	Top margin width in inches.
bottommargin = *number (default* = 1*)*	Bottom margin width in inches.

LaTeX Options

texspec / -texspec	[Include / Do not include] the full LaTeX documentation specification in the LaTeX output. The default behavior is taken from the global default settings.

Excel Options

mode = *arg*	Specify whether to create a new file, overwrite an existing file, or update an existing file. arg may be "create" (create new file only; error on attempt to overwrite) or "update" (update an existing file, only overwriting the area specified by the range = *table_description*). If the "mode = " option is not used, EViews will create a new file, unless the file already exists in which case it will overwrite it. Note that the "mode = update" option is only available for Excel in 1) Excel versions through 2003, if Excel is installed, and 2) Excel 2007 (xml). Note: Excel does not need to be installed for Excel 2007 writing.

Excel 2007 Options

mode = *arg*	Specify whether to create a new file, overwrite an existing file, or update an existing file. arg may be "create" (create new file only; error on attempt to overwrite) or "update" (update an existing file, only overwriting the area specified by the range = *table_description*). If the "mode = " option is not used, EViews will create a new file, unless the file already exists in which case it will overwrite it. Note that the "mode = update" option is only available for Excel in 1) Excel versions through 2003, if Excel is installed, and 2) Excel 2007 (xml). Note: Excel does not need to be installed for Excel 2007 writing.
cellfmt = *arg*	Specify whether to use EViews, pre-existing, or remove cell formatting (colors, font, number formatting when possible, column widths and row heights) for the written range. *arg* may be "eviews" (replace current formatting in the file with the same cell formatting in EViews), "preserve" (leave current cell formatting already in the Excel file), or "clear" (remove current formatting and do not replace).
strlen = *arg* (default = 256)	Specify the maximum the number of characters written for cells containing text. Strings in cells which are longer the max, will be truncated.

Examples

The command:

```
rvector1.export myrvector
```

exports data in RVECTOR1 to a CSV file named "myrvector.CSV" in the default directory.

```
rvector1.export(h,t=csv, n="NaN") myrvector
```

saves the contents of RVECTOR1 along with the column and row headers to a CSV (comma separated value) file named "myrvector.CSV" and writes all NA values as "NaN".

```
rvector1.export(h,t=html, s=50) myrvector
```

writes the data of RVECTOR1 along with the column and row headers to a HTML file named "myrvector.HTM" at half of the original size.

```
rvector1.export(n=".", r=B) myrvector
```

exports the data in the second column to a CSV file named "myrvector.CSV", and writes all NA values as ".".

```
rvector1.export(t=excelxml, cellfmt=clear, mode=update) myrvector
    range=Country!b5
```

writes the data in RVECTOR1 to the preexisting "myrvector.XLSX" Excel file to the "Country" sheet at cell B5, where all cell formatting is cleared.

Cross-references

See also Rowvector::import (p. 717).

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 "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 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,l) -1
```

Cross-references

See Chapter 11. "Matrix Language," on page 279 of *User's Guide II* for a detailed discussion of rowvector and matrix manipulation in EViews.

freq	Rowvector Views

Compute frequency tables.

The `freq` command performs a one-way frequency tabulation.

Frequencies are computed for all of the elements in the rowvector. Missing values are dropped unless included by option. You may use options to control automatic binning (grouping) of values and the order of the entries of the table.

Syntax

rowvector_name.**freq***(options)*

Options

dropna *(default)* / keepna	[Drop/Keep] NA as a category.
v = *integer* *(default* = 1000*)*	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*	(optional) Make bins if average count per distinct value is less than the specified number.
b = *integer* *(default* = 50*)*	Maximum number of categories to bin into if performing automatic binning.
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.

sort = *arg* *(default* = "lohi"*)*	Sort order for entries in the frequency table: high data value to low ("hilo"), low data value to high ("lohi" – *default*), high frequency to low ("freqhilo"), low frequency to high ("freqlohi").
prompt	Force the dialog to appear from within a program.
p	Print the table.

Examples

```
rvec1.freq(nov, noa)
```

tabulates all values (no binning) of RVEC1, with entries in ascending value order. The table will display counts, percentages, and cumulative frequencies.

```
rvec1.freq(v=200, b=50, keepna, noa)
```

tabulates RVEC1 including NAs. The observations will be binned if RVEC1 has more than 200 distinct values; EViews will create at most 50 equal value-width bins. The number of bins may be smaller than 50.

```
rvec1.freq(sort=freqhilo)
```

tabulates RVEC1 with the table rows ordered from values with highest frequency to lowest.

Cross-references

See "One-Way Tabulation" on page 467 of *User's Guide I* for a discussion of frequency tables.

hist — Rowvector Views

Histogram and descriptive statistics of a rowvector.

The `hist` command displays descriptive statistics and a histogram for the data in the rowvector.

Syntax

rowvector_name.**hist***(options)*

Options

p	Print the histogram.

Examples

```
rvec1.hist
```

Displays the histogram and descriptive statistics of RVEC1.

Cross-references

See "Histogram and Stats" on page 450 of *User's Guide I* for a discussion of the descriptive statistics reported in the histogram view.

See `distplot` (p. 1283) for a more fully-featured and customizable method of constructing histograms and `Rowvector::stats` (p. 735) stats for a view with a more extensive set of basic descriptive statistics.

import	Rowvector Procs

Imports data from a foreign file into the rowvector object.

Syntax

rowvector_name.**import***([type =]) source_description import_specification*

- *source_description* should contain a description of the file from which the data is to be imported. The specification of the description is usually just the path and file name of the file, however you can also specify more precise information. See `wfopen` (p. 640) of the *Command and Programming Reference* for more details on the specification of *source_description*.
- The optional "type = " option may be used to specify a source type. For the most part, you should not need to specify a "type = " option as EViews will automatically determine the type from the filename. The following table summaries the various source formats and along with the corresponding "type = " keywords:

	Option Keywords
Excel (through 2003)	"excel"
Excel 2007 (xml)	"excelxml"
HTML	"html"
Text / ASCII	"text"

- *import_specification* can be used to provide additional information about the file to be read. The details of import_specification will depend upon the type of file being imported.

Excel Files

The syntax for reading Excel files is:

rowvector_name.**import**(**type = excel***[xml]*) *source_description [table_description] [variables_description]*

The following *table_description* elements may be used when reading Excel data:

- "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.

- "byrow", transpose the incoming data. This option allows you to read files where the series are contained in rows (one row per series) rather than columns.

The optional *variables_description* may be formed using the elements:

- "colhead = *int*", number of table rows to be treated as 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 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 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.

Excel Examples

```
rowvec_obj.import "c:\data files\data.xls"
```

loads the active sheet of "Data.XLSX" into the ROWVEC_OBJ rowvector object.

```
rowvec_obj.import "c:\data files\data.xls" range="GDP data"
```

reads the data contained in the "GDP data" sheet of "Data.XLS" into the ROWVEC_OBJ object.

HTML Files

The syntax for reading HTML pages is:

rowvector_name.**import(type = html)** *source_description [table_description] [variables_description]*

The following *table_description* elements may be used when reading an HTML file or page:

- "table = *arg*", where *arg* specifies which HTML 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.

- "byrow", transpose the incoming data. This option allows you to import files where the series are contained in rows (one row per series) rather than columns.

The optional *variables_description* may be formed using the elements:

- "colhead = *int*", number of table rows to be treated as 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.

HTML Examples

```
rvec.import "c:\data.html"
```

loads into the RVEC object the data located in the HTML file "Data.HTML" located on the C:\ drive

```
rvec.import(type=html) "http://www.tradingroom.com.au/apps/mkt/
    forex.ac" colhead=3
```

loads into a rowvector RVEC the data with the given URL located on the website site "http://www.tradingroom.com.au". The column header is set to three rows.

Text and Binary Files

The syntax for reading text or binary files is:

rowvector_name.**import(type** = *arg*) *source_description [table_description] [variables_description]*

If a *table_description* is not provided, EViews will attempt to read the file as a free-format text file. The following *table_description* elements may be used when reading a text or binary file:

- "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.

- "byrow", transpose the incoming data. This option allows you to import files where the series are contained in rows (one row per series) rather than columns.

- "lastcol", include implied last column. For lines that end with a delimiter, this option adds an additional column.

  When importing a CSV file, lines which have the delimiter as the last character (for example: 'name,description,date,'), EViews normally determines the line to have 3 columns. With the above option, EViews will determine the line to have 4 columns. Note this is not the same as a line containing 'name,description,date'. In this case, EViews will always determine the line to have 3 columns regardless if the option is set.

A central component of the *table_description* element is the format statement. You may specify the data format using the following table descriptors:

- Fortran Format:

  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 Format:

  rformat = "*[n1]Type[Width][.Precision], [n2]Type[Width][.Precision], ...*)"

  where optional type is "$" for string or "#" for number, and *n1*, *n2*, *n3*, *n4*, *etc*. are the range of columns containing the data.

- C printf/scanf Format:

  cformat = "*fmt*"

  where *fmt* follows standard C language (printf/scanf) format rules.

The optional *variables_description* may be formed using the elements:

- "colhead = *int*", number of table rows to be treated as 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.

Text and Binary File Examples (.txt, .csv, etc.)

```
rvec2.import c:\data.csv skip=5
```

reads "Data.CSV" into a RVEC2, skipping the first 5 rows.

```
rvec2.import(type=text) c:\date.txt delim=comma
```

loads the comma delimited data “Date.TXT” into the RVEC2 rowvector object.

Cross-references

See also `Rowvector::export` (p. 710).

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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User's Guide I* for a discussion of labels.

See also Rowvector::displayname (p. 706).

olepush — Rowvector Procs

Push updates to OLE linked objects in open applications.

Syntax

vector_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

read — Rowvector Procs

Import data from a foreign disk file into a rowvector.

(This is a deprecated method of importing into a rowvector. See Rowvector::import (p. 717) for the currently supported method.)

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

prompt	Force the dialog to appear from within a program.

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 *default* 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.
skipcol = *integer*	Number of columns to skip. Must be used with the "*rect*" option.
skiprow = *integer*	Number of rows to skip. Must be used with the "rect" option.
comment = *symbol*	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.
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

```
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 152 of *User's Guide I* for a discussion and examples of importing data from external files.

See also `Rowvector::export` (p. 710).

resize	Rowvector Procs

Resize the rowvector object.

Syntax

rowvector_name.**resize** *rows*

Examples

```
rvec1.resize 20
```

resizes the rowvector RVEC1 to 20 columns, retaining the contents of any existing elements and initializing new elements to 0.

Cross-references

See also `Rowvector::rowvector` (p. 727).

rowvector	Rowvector Declaration

Declare a rowvector object.

The `rowvector` command declares and optionally initializes a rowvector object.

Syntax

rowvector*(n1) rowvector_name*

rowvector *rowvector_name = assignment*

You may optionally specify the size (number of columns) of the rowvector 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 rowvector 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 rowvector of size one with value 0. COEFVEC is a rowvector of size 20 with all elements equal to 2. NEWCOEF is also a rowvector of size 20 with all elements equal to the same values as COEFVEC.

Cross-references

See also `Coef::coef` (p. 26) and `Vector::vector` (p. 1263).

setattr	Rowvector Procs

Set the object attribute.

Syntax

rowvector_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the "revised" attribute in the object A to the string "never", and extracts the attribute into the string object S.

Cross-references

See "Adding Custom Attributes in the Label View" on page 123 and "Adding Your Own Label Attributes" on page 70 of *User's Guide I*.

setcollabels	Rowvector Procs

Set the column labels in a rowvector object.

Syntax

rowvector_name.**setcollabels** *label1 label2 label3….*

Follow the `setcollabels` command with a space delimited list of column labels. Note that each column label should not contain spaces unless it is enclosed in quotes. If you provide fewer labels than there are columns, EViews will keep the corresponding default column names ("C11", "C12", etc...).

Examples

```
rvec1.setcollabels USA UK FRANCE
```

sets the column label for the first column in rowvector MAT1 to USA, the second to UK, and the third to FRANCE.

Cross-references

See `Rowvector::setrowlabels` (p. 730).

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:

g*[.precision]* significant digits

f*[.precision]*	fixed decimal places
c*[.precision]*	fixed characters
e*[.precision]*	scientific/float
p*[.precision]*	percentage
r*[.precision]*	fraction

To specify a format that groups digits into thousands using a comma separator, place a "t" after the format character. For example, to obtain a fixed number of decimal places with commas used to separate thousands, use "ft*[.precision]*".

To use the period character to separate thousands and commas to denote decimal places, use ".." (two periods) when specifying the precision. For example, 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)"), 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. 731), `Rowvector::setindent` (p. 729) and `Rowvector::setjust` (p. 730) 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 indentation setttings are taken from the Global Defaults for spreadsheet views ("Spreadsheet Data Display" on page 1019 of *User's Guide I*) at the time the spreadsheet was created.

For rowvectors, `setindent` operates on all of the cells in the rowvector.

Examples

To set the indentation for all the cells in a rowvector object:

```
rv1.setindent 2
```

Cross-references

See `Rowvector::setwidth` (p. 731) and `Rowvector::setjust` (p. 730) for details on setting spreadsheet widths and justification.

setjust

Rowvector Procs

Set the horizontal justification for all cells in the spreadsheet view of the rowvector object.

Syntax

rowvector_name.**setjust** *format_arg*

where *format_arg* may be set to left, center, right, or auto (strings are left-justified and numbers are right-justified). Default display settings can be set in General Options; see "Spreadsheet Data Display" on page 1019 of *User's Guide I*.

Examples

```
row1.setjust left
```

left-justifies the cells in the spreadsheet view of the rowvector ROW1.

Cross-references

See `Rowvector::setwidth` (p. 731) and `Rowvector::setindent` (p. 729) for details on setting spreadsheet widths and indentation.

setrowlabels

Rowvector Procs

Set the row label in a rowvector object.

Syntax

vector_name.**setrowlabels** *label*

Follow the `setrowlabels` command with a row label. Note that the row label should not contain spaces unless it is enclosed in quotes.

Examples

```
rv1.setrowlabels USA
```

sets the row label to "USA".

Cross-references

See `Rowvector::setcollabels` (p. 728).

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. 729) and `Rowvector::setjust` (p. 730) 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.

showlabels	Rowvector Procs

Displays the custom row and column labels of a rowvector spreadsheet.

Syntax

rowvector_name.**showlabels** *mode*

where *mode* is either 0 or 1 where 0 displays the default row and column labels and 1 displays the custom row and column labels (if present).

Examples

```
r1.showlabels 1
```

displays the custom row and column labels for the R1 spreadsheet. If custom labels have not been set the default labels will be displayed.

```
r1.showlabels 0
```

displays the default row and column labels for the R1 spreadsheet.

Cross-references

See also `Rowvector::setcollabels` (p. 728) and `Rowvector::setrowlabels` (p. 730).

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 450 and page 667 of *User's Guide I* for a discussion of the descriptive statistics views of series and groups.

teststat	Rowvector Views

Test simple hypotheses of whether the mean, median, or variance of the elements of a rowvector are equal to specific values.

Syntax

rowvector_name.**teststat***(options)*

Specify the type of test and the value under the null hypothesis as an option. You must specify at least one hypothesis.

For tests of means, you may either estimate the variance or specify the variance as an option.

Options

mean = *number*	Test the null hypothesis that the mean equals the specified number.
med = *number*	Test the null hypothesis that the median equals the specified number.
var = *number*	Test the null hypothesis that the variance equals the specified number. The number must be positive.
std = *number*	Test equality of mean conditional on the specified standard deviation. The standard deviation must be positive.
prompt	Force the dialog to appear from within a program.
p	Print the test results.

Examples

```
rvec1.teststat(mean=7)
```

tests the null hypothesis that the mean of RVEC1 is equal to 7, using an estimated standard deviation.

```
rvec1.teststat(mean=7, std=2)
```

tests the null that the mean is 7, using an estimated standard deviation, and also assuming that the standard deviation is known to be 2.

```
rvec1.teststat(var=4)
```

tests the null hypothesis that the variance of RVEC1 is equal to 4.

Cross-references

See "Descriptive Statistics & Tests" on page 450 of *User's Guide I* for a discussion of simple hypothesis tests.

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.

This routine should realistically only be used in the oft-hand chance that you wish to write into a Lotus file. Improved Excel, text, and other format writing is available in `Rowvector::export` (p. 710).

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

prompt	Force the dialog to appear from within a program.

File type

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

If you omit the "*t* = " option, EViews will determine the type based on the file extension. Unrecognized extensions will be treated as ASCII files. For Lotus and Excel spreadsheet files

specified without the "*t* = " option, EViews will automatically append the appropriate extension if it is not otherwise specified.

ASCII text files

na = *string*	Specify text string for NAs. Default is "NA".
d = *arg*	Specify delimiter *(default* is tab*)*: "s" (space), "c" (comma).

Spreadsheet (Lotus, Excel) files

letter_number	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 171 of *User's Guide I* for a discussion.

See also `Rowvector::export` (p. 710) and `Rowvector::read` (p. 724).

stats	Rowvector 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 rowvector.

Syntax

rowvector_name.**stats***(options)*

Options

p	Print the stats table.

Examples

```
rvec1.stats
```

displays the descriptive statistics view of the elements of the rowvector RVEC1.

Cross-references

See "Descriptive Statistics & Tests" on page 450 of *User's Guide I* for a discussion of the descriptive statistics views of series.

Sample

Sample of observations. Description of a set of observations to be used in operations.

Sample Declaration

sample ..... declare a sample object (p. 743).

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

display ..... display table, graph, or spool in object window (p. 740).
label..... label information for the sample (p. 742).
spec ..... display sample specification information (p. 745).

Sample Procs

clearhist ..... clear the contents of the history attribute (p. 739).
clearremarks ..... clear the contents of the remarks attribute (p. 739).
copy..... creates a copy of the sample (p. 740).
displayname..... set display name (p. 741).
olepush ..... push updates to OLE linked objects in open applications (p. 743).
set..... reset the sample range (p. 744).
setattr..... set the value of an object attribute (p. 745).

Sample Data Members

String values

@attr("*arg*") ..... string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@description..... string containing the Sample object's description (if available).
@detailedtype ..... string with the object type: "SAMPLE".
@displayname..... string containing the Sample object's display name. If the Sample has no display name set, the name is returned.
@name ..... string containing the Sample object's name.
@remarks ..... string containing the Sample object's remarks (if available).
@type ..... string with the object type: "SAMPLE".
@updatetime..... string representation of the time and date at which the Sample was last updated.

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 eq1.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.

clearhist	Sample Procs

Clear the contents of the history attribute for sample objects.

Removes the sample's history attribute, as shown in the label view of the sample.

Syntax

sample_name.**clearhist**

Examples

```
s1.clearhist
s1.label
```

The first line removes the history from the sample S1, and the second line displays the label view of S1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Sample::label` (p. 742).

clearremarks	Sample Procs

Clear the contents of the remarks attribute.

Removes the sample's remarks attribute, as shown in the label view of the sample.

Syntax

sample_name.**clearremarks**

Examples

```
s1.clearremarks
s1.label
```

The first line removes the remarks from the sample S1, and the second line displays the label view of S1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Sample::label` (p. 742).

copy	Sample Procs

Creates a copy of the sample.

Creates either a named or unnamed copy of the sample.

Syntax

sample_name.**copy**

sample_name.**copy** *dest_name*

Examples

```
s1.copy
```

creates an unnamed copy of the sample S1.

```
s1.copy s2
```

creates S2, a copy of the sample S1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

display	Sample Views

Display table, graph, or spool output in the sample object window.

Display the contents of a table, graph, or spool in the window of the sample object.

Syntax

sample_name.**display** *object_name*

Examples

```
sample1.display tab1
```

Display the contents of the table TAB1 in the window of the object SAMPLE1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231.

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 123 of *User's Guide I* for a discussion of labels and display names.

See also `Sample::label` (p. 742).

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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User's Guide I* for a discussion of labels.

See also `Sample::displayname` (p. 741).

olepush	Sample Procs

Push updates to OLE linked objects in open applications.

Syntax

sample_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

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. 744) 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.

```
sample s2 @all if @hourf<=9.5 and @hourf<=14.5
```

declares a sample S2 that includes all observations that are between 9:30AM and 2:30PM.

Cross-references

See "Samples" on page 142 of *User's Guide I* and "Dates" on page 104 of the *Command and Programming Reference* for a discussion of using samples and dates in EViews.

See also `Sample::set` (p. 744) and `smpl` (p. 592) of the *Command and Programming Reference*.

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***(options) sample_description*

Follow the `set` command with a sample description. See `sample` for instructions on describing a sample.

Options

prompt	Force the dialog to appear from within a program.

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 142 of *User's Guide I* for a discussion of samples in EViews.

See also `Sample::sample` (p. 743), `Sample::spec` (p. 745), and `smpl` (p. 592) of the *Command and Programming Reference*.

setattr	Sample Procs

Set the object attribute.

Syntax

sample_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the “revised” attribute in the object A to the string “never”, and extracts the attribute into the string object S.

Cross-references

See “Adding Custom Attributes in the Label View” on page 123 and “Adding Your Own Label Attributes” on page 70 of *User’s Guide I*.

spec	Sample Procs

Display the sample in a sample object.

The `spec` procedure displays the sample in an existing sample object.

Syntax

sample_name.**spec***(options)*

Options

p	Print the sample specification

Examples

```
sample s1 @first 1973
s1.spec
```

The first line declares and defines a sample object named S1 from the beginning of the workfile range to 1973. The second line displays the sample specification.

Cross-references

See "Samples" on page 142 of *User's Guide I* for a discussion of samples in EViews.

See also `Sample::sample` (p. 743), `Sample::set` (p. 744), and `smpl` (p. 592) of the *Command and Programming Reference*.

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. 751).

To declare a scalar object, use the keyword `scalar`, followed by a name, an " = " sign and a scalar expression or value.

Scalar Views

display.....display table, graph, or spool in object window (p. 749).
label.....label view (p. 750).
sheet.....spreadsheet view of the scalar (p. 752).

Scalar Procs

clearhist.....clear the contents of the history attribute (p. 748).
clearremarks.....clear the contents of the remarks attribute (p. 748).
copy.....creates a copy of the scalar (p. 749).
displayname.....set display name (p. 750).
olepush.....push updates to OLE linked objects in open applications (p. 751).
setattr.....set the value of an object attribute (p. 752).

Scalar Data Members

String values

@attr("*arg*").....string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@description.....string containing the Scalar object's description (if available).
@detailedtype.....string with the object type: "SCALAR".
@displayname.....string containing the Scalar object's display name. If the Scalar has no display name set, the name is returned.
@name.....string containing the Scalar object's name.
@remarks.....string containing the Scalar object's remarks (if available).
@type.....string with the object type: "SCALAR".
@updatetime.....string representation of the time and date at which the Scalar was last updated.

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.

clearhist	Scalar Procs

Clear the contents of the history attribute.

Removes the scalar's history attribute, as shown in the label view of the scalar.

Syntax

scalar_name.**clearhist**

Examples

```
s1.clearhist
s1.label
```

The first line removes the history from the scalar S1, and the second line displays the label view of S1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Scalar::label` (p. 750).

clearremarks	Scalar Procs

Clear the contents of the remarks attribute.

Removes the scalar's remarks attribute, as shown in the label view of the scalar.

Syntax

scalar_name.**clearremarks**

Examples

```
s1.clearremarks
s1.label
```

The first line removes the remarks from the scalar S1, and the second line displays the label view of S1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Scalar::label` (p. 750).

copy	Scalar Procs

Creates a copy of the scalar.

Creates either a named or unnamed copy of the scalar.

Syntax

scalar_name.**copy**

scalar_name.**copy** *dest_name*

Examples

```
s1.copy
```

creates an unnamed copy of the scalar S1.

```
s1.copy s2
```

creates S2, a copy of the scalar S1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

display	Scalar Views

Display table, graph, or spool output in the scalar object window.

Display the contents of a table, graph, or spool in the window of the scalar object.

Syntax

scalar_name.**display** *object_name*

Examples

```
scalar1.display tab1
```

Display the contents of the table TAB1 in the window of the object SCALAR1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

displayname	Scalar Procs

Display name for a scalar object.

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

Syntax

scalar_name.**displayname** *display_name*

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

Examples

```
sc.displayname Hours Worked
sc.label
```

The first line attaches a display name "Hours Worked" to the scalar object SC, and the second line displays the label view of SC, including its display name.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names. See also `Scalar::label` (p. 750).

label	Scalar Views

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

Syntax

scalar_name.**label**

scalar_name.**label***(options) text*

Options

To modify the label, you should specify one of the following options along with optional text. If there is no text provided, the specified field will be cleared:

c	Clears all text fields in the label.
d	Sets the description field to *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of the scalar S1 with "Mean of Dependent Variable from EQ3":

```
s1.label(r)
s1.label(r) Mean of Dependent Variable EQ3
```

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels.

olepush — Scalar Procs

Push updates to OLE linked objects in open applications.

Syntax

scalar_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

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.

setattr	Scalar Procs

Set the object attribute.

Syntax

scalar_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the “revised” attribute in the object A to the string “never”, and extracts the attribute into the string object S.

Cross-references

See “Adding Custom Attributes in the Label View” on page 123 and “Adding Your Own Label Attributes” on page 70 of *User’s Guide I*.

sheet	Scalar Views

Spreadsheet view of a scalar object.

Syntax

scalar_name.**sheet***(options)*

Options

p	Print the spreadsheet view.

Examples

```
s01.sheet
```

displays the spreadsheet view of S01.

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. 796).
genr .....create numeric series object (p. 798).
series .....declare numeric series object (p. 832).

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.

Note: to convert data between series and vectors, see "Copying Data Between Matrices and Series/Groups" on page 290, `stom` (p. 598), `stomna` (p. 599), `mtos` (p. 522), all in the *Command and Programming Reference*.

Series Views

bdstest .....BDS independence test (p. 765).
bubbletest .....perform tests on the existence of a bubble in the series (p. 769).
buroot.....carries out unit root tests which allow for a single breakpoint (p. 770).
changepoints.....perform tests on a change in the location parameter (mean) of the series (p. 773).
correl.....correlogram, autocorrelation and partial autocorrelation functions (p. 778).
display.....display table, graph, or spool in object window (p. 779).
dups .....duplicates display for observations in the series (p. 785).
edftest.....empirical distribution function tests (p. 786).
forceval.....evaluate different forecasts of a series, and perform the forecast combination test (p. 793).
freq.....one-way tabulation (p. 795).
hist .....descriptive statistics and histogram (p. 798).
label .....label information for the series (p. 804).
lrvar .....compute the symmetric, one-sided, or strict one-sided long-run variance of a series (p. 805).
outliers .....detect outlying observations (p. 818).

pancov ..... compute covariances, correlations, and other measures of association for a panel series (p. 819).
panpcomp ..... perform principal components analysis on a panel series (p. 822).
seasuroot ..... seasonal unit root test on an ordinary series (p. 829).
sheet ..... spreadsheet view of the series (p. 849).
statby ..... statistics by classification (p. 853).
stats ..... descriptive statistics table (p. 855).
testby ..... equality test by classification (p. 857).
teststat ..... simple hypothesis tests (p. 858).
trendtests ..... perform tests on the existence of a trend in the series (p. 862).
uroot ..... unit root test on an ordinary or panel series (p. 863).
uroot2 ..... compute dependent (second generation) panel unit root tests on a series in a panel workfile (p. 868).
vratio ..... compute Lo and MacKinlay variance ratio test, or Wright rank, rank-score, or sign-based forms of the test (p. 874).
waveanova ..... compute the wavelet variance decomposition of the series (p. 876).
wavedecomp ..... compute the wavelet transform of the series (p. 878).
waveoutlier ..... perform wavelet outlier detection for the series (p. 880).
wavethresh ..... perform wavelet thresholding (denoising) of the series (p. 883).

Series Procs

adjust ..... modify or fill in the values in a series (p. 760).
autoarma ..... forecast from a series using an ARIMA model with automatic determination of the specification (p. 763).
bpf ..... compute and display band-pass filter (p. 766).
classify ..... recode series into classes defined by a grid, specified limits, or quantiles (p. 774).
clearcontents ..... clear a contiguous block of observations in a series (p. 776).
clearhist ..... clear the contents of the history attribute (p. 777).
clearremarks ..... clear the contents of the remarks attribute (p. 777).
copy ..... creates a copy of the series (p. 778).
displayname ..... set display name (p. 780).
distdata ..... save distribution plot data to a matrix (p. 780).
dsa ..... seasonally adjust daily data using the DSA method (p. 782).
ets ..... perform Error-Trend-Season (ETS) estimation and exponential smoothing (p. 787).
fill ..... fill the elements of the series (p. 791).
forcavg ..... average forecasts of a series (p. 792).
hpf ..... Hodrick-Prescott filter (p. 799).

Series Graph Views

Graph creation views are discussed in detail in "Graph Creation Command Summary" on page 1267.

area..... area graph of the series (p. 1269).
bar..... bar graph of the series (p. 1275).
boxplot..... boxplot graph (p. 1279).
distplot..... distribution graph (p. 1283).
dot..... dot plot graph (p. 1290).
line..... line graph of the series (p. 1298).
qqplot..... quantile-quantile plot (p. 1306).
seasplot..... seasonal line graph (p. 1322).
spike..... spike graph (p. 1323).

Series Data Members

String values

@attr("*arg*") ..... string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@description..... string containing the Series object's description (if available).
@depends..... string containing a list of the series in the current workfile on which this series depends.
@detailedtype..... string with the object type: "SERIES", if an ordinary series, or "LINK", if defined by link.
@displayname..... string containing the Series object's display name. If the Series has no display name set, the name is returned.
@first..... string containing the date or observation number of the first non-NA observation of the series. In a panel workfile, the first date at which any cross-section has a non-NA observation is returned.
@firstall..... returns the same as `@first`, however in a panel workfile, the first date at which all cross-sections have a non-NA observation is returned.
@hilo..... string containing the series object's high-to-low frequency conversion method.
@last..... string containing the date or observation number of the last non-NA observation of the series. In a panel workfile, the last date at which any cross-section has a non-NA observation is returned.
@lastall..... returns the same as `@last`, however in a panel workfile, the last date at which all cross-sections have a non-NA observation is returned.
@lohi..... string containing the series object's low-to-high frequency conversion method.

@name .....string containing the Series object's name.
@remarks .....string containing the Series object's remarks (if available).
@type .....string with the object type: "SERIES".
@updatetime .....string represent of the time and date at which the Series was last updated.

Scalar values

@obs .....scalar containing the number of non-NA observations.
(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.

adjust	Series Procs

Modify or fill in the values in a series.

Syntax

series_name.**adjust** *[transform] [operator] [values] [interpolation]*

Follow the `adjust` keyword with an expression made up of a combination of *transform*, *operator*, *values* and *interpolation* components. *transform* is used to specify a transformation of the data to which the adjustment will be made. The *operator* contains a mathematical expression defining how you would like to adjust the values in the series. *values* contains the values used during that operation. Finally, the *interpolation* component specifies how any missing values in the *values* component should be filled in via interpolation.

All adjustments are made on the current workfile sample.

Transform

The following *transformations* are available. If a transformation is specified, any adjustments specified in the *operator* or *interpolation* components is made to the transformed data rather than the raw data.

Transform	Description
d	One period difference.
dy	Annual difference.
pch	One period percentage change.
pcha	Annualized one period percentage change.
pchy	Annual percentage change.
log	Natural logarithm.
dlog	One period difference of logged values.

Operators

The following *operators* are available:

Operator	Description
=	Overwrites the existing value with the new value.
+ =	Adds the new value to the existing value.
- =	Subtracts the new value from the existing value.
* =	Multiplies the existing value by the new value.

/=	Divides the existing value by the new value.
=_	Overwrites the existing value with the previous cell's value.
+_	Add the new value to the previous observation's value.
-_	Subtract the new value from the previous observation's value.
*_	Multiply the previous observation's value by the new value.
/_	Divide the previous observation's value by the new value.
\	Reverse the order of the observations. Note this operator cannot be used with a values or interpolation component.

Values

The *values* component should be made up of a space delimited set of values to use during the adjustment. In addition to single numbers, you may use the following keywords as part of the *values* component:

Keyword	Description
.	A single value to be filled in by interpolation.
#	Use the existing series value, unless it is an NA, in which case fill it by interpolation.
NA	Insert an NA (which will not be filled by interpolation).
R*int1*[(*int2*)]	Repeats the previous value *int1* times. You may optionally include a second number in parenthesis indicating how many of the previous values to repeat.
..	Interpolate between all remaining values.

Interpolation

The *interpolation* component specifies how to fill in any missing values in the *values* component designated for interpolation. By default a cubic spline is used for interpolation. The other available choices are show below.

Method Symbol	Description
__ (double underscore)	Repeats previous non-missing value.
^	Linear interpolation.
~	Cubic spline interpolation
&	Catmull-Rom spline interpolation.

^*	Log-linear (multiplicative) interpolation (linear in the log of the data).
~*	Multiplicative cubic spline interpolation (a cubic spline on the log of the data).
&*	Multiplicative Catmull-Rom spline interpolation (a Catmull-Rom spline on the log of the data).

Examples

The following command replaces the first four observations in the current sample of the series UNEMP with the values 2.4, 3.5, 2.9 and 1.4.

```
unemp.adjust = 2.4 3.5 2.9 1.4
```

This command modifies the first ten observations in UNEMP, by replacing them with the values: 3.4, 3.15, 2.9, 3.2, 3.5, 3.7, 3.5, 3.7, 3.5, 3.7. Note that the second observation (3.15) has been interpolated, using linear interpolation, between 3.4 and 2.9. Similarly the 4th observation was interpolated between 2.9 and 3.5. Also note that the values 3.5 and 3.7 were repeated three times.

```
unemp.adjust = 3.4 . 2.9 . 3.5 3.7 R3(2) ^
```

The following command replaces the log of the first observation in the current sample with 3.4 (setting the raw value equal to exp(3.4) = 29.96). The second observation is left alone (unless it contains an NA, in which case the log value is interpolated). The third observation's logged value is replaced with 2.2. The log of the penultimate observation in the current sample is replaced with 3.9, and the last observation with 4.8. All observations between the third and the penultimate are interpolated using a cubic spline interpolation method.

```
unemp.adjust log = 3.4 # 2.2 .. 3.9 4.8
```

This command adjusts all the observations in the current sample by adding to the existing values. The first observation has 3.4 added to it. The second has 2.9 added to it, and the third has 4.5 added. The last observation has 1.9 added to it. The values added to the observations in between are calculated via a multiplicative Catmull-Rom spline interpolation.

```
unemp.adjust += 3.4 2.9 4.5 .. 1.9 &*
```

Cross-references

See Appendix B. "Enhanced Spreadsheet Editing," on page 1037 and "Series Adjust" on page 495 in *User's Guide I* for additional discussion of series adjustment.

autoarma	Series Procs

Forecast from a series using an ARIMA model with the specification of the model selected automatically.

Syntax

series.**autoarma**(*options*) *forecast_name* [*exogenous_regressors*]

Options

tform = *arg*	Specify the type of dependent variable transformation. *arg* may be "auto" (automatically decide between log or no transformation, default), "none" (perform no transformation), "log" (perform a log transformation), and "bc" (perform the Box-Cox transformation.
bc = *int*	Set the power of the Box-Cox transformation. Only applicable if the *tform = bc* option is used.
diff = *int*	Set the maximum level of differencing to test for. Default value is 2.
maxar = *int*	Set the maximum number of AR terms. Default value is 4.
maxma = *int*	Set the maximum number of MA terms. Default value is 4.
maxsar = *int*	Set the maximum number of seasonal AR terms. Default value is 0.
maxsma = *int*	Set the maximum number of seasonal MA terms. Default value is 0.
periods = *int*	Set the periodicity of the seasonal ARMA terms. This defaults to the number of observations in a year, based on current workfile frequency.
avg = *key*	Use forecast averaging, rather than model selection. *key* sets the type of averaging to perform, and may take values of "aic" (SAIC weights), "sic" (BMA weights) or "uni" (uniform weights).
select = *key*	Set the model selection criteria. *key* make take values of "aic" (Akaike Information Criteria, default), "sic" (Schwarz Information Criteria), "hq" (Hannan-Quinn criteria) or "mse" (Mean Square Error criteria). This option is ignored if the "avg = " option is used.
nonconv	Allow non-converged models to be used in model selection or forecast averaging.

mselen = *key*	Set the percentage of the estimation sample to be used for MSE calculation. *key* may take values of "5", "10", "15" or "20". This option is only applicable if the "select = mse" option is used.
msetype = *key*	Set the type of forecast to use when calculating MSE. *key* may either be "dyn" (dynamic, default), or an integer, *n*, between 1 and 12 indicating that an *n*-step static forecast should be performed. This option is only applicable if the "select = mse" option is used.
kpsssig = *key*	Set the significance level of the KPSS test when determining the appropriate level of differencing for the dependent variable. *key* may take values of "1", "5" (default) or "10".
fgraph	Output a forecast comparison graph.
atable	Output a selection criteria comparison table
agraph	Output a selection criteria comparison graph.
etable	Output a final equation output table. Not applicable if the "avg = " option is used.
eqname = *name*	Create an equation object in the workfile with the same specification as the final selected equation. Not applicable if the "avg = " option is used.
seed = *num*	Set the random number generator seed for random starting values.
prompt	Force the dialog to appear from within a program.
p	Print results.

Forecast sample options

The forecast sample will start at the observation immediately after the estimation sample (the current workfile sample). The forecast endpoint is given by either:

forclen = *int*	Number of periods to forecast.
forc = "*date*"	Specify the date of the forecast end point.

If omitted, the end point will be the end of the workfile range.

Example

The commands

```
wfopen elecdmd.wf1
elecdmd.autoarma(maxsar=1, maxsma=1, noconv, forclen=20, agraph,
    atable, fgraph) elecdmd_f @expand(@month) realgdp tempf
```

open the workfile "elecdmd.WF1" and then perform automatic forecasting on the series ELECDMD. The forecasts will be stored in a series called ELECDMD_F. The ARIMAX model includes exogenous regressors of REALGDP, TEMPF and a set of monthly dummy variables, created with the `@expand` keyword.

The number of maximum SAR terms and SMA terms are set to 1 (instead of the default 0). Model selection is used to determine the best ARMA model, with non-converged models included in the selection process.

The forecast covers 20 periods, and upon completion, EViews will display a graph of the Akaike information criteria of each of the ARMA models considered, as well as a table of each of the selection criteria, and a graph of the each of the forecasts.

Cross-references

See "Automatic ARIMA Forecasting" on page 580 of *User's Guide I* for additional discussion.

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 = *arg* *(default* = "p"*)*	Method for calculating ϵ : "p" (fraction of pairs), "v" (fixed value), "s" (standard deviations), "r" (fraction of range).
e = *number*	Value for calculating ϵ .
d = *integer*	Maximum dimension.
b = *integer*	Number of repetitions for bootstrap p-values. If option is omitted, no bootstrapping is performed.
o = *arg*	Name of output vector for final BDS z-statistics.
prompt	Force the dialog to appear from within a program.
p	Print output.

Cross-references

See "BDS Independence Test" on page 855 of *User's Guide II* 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. 1279) 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.

To display the graph, you may need to precede the object command with the "show" keyword.

Options

type = *arg* *(default* = "bk"*)*	Specify the type of band-pass filter: "bk" is the Baxter-King fixed length symmetric filter, "cffix" is the Christiano-Fitzgerald fixed length symmetric filter, "cfasym" is the Christiano-Fitzgerald full sample asymmetric filter.

low = *number*, high = *number*	Low (P_L) and high (P_H) values for the cycle range to be passed through (specified in periods of the workfile frequency). 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". The arguments must satisfy $2 \leq P_L < P_H$. The corresponding frequency range to be passed through will be $(2\pi / P_H, 2\pi / P_L)$.
lag = *integer*	Fixed lag length (positive integer). Sets the fixed lead/lag length for fixed length filters ("type = bk" or "type = cffix"). Must be less than half the sample size. Defaults to the workfile equivalent of 3 years for semi-annual to daily workfiles; otherwise sets "lag = 3".
iorder = *[0,1]* *(default = 0)*	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. The integration order is only used in the computation of Christiano-Fitzgerald filter weights ("type = cffix" or "type = cfasym"). When "iorder = 1", the filter weights are constrained to sum to zero.
detrend = *arg* *(default = "n")*	Detrending method for Christiano-Fitzgerald filters ("type = cffix" or "type = cfasym"). You may select the default argument "n" for no detrending, "c" to demean, or "t" to remove a constant and linear trend. You may use the argument "d" to remove drift, if the option "iorder = 1" is also specified.
nogain	Suppresses plotting of the frequency response (gain) function for fixed length symmetric filters ("type = bk" or "type = cffix"). By default, EViews will plot the gain function.
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 $t = q + 1, \ldots, n - q$.

For time-varying filters, the weight matrix is of dimension $n \times n$ where n is the number of non-missing observations in the current sample. Row r of the matrix contains the weighting vector used to generate the r-th observation of the filtered series, where column c contains the weight on the c-th observation of the original series. The filtered series may be computed as:

$$z_t = \sum_{c=1}^{T} w(r, c) y_c \qquad r = 1, \ldots, T$$

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.

prompt

Force the dialog to appear from within a program.

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 628 of *User's Guide I*.

See also `Series::hpf` (p. 799).

bubbletest	Series Views

Perform tests on the existence of a bubble in the series.

Calculate either the rolling ADF (RADF), supremum ADF (SADF) or generalized supremum ADF (GSADF) test for the existence of a bubble in an asset.

Syntax

series_name.**bubbletest***(options)*

Options

test = *arg*	Specify the type of test to perform. Use "SADF" to perform the supremum ADF test, and "GSADF" to perform the generalized supremum ADF test. By default, the rolling ADF test is performed.
window = *int*	Set the length of the rolling window (for RADF test), or the initial window length (for SADF and GSADF tests).
exog = *arg*	Specification of exogenous trend variables in the test equation: "trend" (include a constant and a linear time trend), "none" (do not include any exogenous regressors). By default, a constant is included.
lagmethod = *arg*	Method for selecting lag length (number of first difference terms) to be included in the Dickey-Fuller test regression or number of lags in the AR spectral density estimator: "aic" (Akaike), "sic" (Schwarz), or "hqc" (Hannan-Quinn).
lag = *int*	User-specified fixed lag.

maxlag = *int*	Maximum lag length to consider when performing automatic lag length selection.
reps = *arg*	Number of bootstrap replications.
seed = *int*	Set the random number generator seed.
rng = *arg*	Set random number generator type. Available types are: 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").
nocv	Do not display critical value line in graph.
cvsig = *arg* *(default* = 0.95*)*	Critical value for display in graph and for critical value output series (using "cvout = ").
out = *name*	Name of a series to hold the test statistics.
cvout = *name*	Name of a series to hold the test critical values.
prompt	Force the dialog to appear from within a program.
p	Print results.

Examples

```
spy.bubbletest(test=gsadf, window=30, lagmethod=hqc, seed=40)
```

Performs a generalized supremum bubble test on the SPY series, with an initial window length of 30 observations, using the Hannan-Quinn Criterion to determine the appropriate lag length, and using a seed value of 40 for the random number generator used in the bootstrapped p-values.

Cross-references

See "Explosive Bubble Tests" on page 767 of *User's Guide II*.

buroot	Series Views

Carries out unit root tests which allow for a single breakpoint.

Syntax

series_name.**buroot***(options)*

Basic Specification Options

You should specify the exogenous variables and order of dependent variable differencing in the test equation using the following options:

exog = *arg* (*default* = "const")	Specification of exogenous trend variables in the test equation: "const" "trend" (include a constant and a linear time trend).
dif = *integer* (*default* = 0)	Order of differencing of the series prior to running the test. Valid values are {0, 1, 2}.

Break Options

break = *arg* (*default* = "const")	Specification of breaking trend variables in the test equation: "const" (intercept only), "both" (intercept and trend), "trend" (trend only). The latter two are applicable only if "exog = trend").
breakmethod = *arg* (*default* = "dfuller")	Method of specifying the break date: "dfuller" (minimize Dickey-Fuller *t*-statistic), "minincpt" (minimize intercept break *t*-statistic), "maxincpt" (maximize intercept break *t*-statistic), "absincpt" (maximize intercept break absolute *t*-statistic), "mintrend" (minimize trend break *t*-statistic), "maxtrend" (maximize trend break *t*-statistic), "abstrend" (maximize trend break absolute *t*-statistic), "both" (maximize joint intercept and trend break F-statistic), "user" (fixed break date specified using the "userbreak = " option).
trim = *arg* (*default* = 10)	Trimming percentage for allowable break dates to consider in automatic break selection (applicable if the specified break method selects a date on the basis of intercept or trend break coefficients).
userbreak = *dateobs*	User-specified break date.
type = *arg* (*default* = "io")	Break type: innovation outlier ("io"), additive outlier ("ao").

Lag Difference Options

Specifies the number of lag difference terms to be included in the test equation. The default is to perform automatic selection using the Schwarz information criterion. You may specify a fixed lag using the "lag = " option.

lagmethod = *arg* (*default* = "sic")	Method for selecting lag length (number of first difference terms) to be included in the Dickey-Fuller test regressions: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn), "msaic" (Modified Akaike), "msic" (Modified Schwarz), "mhqc" (Modified Hannan-Quinn), "tstat" (Ng-Perron first backward significant *t*-statistic), "fstat" (significant *F*-statistic).
lag = *integer*	Use-specified fixed lag.
maxlag = *integer*	Maximum lag length to consider when performing automatic lag length selection. $$default = \text{int}((12\,T/100)^{0.25})$$
lagpval = *arg* (*default* = 0.1)	Probability value for test-based automatic lag selection (when "lagmethod = tstat" and "lagmethod = fstat).

General options

nograph	Do not display breakpoint selection graph (by *default*, EViews shows a graph of all of the individual unit root tests and AR coefficients when there is endogenous breakpoint selection).
output = *arg*	Output matrix containing individual unit root regression results for all candidate break dates. Each row contains the relevant workfile observation ID (as reported by @TREND), AR coefficient, AR coefficient standard error, number of observations, number of coefficients, number of lags, and if applicable, the *t*-statistic or *F*-statistic used in break selection.
prompt	Force the dialog to appear from within a program.
p	Print output from the test.

Cross-references

See "Unit Root Tests with a Breakpoint" on page 786 of *User's Guide II* for additional discussion.

See also `Series::uroot` (p. 863).

cdfplot	Series Views

Empirical distribution functions.

The `cdfplot` command is no longer supported. See `distplot` (p. 1283).

cdtest	Series Views

Test for the presence of cross-sectional dependence in a panel series.

Computes the Breusch-Pagan (1980) LM, Pesaran (2004) scaled LM, Pesaran (2004) CD, and Baltagi, and Feng and Kao (2012) bias-corrected scaled LM test for a panel series.

Syntax

series_name.**cdtest**

Options

p	Print test results

Examples

```
ser1.cdtest
```

will compute and display the panel cross-section dependence test results.

Cross-references

See "Panel Cross-section Dependence Test" on page 1365 of *User's Guide II* for discussion.

changepoints	Series Views

Perform tests on a change in the location parameter (mean) of the series.

Calculate the standard normal, Quandt-Andrews, Pettitt and Buishand tests for a change in the location of the distribution of a series.

Syntax

series_name.**changepoints***(options)*

Options

noboot	Do not calculate bootstrapped p-values.
simple	Use the simple bootstrap. By default, the sieve bootstrap is used.
mle	Calculate the bootstrap AR estimates using MLE. Default is to use the HVK estimates.
reps = *arg*	Number of bootstrap replications.
seed = *int*	Set the random number generator seed.

rng = *arg*	Set random number generator type. Available types are: 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").
out = *name*	Specify the name of the matrix containing the test statistics and p-values.
prompt	Force the dialog to appear from within a program.
p	Print results.

Examples

```
gdpc1.changepoints(iter=19999, out=cpstats)
```

Performs a change point test on the GDPC1 series, performing 19,999 bootstraps using HVK estimates in a sieve bootstrap, and saving the test statistics and p-values into a matrix object named CPSTATS.

Cross-references

See "Change Point Tests" on page 755 of *User's Guide II* for discussion.

classify — Series Procs

Recode the series into classes defined by a grid, specified limits, or quantiles.

Syntax

series_name.**classify**(*options) spec @ outname [mapname]*

You should the `classify` keyword with any desired options, a specification *spec*, the "@"-sign, the name to be given the output series, and optionally the name for a new valmap object describing the classification.

The form for the *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**min* or 1.05**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* in 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 vector objects containing limit values. The first limit value defines the upper limit of the first interval, and the last limit value defines the lower limit of the last interval. Note that there must be at least one limit value 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

method = *arg* *(default* = "step"*)*	Method for determining classification values: "step"– create a grid from *start* through *end* using the *stepsize*; "bins" – create bins by dividing the region from *start* to *end* into a specified number of bins; "quants" – create bins using the quantile values; "limits" - create bins using the specified limit points.
rightclosed	Bins formed using right-closed intervals. x is defined to be in the bin from a to b if $a < x \leq b$. The default is to use intervals closed on the left.
rangeerr	Generate error if data value is found outside of defined bins. The default is to classify out-of-range values as NAs.
q = *arg* *(default* = "r"*)*	Quantile calculation method. "b" (Blom), "r" (Rankit-Cleveland), "o" (Ordinary), "t" (Tukey), "v" (van der Waerden), "g" (Gumbel). Only relevant where "method = quants".

encode = *arg* *(default* = "index")	Encoding method for output series: "index" – encode as integers from 0 to k where k 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).
prompt	Force the dialog to appear from within a program.
p	Print the results.

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 contain values taken from the right endpoints of the classification intervals. Observations with out-of-range values will be assigned an NA.

```
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 bin index identifiers in 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. Observations with NA values for API5B will be encoded as 0 in the output series.

Cross-references

See "Generate by Classification" on page 497 of *User's Guide I* for additional discussion.

clearcontents	Series Procs

Clear (i.e., replace with NAs) a contiguous block of observations in a series.

Removes the series's history attribute, as shown in the label view of the series.

Syntax

series_name.**clearcontents**(*start_point*) *n*

where *start_point* specifies the first of *n* observations to clear. If *n* is negative, *start_point* specifies the last of |*n*| observations to clear. For dated workfiles, *start_point* should be entered as a date. For panels and undated workfiles, *start_point* should be an observation number.

Examples

```
ser.clearcontents(1952Q2) 10
```

clears 10 observations starting at 1952 quarter 2.

```
ser.clearcontents(10) -5
```

clears 5 observations ending at observation number 10.

clearhist	Series Procs

Clear the contents of the history attribute for series objects.

Removes the series's history attribute, as shown in the label view of the series.

Syntax

series_name.**clearhist**

Examples

```
s1.clearhist
s1.label
```

The first line removes the history from the series S1, and the second line displays the label view of S1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names.

See also `Series::label` (p. 804).

clearremarks	Series Procs

Clear the contents of the remarks attribute.

Removes the series's remarks attribute, as shown in the label view of the series.

Syntax

series_name.**clearremarks**

Examples

```
s1.clearremarks
s1.label
```

The first line removes the remarks from the series S1, and the second line displays the label view of S1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names.

See also `Series::label` (p. 804).

copy	Series Procs

Creates a copy of the series.

Creates either a named or unnamed copy of the series.

Syntax

series_name.**copy**

series_name.**copy** *dest_name*

Examples

```
s1.copy
```

creates an unnamed copy of the series S1.

```
s1.copy s2
```

creates S2, a copy of the series S1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

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

d = *integer (default = 0)*	Compute correlogram for specified difference of the data.
prompt	Force the dialog to appear from within a program.
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 489 and "Partial Autocorrelations (PAC)" on page 490 of *User's Guide I* for a discussion of autocorrelation and partial correlation functions, respectively.

display — Series Views

Display table, graph, or spool output in the series object window.

Display the contents of a table, graph, or spool in the window of the series object.

Syntax

series_name.**display** *object_name*

Examples

```
series1.display tab1
```

Display the contents of the table TAB1 in the window of the object SERIES1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

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 123 of *User's Guide I* for a discussion of labels and display names.

See also `Series::label` (p. 804) and `Series::label` (p. 804).

distdata	Series Procs

Save a matrix containing distribution plot data computed from the series.

Saves the data used to display a histogram, kernel density, theoretical distribution, empirical CDF or survivor plot, or quantile plot to the workfile.

Syntax

series_name.**distdata**(**dtype** = *dist_type, dist_options*) *matrix_name*

saves the distribution plot data specified by *dist_type*, where *dist_type* must be one of the following keywords:

hist	Histogram *(default)*.
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.
theoryqq	Theoretical quantile-quantile plot.

Options

The theoretical quantile-quantile plot type "theoryqq" takes the options described in `qqplot` (p. 1306) under "Theoretical Options" on page 1308.

For the remaining types, *dist_options* are any of the distribution type-specific options described in `distplot` (p. 1283).

Note that the graph display specific options such as "fill," "nofill," and "leg," and "noline" are not relevant for this procedure.

You may use the "prompt" option to force the dialog display

prompt	Force the dialog to appear from within a program.

Examples

```
gdp.distdata(dtype=hist, anchor=0, scale=dens, rightclosed)
    matrix01
```

creates the data used to draw a histogram from the series GDP with the anchor at 0, density scaling, and right-closed intervals, and stores that data in a matrix called MATRIX01 in the workfile.

```
unemp.distdata(dtype=kernel, k=b,ngrid=50,b=.5) matrix02
```

generates the kernel density data computed with a biweight kernel at 50 grid points, using a bandwidth of 0.5 and linear binning, and stores that data in MATRIX02.

```
wage.distdata(dtype=theoryqq, q=o, dist=logit, p1=.5) matrix03
```

creates theoretical quantile-quantile data from the series WAGE using the ordinary quantile method to calculate quantiles. The theoretical distribution is the logit distribution, with the location parameter set to 0.5. The data is saved into the matrix MATRIX03.

Cross-references

For a description of distribution graphs and quantile-quantile graphs, see "Analytical Graph Types," on page 836 of *User's Guide I*.

See also `distplot` (p. 1283) and `qqplot` (p. 1306).

dsa	Series Procs

Seasonally adjust daily series using the DSA method.

Syntax

series_name.**dsa***(options) seas_name [@***fa** *factor_name] [@***trnd** *trend_name]*

You may follow the `dsa` keyword with a name to save the seasonally adjusted series. Further, you may use the `@fa` and `@trnd` keywords to provide names for the saved seasonal factors and the trend series.

Options

forc = *arg*	Specify the end date of the forecast. If not specified, the last observation in the workfile is used. The forecast begins at the observation following the current workfile sample (note, if the workfile sample is equal to the workfile range, no forecasting is performed).
extendfri	For 5-day week data, interpolate to 7-day weeks by repeating the Friday value for Saturday and Sunday. Default is to perform 5-day DSA instead of converting to 7-day.
interwkend	For 5-day week data, interpolate to 7-day weeks by using linear interpolation between the Friday value and Monday values for Saturday and Sunday. Default is to perform 5-day DSA instead of converting to 7-day.
fixedarima	Use a fixed ARIMA model. Default is to use model selection to determine the ARIMA model.
nodiff	Set the level of differencing in the ARIMA model to 0. Default is 1 if using a fixed ARIMA model, or a choice between 0 and 1 if using automatic selection.
maxar = *integer*	If using fixed ARIMA model (see the `fixedarima` option), specify the AR order. If using automatic selection, specify the maximum AR order.

maxma = *integer*	If using fixed ARIMA model (see the `fixedarima` option), specify the MA order. If using automatic selection, specify the maximum MA order.
fixedtrig	Use a fixed number of trigonometric terms to model the seasonal patters in the ARIMA model. Default is to use model selection to determine the number of terms.
maxtrig = *integer*	If using fixed number of trigonometric terms (see the `fixedtrig` option), specify the number of terms. If using automatic selection, specify the maximum number of terms.
olnoao	Do not perform detection of AO outliers. Default is to detect AO outliers.
olnoio	Do not perform detection of IO outliers. Default is to detect AO outliers.
olls	Include detection of LS outliers. Default is to not detect LS outliers.
oltc	Include detection of TC outliers. Default is to not detect TC outliers.
olcvalue = *arg*	Specify the critical value for the outlier detection process.
oldelta = *arg*	Specify the delta value for the TC outlier detection process.
olinits = *integer*	Specify number of inner iterations in the outlier detection process.
oloutits = *integer*	Specify number of outer iterations in the outlier detection process.
extenddow	When forecasting day-of-week factors, repeat the last week of actual data throughout the forecast period. Default is to use exponential smoothing to forecast the factors.
prompt	Force the dialog to appear from within a program.
p	Print view.

STL options

Day-of-week

weeksp = *integer*	Specify the seasonal polynomial degree. Default is 0.
weektp = *integer*	Specify the trend polynomial degree. Default is 1.
weekfp = *integer*	Specify the filter polynomial degree. Default is 1.
weeksl = *integer*	Specify the length of the seasonal smoothing window (odd integers only). Default is 151.

weektl = *integer*	Specify the length of the trend smoothing window (odd integers only). Default is based upon the seasonal smoothing window length.
weekfl = *integer*	Specify the length of the filter smoothing window (odd integers only). Default is 1.
weekinits = *integer*	Specify number of inner iterations. Default is 1.
weekoutits = *integer*	Specify the number of outer iterations. Default is 15.

Day-of-month

monthsp = *integer*	Specify the seasonal polynomial degree. Default is 0.
monthtp = *integer*	Specify the trend polynomial degree. Default is 1.
monthfp = *integer*	Specify the filter polynomial degree. Default is 1.
monthsl = *integer*	Specify the length of the seasonal smoothing window (odd integers only). Default is 51.
monthtl = *integer*	Specify the length of the trend smoothing window (odd integers only). Default is based upon the seasonal smoothing window length.
monthfl = *integer*	Specify the length of the filter smoothing window (odd integers only). Default is 1.
monthinits = *integer*	Specify number of inner iterations. Default is 1.
monthoutits = *integer*	Specify the number of outer iterations. Default is 15.

Day-of-year

yearsp = *integer*	Specify the seasonal polynomial degree. Default is 0.
yeartp = *integer*	Specify the trend polynomial degree. Default is 1.
yearfp = *integer*	Specify the filter polynomial degree. Default is 1.
yearsl = *integer*	Specify the length of the seasonal smoothing window (odd integers only). Default is 13.
yeartl = *integer*	Specify the length of the trend smoothing window (odd integers only). Default is based upon the seasonal smoothing window length.
yearfl = *integer*	Specify the length of the filter smoothing window (odd integers only). Default is 1.
yearinits = *integer*	Specify number of inner iterations. Default is 1.
yearoutits = *integer*	Specify the number of outer iterations. Default is 15.

Example

```
elecdmd.dsa(forc="2015/6/30") elecdmd_adjusted
```

Performs daily seasonal adjustment on the ELECDMD series, specifying that the forecast end point should be 30 June 2015, and that the final adjusted series should be named ELECDMD_ADJUSTED.

```
elecdmd.dsa(fixedtrig, nodom, nodoy) elecdmd_adjusted @fa
    elecdmd_factors
```

Performs daily seasonal adjustment on ELECDMD, using a fixed number of trigonometric terms in the ARIMA step, and without using day-of-month or day-of-year STL. As well as saving the final adjusted series as ELECDMD_ADJUSTED, the final seasonal factor are also saved under ELECDMD_FACTORS.

Cross-references

See also `Series::x12` (p. 885), `Series::x13` (p. 890), and "Seasonal Adjustment" on page 507 of *User's Guide I*.

dups — Series Views

Duplicate observations display for observations in the series.

Syntax

series_name.**dups**(*opts*)

By default, EViews displays a summary table showing the number of duplicate groups of a given size, but you may use the options to display an alternative view.

Of particular note is that the spreadsheet and individual duplicates displays are interactive - clicking on rows in one will open the display to show the other. Thus, clicking on a duplicate in the spreadsheet view will jump to show all of the observations that share that duplicate. Similarly, clicking on an observation in the shared individual duplicates view will jump to the corresponding observation in the full spreadsheet.

Options

graph	Display observation graph showing duplicates.
sheet	Display spreadsheet view of duplicates.
individ	Display first individual duplicates.

Examples

```
ser1.dups
```

displays the duplicates summary for the series SER1.

```
ser1.dups(sheet)
```

displays a spreadsheet showing highlighted duplicates.

Cross-references

For a description of the duplicates view, see "Duplicates Analysis," on page 468 of *User's Guide I*.

edftest	Series Views

Computes goodness-of-fit tests based on the empirical distribution function.

Compute Kolmogorov-Smirnov, Lilliefors, Cramer-von Mises, Anderson-Darling, and Watson empirical distribution function tests using the current sample.

Syntax

series_name.**edftest***(options)*

Options

General Options

dist = *arg* *(default* = "nomal"*)*	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).
p1 = *number*	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.
p2 = *number*	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.
p3 = *number*	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.
prompt	Force the dialog to appear from within a program.
p	Print test results.

Estimation Options

The following options apply if iterative estimation of parameters is required:

b	Use Berndt-Hall-Hall-Hausman (BHHH) algorithm. The default is Marquardt.
m = *integer*	Maximum number of iterations.
c = *number*	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
s	Take starting values from the C coefficient vector. By default, EViews uses distribution specific starting values that typically are based on the method of the moments.

Examples

The command

```
x.edftest
```

uses the default settings to test whether data in the series X for the current sample comes from a normal distribution. Both the location and scale parameters are estimated from the data in X.

```
x.edftest(type=chisq)
```

tests whether the data in X follow a χ^2 distribution with degrees-of-freedom estimated from the data.

```
freeze(tab1) x.edftest(type=chisq, p1=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 465 of *User's Guide I* for a description of the goodness-of-fit tests.

See also `qqplot` (p. 1306).

ets	Series Procs

Perform Error-Trend-Season (ETS) exponential smoothing.

The `ets` procedure forecasts a series using the ETS model framework with state-space based likelihood calculations, support for model selection, and calculation of forecast standard errors.

The ETS framework defines an extended class of exponential smoothing models, including the standard exponential smoothing models (e.g., Holt and Holt-Winters additive and multiplicative models).

Syntax

series_name.**ets***(options) smooth_name*

You should enter the `ets` keyword followed by options and then the a name for the smoothed output series. You can specify the smoothing method (the default setting is additive error, no trend, no seasonality) and the smoothing options in the parenthesis.

Options

Forecast sample options

The forecast sample will start at the observation immediately after the estimation sample (the current workfile sample). The forecast endpoint is given by either:

forclen = *int*	Number of periods to forecast.
forc = "*date*"	Specify the date of the forecast end point.

One of these options is required.

General

prompt	Force the dialog to appear from within a program.
p	Print the view.

Model specification

e = *arg* (default = "a")	Set error type: "a" (additive), "m" (multiplicative), "e" (auto).
t = *arg* (default = "n")	Set trend type. *key* can be: "n" (none), "a" (additive), "m" (multiplicative), "ad" (additive dampened), "md" (multiplicative dampened), "e" (auto).
s = *arg* (default = "n")	Set season type. *key* can be: "n" (none), "a"(additive), "m" (multiplicative), "e" (auto).
modsel = *arg* (default = "aic")	Model selection method: "aic" (Akaike information criterion), "bic" (Bayesian information criterion/Schwartz criterion), "hq" (Hannan-Quinn information criterion), "amse" (average mean squared errors).
alpha = *arg*	Specify fixed value for level parameter α.
beta = *arg*	Specify fixed value for trend parameter β in models with trend.

gamma = *arg*	Specify fixed value for seasonal parameter γ in models with a seasonal component.
phi = *arg*	Specify fixed value for dampening parameter ϕ in models with dampened trends.
nomult	Do not allow multiplicative trend or seasonal terms. Only applies if the t = e or s = e options are set.

Optimization options

amse	Set Average Mean Square Error (AMSE) as the objective function (The default is log-likelihood as the objective function).
namse = *integer*	Specify the AMSE length—the number of observations over which to calculate AMSE if "amse" is selected.
c = *number*	Set the convergence criteria.
m = *integer*	Set the maximum number of iterations.
ustart	Employ user-supplied starting values (taken from the C vector in the workfile).
noi	Do not optimize the initial state values (fix at their starting values).

Output options

dgraph = *arg*	Include a decomposition graph for each specified element. *arg* may be composed of any of the following elements: "f" (forecast), "l" (level), "t" (trend), "s" (season).
dgopt = *arg* (default = "m")	Format for display of decomposition graph: "m" (multiple graph), "s" (single graph)
graph = *arg*	Include a comparison graph in the output for each specified element (if model selection is employed). *arg* may be composed of any of the following elements: "c" (forecast comparison) and "l" (likelihood comparison).
table = *arg*	Include a comparison table in the output (if model selection is employed). *arg* may be composed of any of the following elements: "c" (forecast comparison) and "l" (likelihood comparison).

level = *name*	Save the level component as a separate series in the workfile.
trend = *name*	Save the trend component as a separate series in the workfile (if applicable).
season = *name*	Save the seasonal component as a separate series in the workfile (if applicable).

Examples

```
sales.ets(e=a, t=n, s=a)sales_f
```

smooths the series SALES using the an ANN (additive error, no trend, no seasonal) model and creates the smoothed series named "SALES_F".

```
tb3.ets(e=e, t=e, s=n) tb3_smooth
```

will smooth TB3, automatically selecting the best smoothing model amongst the different Error and Trend specifications (the Seasonal specification is set at none).

```
sales.ets(e=a, t=a, s=a, dgopt=m, dgraph=flts)
```

will smooth the series SALES using the an AAA (additive error, additive trend, additive seasonal) model and display the output in a spool object which contains the actual and decomposition series (i.e., forecast, trend, level, and seasonal series) in multiple graphs.

```
sales.ets(e=a, t=a, s=a, level=level1, trend=trend1,
    season=season1, dgopt=s, dgraph=flts)
```

will smooth the series SALES using the an AAA (additive error, additive trend, additive seasonal) model, create the decomposition series named level, trend, and season series as level1, trend1, and season1, respectively, and display a spool object which contains the actual and decomposition graphs in a single graph.

```
tb3.ets(e=e, t=e, s=e, graph=cl)
```

will find out the best model amongst the different Error, Trend, and Seasonal specifications and present the estimation results in a spool object which contains the graphs with forecast and likelihood comparison graphs between all available models.

```
tb3.ets(e=a, t=e, s=e, amse, table=cl)
```

will search for the best model using average mean square errors calculations and display the estimation results in a spool object with forecast and likelihood comparison tables.

Cross-references

See "Exponential Smoothing" on page 599 of *User's Guide I* for a discussion of exponential smoothing methods.

See also `Series::smooth` (p. 850).

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,l) 1,2,3
```

Note that the last argument in the fill command above is the *letter* "l". The next three lines generate a dummy series D70S that takes the value one and two for observations from 1970:1 to 1979:4:

```
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 308 of the *Command and Programming Reference*).

forcavg	Series Procs

Average different forecasts of a series.

Syntax

series.**forcavg**(*options*) *forecast_data*

You should specify the forecast data to be averaged by entering a list of objects as *forecast_data*. The list may be a list of series objects, a group object, a series naming pattern (such as "f*" to indicate all series starting with the letter "F"), or a list of equation objects.

If a list of equations is entered, EViews will automatically forecast from those equation objects over the forecast sample (the current workfile sample).

Options

wgt = "key"	Set the type of averaging to use. *key* can be "mean" (default), "trmean" (trimmed-mean), "med" (median), "ols" (least squares weights), "mse" (mean square error weights), "ranks", (MSE ranks), "aic" (Smoothed AIC weights), or "sic" (BMA weights). "aic" and "sic" are only available if a list of equations is provided as the forecast_data.
trim = *num*	Set the level of trimming for the Trimmed mean method. *Num* should be a number between 1 and 100. Only applicable if the "trmean" option is used.
msepwr = *int*	Set the power to which the MSE values are raised in the MSE ranks method. Only applicable if the "mseranks" option is used.
s	Use a static (rather than dynamic) forecast when computing the forecasts over the training sample. Only applicable if *forecast_data* is a list of equation objects.
forcsmpl = *arg*	Forecast sample (optional). If forecast sample is not provided, the workfile sample will be employed.

trainsmpl = *arg*	Specify the sample used for calculating the averaging weights. Only applicable if the "ols", "mse", "mseranks", "aic" or "sic" options are used.
name = *arg*	Set the name of the final averaged series.
wgtname = *arg*	Save the weights into a vector in the workfile with the name *wgtname*.

Example

The commands

```
wfopen elecdmd.wf1
elecdmd.forcavg(trainsmpl="2012M1 2012M12", wgt=mse) elecf_fe*
```

open the workfile elecdmd.wf1 and then perform forecast averaging using the actual series ELECDMD, and the forecast series specified by the naming pattern ELECF_FE*.

The averaging method MSE is used. A training sample of 2012M1 to 2012M12 is used to calculate the weights in the MSE and MSE Ranks methods.

Cross-references

See "Forecast Averaging" on page 593 of *User's Guide I* for additional discussion.

See also `Series::forceval` (p. 793).

forceval	Series Views

Evaluate different forecasts of a series, and perform the forecast combination test.

Syntax

series.**forceval**(*options*) *forecast_data*

You should specify the forecast data to be evaluated by entering a list of objects as *forecast_data*. The list may be a list of series objects, a group object, a series naming pattern (such as "f*" to indicate all series starting with the letter "F"), or a list of equation objects.

If a list of equations is entered, EViews will automatically forecast from those equation objects over the evaluation sample (the current workfile sample).

Options

mean	Include the Mean averaging method.
trmean	Include the Trimmed mean averaging method.
median	Include the Median averaging method.

ols	Include the Least-squares averaging method.
mse	Include the Mean Square Error averaging method.
mseranks	Include the MSE ranks averaging method.
aic	Include the Smoothed AIC weights averaging method. Only applicable if *forecast_data* is a list of equation objects.
sic	Include the Bayesian model averaging method. Only applicable if *forecast_data* is a list of equation objects.
trim = *num*	Set the level of trimming for the Trimmed mean method. *Num* should be a number between 1 and 100. Only applicable if the "trmean" option is used.
msepwr = *int*	Set the power to which the MSE values are raised in the MSE ranks method. Only applicable if the "mseranks" option is used.
s	Use a static (rather than dynamic) forecast when computing the forecasts over the training sample. Only applicable if *forecast_data* is a list of equation objects.
trainsmpl = *arg*	Specify the sample used for calculating the averaging weights. Only applicable if the "ols", "mse", "mseranks", "aic" or "sic" options are used.
testname = *arg*	Save the combination test statistics into a matrix named *arg*.
statname = *arg*	Save the names of the best performing forecasts into an svector named *arg*.

Example

The commands

```
wfopen elecdmd.wf1
elecdmd.forcval(trainsmpl="2012M1 2012M12", mean, mse, mseranks,
    msepwr=2) elecf_fe*
```

open the workfile elecdmd.wf1 and then perform forecast evaluation using the actual series ELECDMD, and the forecast series specified by the naming pattern ELECF_FE*.

The averaging methods Mean, MSE and MSE Ranks are used, with the power of the MSE Ranks method set at "2". A training sample of 2012M1 to 2012M12 is used to calculate the weights in the MSE and MSE Ranks methods.

Cross-references

See "Label" on page 493 of *User's Guide I* for additional discussion.

See also Series::forcavg (p. 792).

freq	Series Views

Compute frequency tables.

The `freq` command performs a one-way frequency tabulation.

Frequencies are computed for the current sample of observations. Observations with NAs are dropped unless included by option. You may use options to control automatic binning (grouping) of values and the order of the entries of the table.

Syntax

series_name.**freq***(options)*

Options

dropna *(default)* / keepna	[Drop/Keep] NA as a category.
v = *integer* *(default* = 1000*)*	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*	(optional) Make bins if average count per distinct value is less than the specified number.
b = *integer* *(default* = 50*)*	Maximum number of categories to bin into if performing automatic binning.
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.

sort = *arg* *(default* = "lohi"*)*	Sort order for entries in the frequency table: high data value to low ("hilo"), low data value to high ("lohi" – *default*), high frequency to low ("freqhilo"), low frequency to high ("freqlohi").
prompt	Force the dialog to appear from within a program.
p	Print the table.

Examples

```
hrs.freq(nov, noa)
```

tabulates each value (no binning) of HRS, with entries in ascending value order. The table will display counts, percentages, and cumulative frequencies.

```
inc.freq(v=200, b=50, keepna, noa)
```

tabulates INC including NAs. The observations will be binned if INC has more than 200 distinct values; EViews will create at most 50 equal value-width bins. The number of bins may be smaller than 50.

```
inc.freq(sort=freqhilo)
```

tabulates INC with the table rows ordered from values with highest frequency to lowest.

Cross-references

See "One-Way Tabulation" on page 467 of *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 219 of *User's Guide I*.

See also `Link::link` (p. 528).

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. 832) for a discussion of the expressions allowed in `genr`.

hist	Series Views

Histogram and descriptive statistics of a series.

The `hist` command displays descriptive statistics and a histogram for the data in 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 450 of *User's Guide I* for a discussion of the descriptive statistics reported in the histogram view.

See `distplot` (p. 1283) for a more fully-featured and customizable method of constructing histograms and `Series::stats` (p. 855) stats for a view with a more extensive set of basic descriptive statistics.

hpf	Series Procs

Smooth a series using the Hodrick-Prescott filter.

Syntax

series_name.**hpf***(options) filtered_name [@ cycle_name]*

You may need to prepend the "show" keyword to display the graph the smoothed and original series.

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 = *arg*	Set smoothing parameter value to *arg*; a larger number results in greater smoothing.
power = *arg* *(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 *arg*, and multiplied by 1600). Hodrick and Prescott recommend the value 2; Ravn and Uhlig recommend the value 4.
m = *arg* *(default = 1)*	Set number of iterations.
ic	Use information criteria to determine the optimal number of iterations.
prompt	Force the dialog to appear from within a program.

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_hpmodsel
```

smooths the GDP series with a smoothing parameter "1000" and saves the smoothed series as GDP_HP.

```
gdp.hpf(power=4) gdp_hp @ gdp_cycle
```

smooths the same series with a power parameter of "4" and saves the smoothed series as GDP_HP, and the cycle series as GDP_CYCLE.

Cross-references

See "Hodrick-Prescott Filter" on page 625 of *User's Guide I* for details.

insertobs	Series Procs

Shift the observations of the series up or downwards, inserting blank observations.

Syntax

series_name.**insertobs***(startpoint) n*

Where *startpoint* specifies the first or last observation from which the observations are shifted. For dated workfiles, *startpoint* should be entered as a date. For panels and non-dated workfiles *startpoint* should be an observation number.

n specifies the number of observations shifted.

Examples

```
x.insertobs(1952q2) 2
```

Inserts 2 new observations beginning at observation 1952 quarter 2. The previous value associated with 1952Q2 will now correspond to 1952Q4.

```
y.insertobs(10) -5
```

Inserts 5 new observations ending at observation number 10 in the workfile.

ipolate	Series Procs

Fill in missing values, or NAs, within a series by interpolating from values that are not missing.

Syntax

series_name.**ipolate***(options) series_name*

Options

type = *key*	Specify the interpolation method. *key* is either “lin” (linear, default), “log” (log-linear), “cs” (Cardinal spline), “cr” (Catmull-Rom spline), “cb” (Cubic spline), “lcs” (log-cardinal spline), “lcr” (log-Catmull-Rom spline), or “lcb” (log-cubic spline).
tension = *number*	Sets the tension parameter for the Cardinal spline method of interpolation. *number* should be a number between 0 and 1.
f = *arg* (default = “actual”)	Out-of-sample fill behavior: “actual” (fill observations outside the interpolated sample with values from the source series). “na” (fill observations outside the sample with missing values”
prompt	Force the dialog to appear from within a program.

Examples

The following lines interpolate the missing values of series X1 using linear interpolation, and store the new interpolated series with a name X_INTER:

```
x1.ipolate x_inter
```

This line performs the same interpolation, but this time using the Cardinal spline, with a tension value of 0.8:

```
x1.ipolate(type=cs, tension=0.8) x_inter
```

Cross-references

See “Interpolate” on page 504 of *User’s Guide I* for discussion.

jdemetra	Series Procs

Executes the JDemetra+ seasonal adjustment routine on the series.

JDemetra+ is available only for quarterly and monthly series, and only performs X-13 style seasonal adjustment.

Syntax

series_name.**jdemetra***(options)*

Options

spec = *arg*	Set the JDemetra+ default specification. *arg* can be "X11", "RSA0", "RSA1", "RSA2c", "RSA3", "RSA4c", "RSA5c". The default is "RSA4c".
d10	Export the seasonal factors d10 series to the workfile.
nod11	Do not export the seasonal adjustment d11 series to the workfile.
d12	Export the trend d12 series to the workfile.
d13	Export the irregular component d13 series to the workfile.
suffix = *arg*	Suffix to add to the exported series names before the series type. For example, if the underlying series is named GDP, and a suffix of "_JDSA" is provided, the exported d11 series will be named GDP_JDSA_D11. By default, no additional suffix is provided, so the d11 series would be named GDP_D11.
tform = *arg*	Set the dependent variable transformation in the pre-adjustment regression. *arg* can be "none", "auto", "log". If omitted, JDemetra+ will use the transformation setting of the default specification.
noarma	Do not perform ARIMA estimation during the pre-adjustment regression of the seasonal adjustment. If one of the noarma, autoarma and fixedarma options are not provided, JDemetra+ will use the ARIMA estimation type of the default specification.
autoarma	Perform automatic ARIMA order selection during the pre-adjustment regression of the seasonal adjustment. If one of the noarma, autoarma and fixedarma options are not provided, JDemetra+ will use the ARIMA estimation type of the default specification.
fixedarma	Perform ARIMA estimation during the pre-adjustment regression of the seasonal adjustment. If one of the noarma, autoarma and fixedarma options are not provided, JDemetra+ will use the ARIMA estimation type of the default specification.
ar = *int*	If using fixedarma, specify the AR order. If omitted, JDemetra+ will use its default ARIMA model order.
ma = *int*	If using fixedarma, specify the MA order. If omitted, JDemetra+ will use its default ARIMA model order.
d = *int*	If using fixedarma, specify the ARIMA differencing. If omitted, JDemetra+ will use its default ARIMA model order.

sar = *int*	If using fixedarma, specify the seasonal AR order. If omitted, JDemetra+ will use its default ARIMA model order.
sma = *int*	If using fixedarma, specify the seasonal MA order. If omitted, JDemetra+ will use its default ARIMA model order.
sd = *int*	If using fixedarma, specify the ARIMA seasonal differencing. If omitted, JDemetra+ will use its default ARIMA model order.
outliers	Include automatic outlier detection in the pre-adjustment regression regression. If omitted, JDemetra+ will use the outlier detection setting of the default specification.
nooutliers	Do not include automatic outlier detection in the pre-adjustment regression. If omitted, JDemetra+ will use the outlier detection setting of the default specification.
tradedays = *arg*	Specify the type of trading day adjustment made in the pre-adjustment regression. *arg* can be "none", "td2c", "td2", "td3", "td3c", "td4", "td7". If omitted, JDemetra+ will use the trading day type set by the default specification.
leapyr	Use the leap year adjustment in the pre-adjustment regression. If both leapyr and noleapyr are omitted, JDemetra+ will use the option set by the default specification.
noleapyr	Do not use the leap year adjustment in the pre-adjustment regression. If both leapyr and noleapyr are omitted, JDemetra+ will use the option set by the default specification.
fcast	Forecast from the pre-adjustment regression model. If fcast and nofcast are omitted, JDemetra+ will use the setting of the default specification.
nofcast	Do not forecast from the pre-adjustment regression model. If fcast and nofcast are omitted, JDemetra+ will use the setting of the default specification.

flen = *int*	If forecasting the pre-adjustment regression model, set the number of observations to be forecast. If omitted, a year of observations will be forecasted.
userregs = *"arg"*	Provide a list of user-variables used in the pre-adjustment regression model. *arg* should be a space delimited list of series objects or series generation expressions, surrounded in quotes.
usertypes = *"arg"*	Specify the variable types for the user-variables provided with a userregs = option. *arg* should be a space delimited list of keywords, with one keyword per user-variable, in the same order as given in the userregs option. Keyword values can be; "undef" (undefined), "ser" (series), "trnd" (trend), "seas" (seasonal), "sa" seasonally adjusted, "irreg" (irregular), or "cal" (calendar). If omitted, user-variables are set to undefined.

Examples

```
hstartsnsa.jdemetra(d10, suffix=_jdm, spec=rsa5c, nooutliers)
```

This will execute the JDemetra+ X-13 based seasonal adjustment routine on the series HSTARTSNSA. The JDemetra+ RSA5c default specification is used, but outlier detection is turned off. The seasonal factors and seasonally adjusted data are stored into the workfile with the names HSTARTSNA_JDM_D10 and HSTARTSNA_JDM_D11 respectively.

Cross-references

See "JDemetra+" on page 507 of *User's Guide II* for discussion.

kdensity

Series Views

Kernel density plots.

The `kdensity` command is no longer supported. See `distplot` (p. 1283).

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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User's Guide I* for a discussion of labels.

See also `Series::displayname` (p. 780).

lrvar	Series Views

Compute the symmetric, one-sided, or strict one-sided long-run variance of a series.

Syntax

Series View: series_name.**lrvar**(*options*)

Options

window = *arg*	Type of long-run covariance to compute: “sym” (symmetric), “lower” (lower - lags in columns), “slower” (strict lower - lags only), “upper” (upper - leads in columns), “supper” (strict upper - leads only)
noc	Do not remove means (center data) prior to whitening.
out = *arg*	Name of output sym or matrix (optional)
panout = arg	Name of ee output matrix (optional).
prompt	Force the dialog to appear from within a program.
p	Print results.

Whitening Options

lag = *arg*	Lag specification: *integer* (user-specified number of lags), “a” (automatic selection).
infosel = *arg* *(default* = “aic”*)*	Information criterion for automatic selection: “aic” (Akaike), “sic” (Schwarz), “hqc” (Hannan-Quinn) (if “lag = a”).
maxlag = *integer*	Maximum lag-length for automatic selection *(optional)* (if “lag = a”). The default is an observation-based maximum of $T^{1/3}$.

Kernel Options

kern = *arg* *(default* = “bart”*)*	Kernel shape: “none” (no kernel), “bart” (Bartlett, *default*), “bohman” (Bohman), “daniell” (Daniel), “parzen” (Parzen), “parzriesz” (Parzen-Riesz), “parzgeo” (Parzen-Geometric), “parzcauchy” (Parzen-Cauchy), “quadspec” (Quadratic Spectral), “trunc” (Truncated), “thamm” (Tukey-Hamming), “thann” (Tukey-Hanning), “tparz” (Tukey-Parzen), “user” (User-specified; see “kernwgt = ” below).
kernwgt = *vector*	User-specified kernel weight vector (if “kern = user”).
bw = *arg* *(default* = ”nwfixed”*)*	Bandwidth: “fixednw” (Newey-West fixed), “andrews” (Andrews automatic), “neweywest” (Newey-West automatic), *number* (User-specified bandwidth).

nwlag = *integer*	Newey-West lag-selection parameter for use in nonparametric bandwidth selection (if "bw = neweywest").
bwoffset = *integer* (*default* = 0)	Apply integer offset to bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").
bwint	Use integer portion of bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").

Examples

```
ser1.lrvar(out=outsym)
```

computes the symmetric long-run variance of the series SER1 and saves the results in the output sym matrix OUTSYM.

```
ser1.lrvar(kern=quadspec, bw=andrews)
```

computes the long-run variance SER1 using the quadratic spectral kernel, Andrews automatic bandwidth.

```
ser1.lrvar(kern=quadspec, lag=3, bw=andrews)
```

performs the same calculation but uses AR(3) prewhitening prior to computing the kernel estimator.

```
ser1.lrvar(kern=none, window=upper, lag=a, infosel=aic,
    bw=neweywest, rwgt=res)
```

computes parametric VAR estimates of the upper long-run variance using an AIC based automatic lag-length prewhitening procedure, Newey-West bandwidth selection, and row weight series RES.

Cross-references

See "Long-run Variance," on page 491 of *User's Guide I*, "Panel Long-run Variances," on page 1449 of *User's Guide II*, Appendix F. "Long-run Covariance Estimation," on page 1559 of *User's Guide II*.

See also `Group::lrcov` (p. 479).

makepanpcomp	Series Procs

Save the scores from a principal components analysis of a panel series.

Syntax

series_name.**makepanpcomp***(options) output_list*

where the *output_list* is a list of names identifying the saved components. 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

scale = *arg* *(default* = "normload"*)*	Diagonal matrix scaling of the loadings and the scores: normalize loadings ("normload"), normalize scores ("normscores"), symmetric weighting ("symmetric"), user-specified (*arg* = *number*).
cpnorm	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).
eigval = *vec_name*	Specify name of vector to hold the saved the eigenvalues in workfile.
eigvec = *mat_name*	Specify name of matrix to hold the save the eigenvectors in workfile.
prompt	Force the dialog to appear from within a program.

Covariance Options

period	Compute period (within cross-section) panel covariances and related statistics. The default is to compute contemporaneous (between cross-section) measures.
cov = *arg* *(default* = "corr"*)*	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.
pairwise	Compute using pairwise deletion of observations with missing cases (pairwise samples).
df	Compute covariances with a degree-of-freedom correction accounting for the estimation of the mean (for centered 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

```
ser1.makepanpcomp(eigval=v1, eigvec=m1) comp1 comp2 comp3
```

saves the first three principal components (in normalized loadings form) of the panel series SER1 to the workfile. The components will have variances that are proportional to the eigenvalues of the contemporaneous correlation matrix. In addition, the vector V1 and matrix M1 will contain the eigenvectors and eigenvalues of the decomposition.

```
smpl 1990 2010
ser.makepanpcomp(period, cov=rcorr, scale=normscore) comp1
```

saves the first principal component of the period (within cross-section) Spearman rank correlations. The scores will be normalized so that the variances of the scores are equal to 1.

Cross-references

See "Saving Component Scores," beginning on page 1443 of *User's Guide I* and "Panel Principal Components" on page 1441 of *User's Guide II* for further discussion.

To display the results of the panel principal components decomposition, see `Series::panpcomp` (p. 822).

makewavelets	Series Procs

Save wavelet decomposition results to the workfile.

Syntax

Series View: series_name.**makewaveobj***(options)*

Options

Basic Options

proc = *arg* *(default* = "decomp"*)*	Wavelet analysis type: "decomp" (transform), "anova" (variance decomposition), "outlier" (outlier detection), "threshold" (threshold - denoising).

Wavelet Transform Options

transform = *arg* *(default* = "dwt"*)*	Wavelet transform type: "dwt" (discrete wavelet transform – DWT), "modwt" (maximum overlap DWT – MODWT), "mra" (DWT multiresolution analysis – DWT MRA), or "momra" (MODWT MRA). Note that when performing DWT or MRA, if the series length is not dyadic, a dyadic fix may be set with the "fixlen = " option
fixlen = *arg* *(default* = "mean"*)*	Fix dyadic lengths in DWT and MRA transforms: "zeros" (pad remainder with zeros), "mean" (pad remainder with mean of series), "median" (pad remainder with median of series), "shorten" (cut series length to dyadic length preceding series length).
maxscale = *integer* *(default* = max possible)	Maximum scale for wavelet transform. The max possible is obtained as follows. Let T denote the series length and decompose T into its dyadic component and a remainder: $T = 2^z + r$, $r \geq 0$. The default *maxscale* m is then set with the following rules: DWT: (1) if $r = 0$ then $m = z$, otherwise (2) if expanding the series, $m = z + 1$ and (3) if contracting the series $m = z$. MODWT: $m = z$.
filter = *arg* *(default* = "h"*)*	Wavelet filter class: "h" (Haar), "d" (Daubechies), "la" (least asymmetric). If "filter = h" or "filter = la", the filter length may be specified using "flen = ". Wavelet filter boundary conditions are specified using the "bound = " option

flen = *integer*	Wavelet filter excess length as an even number between 2 and 20. For use when "filter = d" (*default* = 4) or "filter = la" (*default* = 8).
bound = *arg (default* = "p"*)*	Filter boundary handling: "p" (periodic), "r" (reflective).
basename = *arg* (*default* is series name)	Basename for output: 1. When "transform = dwt" or "transform = modwt": Wavelet coefficient output will be saved in vectors with names given by basename followed by "_w#" where "#" is the scale level. Scaling coefficient output will be saved in vectors with names given by basename followed by "_v#" where "#" is the scale level. 2. When "transform = mra" or "transform = momra": Detail output will be saved in vectors with names given by basename followed by "_d#" where "#" is the scale level. Smooth output will be saved in vectors with names given by basename followed by "_s#" where "#" is the scale level.

Wavelet Variance Decomposition Options

variance = *arg (default* = "nobias"*)*	Wavelet variance type: "nobias" (unbiased variance), "bias" (biased variance).
ci = *arg (default* = "none"*)*	Confidence interval type: "none" (no CIs computed), "gauss" (asymptotic normal), "chisq" (asymptotic chi-square), "blimit" (band-limited).
cilevel = *arg (default* = 0.95*)*	Confidence interval coverage as a number between 0 and 1.
transform = *arg (default* = "dwt"*)*	Wavelet transform type: "dwt" (discrete wavelet transform – DWT), "modwt" (maximum overlap DWT – MODWT). Note that when performing DWT, if the series length is not dyadic, a dyadic fix may be set with the "fixlen = " option
fixlen = *arg (default* = "mean"*)*	Fix dyadic lengths in DWT and MRA transforms: "zeros" (pad remainder with zeros), "mean" (pad remainder with mean of series), "median" (pad remainder with median of series), "shorten" (cut series length to dyadic length preceding series length).

maxscale = *integer* (*default* = max possible)	Maximum scale for wavelet transform. The max possible is obtained as follows. Let T denote the series length and decompose T into its dyadic component and a remainder: $T = 2^z + r$, $r \geq 0$. The default *maxscale* m is then set with the following rules: DWT: (1) if $r = 0$ then $m = z$, otherwise (2) if expanding the series, $m = z + 1$ and (3) if contracting the series $m = z$. MODWT: $m = z$.
filter = *arg* (*default* = "h")	Wavelet filter class: "h" (Haar), "d" (Daubechies), "la" (least asymmetric). If "filter = h" or "filter = la", the filter length may be specified using "flen = ". Wavelet filter boundary conditions are specified using the "bound = " option
flen = *integer*	Wavelet filter excess length as an even number between 2 and 20. For use when "filter = d" (*default* = 4) or "filter = la" (*default* = 8).
bound = *arg* (*default* = "p")	Filter boundary handling: "p" (periodic), "r" (reflective).
basename = *arg* (*default* is series name)	Basename for variance output. Variance output will be saved in vector with name given by basename followed by "_var". Confidence interval output will be saved in matrix with name given by basename followed by "_varci".

Wavelet Threshold and Outlier Options

threshtype = *arg* (*default* = "soft")	Wavelet threshold type: "hard" (hard thresholding), "soft" (soft thresholding).
threshlim = *arg* (*default* = "universal")	Wavelet threshold limit type: "universal" (universal), "adaptive" (universal adaptive), "minimax" (minimax), "sureshrink" (SureShrink), "fdr" (false discovery rate). If "threshlim = sureshrink", the grid length may be specified using "ssglen = ". If "threshlim = fdr", the significance level may be specified using "fdrsig = "
wavevar = *arg* (*default* = "gauss")	Wavelet coefficient variance method: "mean" (mean absolute deviation), "gauss" (median absolute deviation with Gaussian adjustment), "median" (median absolute deviation), "meanmedian" (mean median absolute deviation).

sslen = *arg* *(default* = 10)	Grid length used in determining the SureShrink limit.
fdrsig = *arg* *(default* = .05)	Significance level as a number between 0 and 1 for false discovery rate limit determination.
transform = *arg* *(default* = "dwt"*)*	Wavelet transform type: "dwt" (discrete wavelet transform – DWT), "modwt" (maximum overlap DWT – MODWT). Note that when performing DWT, if the series length is not dyadic, a dyadic fix may be set with the "fixlen = " option
fixlen = *arg* *(default* = "mean"*)*	Fix dyadic lengths in DWT and MRA transforms: "zeros" (pad remainder with zeros), "mean" (pad remainder with mean of series), "median" (pad remainder with median of series), "shorten" (cut series length to dyadic length preceding series length).
maxscale = *integer* *(default* = max possible)	Maximum scale for wavelet transform. The max possible is obtained as follows. Let T denote the series length and decompose T into its dyadic component and a remainder: $T = 2^z + r$, $r \geq 0$. The default *maxscale* m is then set with the following rules: DWT: (1) if $r = 0$ then $m = z$, otherwise (2) if expanding the series, $m = z + 1$ and (3) if contracting the series $m = z$. MODWT: $m = z$.
filter = *arg* *(default* = "h"*)*	Wavelet filter class: "h" (Haar), "d" (Daubechies), "la" (least asymmetric). If "filter = h" or "filter = la", the filter length may be specified using "flen = ". Wavelet filter boundary conditions are specified using the "bound = " option
flen = *integer*	Wavelet filter excess length as an even number between 2 and 20. For use when "filter = d" (*default* = 4) or "filter = la" (*default* = 8).
bound = *arg (default* = "p"*)*	Filter boundary handling: "p" (periodic), "r" (reflective).

Outlier Output Options

basename = *arg* (*default* is series name)	Basename for outlier output (when "proc = outlier"). Output will be saved in vectors with name given by basename followed by "_olr#" where "#" is the scale level.

Threshold Output Options

basename = *arg* (*default* is series name)	Basename for threshold output (when "proc = threshold"): Fit (signal) output will be saved in vector with name given by basename followed by "_sig_th". Noise output will be saved in vector with name given by basename followed by "_res_th". Threshold coefficient output will be saved in vectors with names given by basename followed by "_w#_th" (threshold coefficients), and "_v#" (scaling coefficients) where "#" is the scale level.

Examples

```
srs.makewaveobj(base=out)
```

creates vectors OUT_W1, OUT_W2, *etc.*, and OUT_V1, OUT_V2, *etc.*, associated with the wavelet coefficients W and scaling coefficients V from the DWT using a Haar filter.

```
srs.makewaveobj(transform=modwt, filter=d, flen=6, base=out)
```

creates vectors OUT_V1, OUT_V2, *etc.*, associated with the scaling coefficients V from the MODWT using a Daubechies filter of length 6.

```
srs.makewaveobj(proc=anova, maxscale=2, fixlen=median, base=out)
```

creates the vector OUT_VAR with containing variance decomposition per scale using a DWT with maximum scale 2, and a series length adjustment by padding with median values of the series SRS.

```
srs.makewaveobj(proc=outlier, base=out)
```

creates vectors OUT_OLR1, OUT_OLR1, *etc.* with outlier identifiers using a DWT, Haar filter, soft threshold, universal threshold limit, and Gaussian median computation for the wavelet coefficient variance.

```
srs.makewaveobj(proc=threshold, transform=modwt, threshtype=hard,
    threshlim=sureshrink, base=OUT)
```

creates a vector OUT_SIG_TH containing values for the thresholded series, a vector OUT-_RES_TH containing noise values from the thresholding procedure, and vectors OUT-_W1_TH, OUT_W2_TH, *etc.* and OUT_W1_TH, CVEC_W2_TH associated with the thresholded wavelet coefficients W and the original scaling coefficient V for the series SRS. The underlying computation uses a MODWT with a Haar filter, a hard threshold with a SureShrink threshold limit.Cross-references

Examples

See "Wavelet Analysis" on page 859 and "Wavelet Variance Decomposition" on page 867 of *User's Guide II* for discussion. See also "Wavelet Objects" on page 635 of *User's Guide I*.

See also `Series::wavedecomp` (p. 878), `Series::waveanova` (p. 876), `Series::waveoutlier` (p. 880), and `Series::wavethresh` (p. 883).

makewhiten	Series Procs

Whiten the series.

Estimate an AR(p), compute the residuals, and save the results into a whitened series.

Syntax

Series View: series_name.**makewhiten**(*options*) *out_specification*

where *out_name* is either a name for the output series or a wildcard expression. Note that a wildcard may not be used if the original group contains series expressions.

Options

lag = *arg* *(default* = 1*)*	Lag specification: *integer* (user-specified number of lags), "a" (automatic selection).
noc	Do not remove means (center data) prior to whitening.
infosel = *arg* *(default* = "aic"*)*	Information criterion for automatic selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn).
maxlag = *integer*	Maximum lag-length for automatic selection *(optional)*. The default is an observation-based maximum of the integer portion of $T^{1/3}$.
prompt	Force the dialog to appear from within a program.

Examples

```
ser1.makewhiten(lag=a, infosel=sic, maxlag=10) *a
```

whitens the series in GRP1 using a VAR with auto-selected number of lags based on the SIC information criterion and a maximum of 10 lags. The resulting series is named ASER1.

```
ser1.makewhiten(noc, lag=5) aser1
```

whitens the series using a no-constant VAR and 5 lags.

Cross-references

See "Make Whitened" on page 715 of *User's Guide I* for details.

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 235 of *User's Guide I* for a discussion of valmap objects in EViews.

movereg	Series Procs

Seasonally adjust series using the movereg method.

movereg is only available for weekly data.

Syntax:

series.**movereg***(options) [@ao additiveoutliers] [@ls levelshiftoutliers] [@holiday holiday(holiday weighting)] [@user userregs]*

You should follow the *movereg* command with any additive or level-shift outliers, holiday events or user variables to be included as part of the procedure. Outliers are specified as full dates or in "weeknumber year" format. Holidays are specified with one of the built-in holiday keywords, followed by its holiday weighting pattern. Available holiday keywords are: "easter", "labor", "newyear", "memorial", "taxday", "christmas", "july4th", "february", "thanksgiving", "mlking", "veterans", and "columbus".

Options

Sa = *name*	Specify the name of the output seasonal adjusted data. If not specified the output series will be the name of the underlying series with an appended _sa
Facname = *name*	Specify the name of the output seasonal factors. If not specified the output series will be the name of the underlying series with an appended _saf
Holname = *name*	Specify the name of the output holiday series.
Outname = *name*	Specify the name of the output outlier series.
Nfilt = *integer*	Specify the width of the detrending filter. Default is 2.
Nfs = *integer*	Specify number of trigonometric terms. Default is 60. Must be a positive even integer.
phi = *number*	Specify the AR coefficient. Default is 0.4.
Sigr = *number*	Specify the variance ratio parameter. Default is 16.

Examples

```
icnsa.movereg
```

Performs movereg seasonal adjustment on the series ICNSA, saving the adjusted data and factors under the default names ICNSA_SA and ICNSA_SAF.

```
icnsa.movereg(outname=icnsa_hol) @ao  37 2001 @holiday
    christmas(5, 4, 1, 1, 1, 2, 2)
```

Performs a seasonal adjustment with the 37th week of 2001 as an additive outlier, and specifying an adjustment for Christmas, where the Christmas effect is felt over 5 weeks, 3 weeks before Christmas through one after, with the week of Christmas and after Christmas having slightly more weight. The output holiday series is stored under ICNSA_HOL.

olepush

Series Procs

Push updates to OLE linked objects in open applications.

Syntax

series_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

outliers	Series Views

Detect outlying observations.

Use Tukey fences, mean/standard deviation fences, wavelet outliers, or ARMA outliers detection methods to identify observations that deviate from the natural pattern of the underlying series.

Syntax

series_name.**outliers***(options)*

Options

hp	Perform the Hodrick-Prescott filter on the series, and then detect outliers on the cycle series.
diff	First-difference the series before detecting outliers.
sens = *arg*	Set the sensitivity level. Valid arguments are “low”, “medium” (default), and “high”.
nofence	Do not perform Tukey and mean/standard deviation fences.
nowave	Do not perform Wavelet Outlier detection.
arma	Perform ARMA based outlier detection (this option is turned on by default in dated workfiles).
noarma	Do not perform ARMA based outlier detection (this option is turned on by default in undated workfiles).
tukeyk = *arg*	Set the value *k* in the Tukey fence detection routine. This will override the value of *k* set by the sens = option.
meanstdevk = *arg*	Set the value *k* in the mean/standard deviation fence detection routine. This will override the value of *k* set by the sens = option.
wavesig = *arg*	Set the value false discovery rate significance value used in the Wavelet Outlier detection routine. This will override the value set by the sens = option.
armac = *arg*	Set the value *c* in the ARMA outlier detection routine. This will override the value of *c* set by the sens = option.
setsmpl	Set the workfile sample to be any observations identified as outliers.
excludesmpl	Set the workfile sample to be any observations identified as not outliers.

series = *name*	Create a new series in the workfile, named *name*, containing a value of 1 for any observations identified as an outlier, and a value of 0 for any observation identified as not an outlier.
datestring = *name*	Create a new string object in the workfile containing the dates (or observation identifiers) for any observations identified as an outlier.
nogrlabels	Turn off observation labels on the outlier graph.
prompt	Force the dialog to appear from within a program.
p	Print results.

Examples

```
gdpc1.outliers(hp, nowave, sens=high, meanstdevk=3)
```

Performs outlier detection on the cycle series from a Hodrick-Prescott filter of the GDPC1 series, opting to not use Wavelet Outlier detection, and setting the sensitivity of the detection to "high", but overwriting the *k* value used in the mean/standard deviation fence to 3.

Cross-references

See "Outlier Detection," on page 478 in *User's Guide I*. See also `Equation::fitoutliers` (p. 137).

pancov	Series Views

Compute covariances, correlations, and other measures of association for a panel series.

You may compute measures related to Pearson product-moment (ordinary) covariances and correlations, Spearman rank covariances, or Kendall's tau along with test statistics for evaluating whether the correlations are equal to zero.

Syntax

series_name.**pancov**(*options*) *[keywords]*

By default, EViews will compute the contemporaneous (between cross-section) covariances, correlations and related statistics for the panel series. You may use the "period" option to instruct EViews to compute the between period (within cross-section) measures.

You should specify keywords indicating the statistics you wish to display from the list below.

You may specify keywords from one of the four sets (Pearson correlation, Spearman rank 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.

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.

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.

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.

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.

Note that `cases` and `obs` are available for each of the methods.

Options

period	Compute period (within cross-section) panel covariances and related statistics. The default is to compute contemporaneous (between cross-section) measures.
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).
multi = *arg* *(default* = "none")	Adjustment to *p*-values for multiple comparisons: none ("none"), Bonferroni ("bonferroni"), Dunn-Sidak ("dunn").
outfmt = *arg* *(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").
prompt	Force the dialog to appear from within a program.
p	Print the result.

Examples

```
ser1.pancov
```

displays the contemporaneous Pearson covariance matrix of SER1 using the cross-sections in sample.

```
ser1.pancov corr stat prob
```

displays a table containing the contemporaneous Pearson correlation matrix for SER1, along with t-statistics for testing for zero correlation, and associated p-values,.

```
smpl 1990 2010
ser1.pancov(period, pairwise) taub taustat tauprob
```

computes the between period Kendall's tau-b, score statistic, and p-value for the score statistic, for the periods in the sample "1990 2010" using samples with pairwise missing value exclusion.

```
ser1.pancov(out=aa, list) cor
```

computes the contemporaneous Pearson correlation for the series SER1, displays it in list form, and saves the results in the symmetric matrix object AACORR.

Cross-references

See "Covariance Analysis" on page 668 of *User's Guide I* and "Panel Covariances" on page 1436 of *User's Guide II* for discussion.

To display the results of the panel principal components decomposition, see `Series::panpcomp` (p. 822).

See `Group::cor` (p. 451) in the *Command and Programming Reference* for the command to compute these measures across series.

panpcomp	Series Views

Panel principal components analysis.

Syntax

group_name.**panpcomp***(options) [indices]*

where the elements to display in loadings, scores, and biplot graph form ("out = loadings", "out = scores" or "out = biplot") are given by the optional *indices*, (*e.g.*, "1 2 3" or "2 3"). If *indices* is not provided, the first two elements will be displayed.

Basic Options

out = *arg* *(default* = "table"*)*	Output type: eigenvector/eigenvalue table ("table"), eigenvalues graph ("graph"), loadings graph ("loadings"), scores graph ("scores"), biplot ("biplot").

eigval = *vec_name*	Specify name of vector to hold the saved the eigenvalues in workfile.
eigvec = *mat_name*	Specify name of matrix to hold the save the eigenvectors in workfile.
prompt	Force the dialog to appear from within a program.
p	Print results.

Table and Eigenvalues Plot Options

The number of elements to display in the table and eigenvalue graph form is given by the minimum of the elements specified using the "n = ", "mineigen = " and "cproport = " options.

The default eigenvalue graph shows a scree plot of the ordered eigenvalues. You may use the "scree", "cproport", and "diff" option keywords to display any combination of the scree plot, cumulative eigenvalue proportions plot, or eigenvalue difference plot.

n = *arg (default* = all*)*	Maximum number of components.
mineigen = *arg (default* = 0*)*	Minimum eigenvalue.
cproport = *arg (default* = 1.0*)*	Cumulative proportion of eigenvalue total to attain.
scree	Display a scree plot of the eigenvalues (if "output = graph).
diff	Display a graph of the eigenvalue differences (if "output = graph).
cproport	Display a graph of the cumulative proportions (if "output = graph).

Loadings, Scores, Biplot Graph Options

scale = *arg, (default* = "normload"*)*	Diagonal matrix scaling of the loadings and the scores: normalize loadings ("normload"), normalize scores ("normscores"), symmetric weighting ("symmetric"), user-specified (*arg* = *number*).
cpnorm	Compute the normalization for the scores so that cross-products match the target (by default, EViews chooses a normalization scale so that the moments of the scores match the target).
nocenter	Do not center the elements in the graph.
mult = *arg (default* = "first"*)*	Multiple graph options: first versus remainder ("first"), pairwise ("pair"), all pairs arrayed in lower triangle ("lt")

labels = *arg* *(default* = "outlier"*)*	Scores label options: identify outliers only ("outlier"), all points ("all"), none ("none").
labelprob = *arg* *(default* = 0.1*)*	Outlier label probability (if "labels = outlier").
autoscale = *arg* *(default* = 1.0*)*	Rescaling factor for auto-scaling.
userscale = *arg*	User-specified scaling.

Covariance Options

period	Compute period (within cross-section) panel covariances and related statistics. The default is to compute contemporaneous (between cross-section) measures.
cov = *arg* *(default* = "corr"*)*	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.
pairwise	Compute using pairwise deletion of observations with missing cases (pairwise samples).
df	Compute covariances with a degree-of-freedom correction accounting for the mean (for centered specifications). The default behavior in these cases is to perform no adjustment (*e.g.* – compute sample covariance dividing by n rather than $n-1$).

Examples

```
ser1.panpcomp(eigval=v1, eigvec=m1)
```

computes the principal components decomposition of the contemporaneous (between cross-section) Pearson correlation matrix for the series SER1.

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.

```
ser1.panpcomp(out=graph)
ser2.panpcomp(out=graph, scree, cproport)
```

displays a screen plot of the eigenvalues, and a graph containing both a screen plot and a plot of the cumulative eigenvalue proportions.

```
ser.panpcomp(period, cov=rcorr, out=loading)
```

displays a loadings plot for the principal component decomposition of the period (within cross-section) Spearman rank correlation matrix, and

```
ser.panpcomp(period, cov=rcorr, out=biplot, scale=symmetric,
    mult=lt) 1 2 3
```

displays a symmetric biplot of the period Spearman correlation matrix for all three pairwise comparisons.

Cross-references

See "Principal Components" on page 685 of *User's Guide I* and "Panel Principal Components" on page 1441 of *User's Guide II* for further discussion.

To compute principal components scores and save them in series in the workfile, see `Series::makepanpcomp` (p. 807).

prophet	Series Procs

Performs Facebook's Prophet forecasting on the underlying series.

Prophet requires the Prophet Python library installed with EViews.

Syntax

series_name.**prophet***(options) forecast_name [lower_name upper_name]*

Enter a name for the forecast series, and optionally, a name for the lower and upper bounds of the forecast.

Options

cpscale = *number*	Set the change point prior scale.
growth = *arg*	Set the growth type. *arg* can be "linear" (default) or "log".
season = *arg*	Set the seasonality type. *arg* can be "add" (additive, default) or "mult" (multiplicative).
f = *arg*	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, default), or "forc" (fill training-data observations with in-sample forecasts, and other data with NA).
prompt	Force the dialog to appear from within a program.

Forecast sample options

The forecast sample will start at the observation immediately after the estimation sample (the current workfile sample). The forecast endpoint is given by either:

forclen = *int*	Number of periods to forecast.
forc = "*date*"	Specify the date of the forecast end point.

If omitted, the end point will be the end of the workfile range.

Examples

```
elecdmd.prophet(f=actual) elec_f elec_l elec_u
```

This will execute the Facebook Prophet routine on the ELECDMD series, inserting the actual values for out-of-forecast sample observations. The forecasted values will be stored in the new series ELEC_F, with the lower and upper bounds to the forecast being stored in ELEC_L and ELEC_U respectively.

Cross-references

See "Facebook Prophet Forecasting" on page 589 of *User's Guide I*.

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, if desired, an *output_spec* containing 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

outsmpl = *smpl_spec*	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.
permute	Draw from rows without replacement. Default is to draw with replacement.
weight = *series_name*	Name of series to be used as weights. The weight series must have non-missing, non-negative values in the current workfile sample. If no weights are specified, observations will be drawn with equal probability weights.
block = *integer* (default = *1)*	Block length for each draw. Must be a positive integer. The default block length is 1.
withna *(default)*	[Draw / Do not draw] from all rows in the current sample, including those with NAs.
dropna	Do not draw from rows that contain missing values in the current workfile sample.
fixna	Excludes NAs from draws but copies rows containing missing values to the output series.
prompt	Force the dialog to appear from within a program.

- You may not use this proc with an auto-series unless you provide an *output_spec*. For example, resampling from X(–1) or LOG(X) without providing explicit output names will produce an error since we will attempt to append a suffix to the original name, producing an invalid object name.
- 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 502 of *User's Guide I* for a discussion of the resampling procedure. For additional discussion of wildcards, see Appendix A. "Wildcards," on page 1227 of *User's Guide II*.

See also `@resample` (p. 1071) and `@permute` (p. 1037) in the *Command and Programming Reference* 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, Census X12, and Census X-13ARIMA-SEATS seasonal adjustment. For details, see `Series::x11` (p. 885), `Series::x12` (p. 885), and `Series::x13` (p. 890).

Syntax

series_name.**seas***(options) name_adjust [name_fac]*

Options

m	Multiplicative (ratio to moving average) method.
a	Additive (difference from moving average) method.
prompt	Force the dialog to appear from within a program.

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 507 of *User's Guide I* for a discussion of seasonal adjustment methods.

See also `seasplot` (p. 1322), `Series::x11` (p. 885), `Series::x12` (p. 885), `Series::x13` (p. 890), and `Series::tramoseats` (p. 859).

seasuroot

Series Views

Carries out seasonal unit root tests on a series.

Computes Hylleberg, Engle, Granger, and Yoo (HEGY, 1990) test, Smith and Taylor (1998) likelihood HEGY test, Canova and Hansen (CH, 1995), and Taylor (2005) HEGY variance ratio tests.

Syntax

series_name.**seasuroot***(options)*

Options

type = *arg* (default = "hegy")	Test type: HEGY ("hegy"), HEGY likelihood ("lklhd"), Canova-Hansen ("ch"), HEGY variance ratio ("hegyvr").
out = *arg*	Output matrix name (optional).

Basic Specification

periods = *arg*	Periodicity: "2", "4", "5", "6", or "12". The default is workfile frequency specific: semi-annual (2), quarterly (4), bimonthly (6), monthly (12), all others (4).
rstrct = *arg*	Restrictions to test (if "test = ch"): *arg* = space separated list of integers, "all" (all frequencies). For example, if the number of periods is 4, the restriction "0 1 2" corresponds to the 0, harmonic pair $(\pi/2, 3\pi/4)$, and π frequencies.
dtr = *arg*	Non-seasonal deterministic variables to include in estimation: "none" (no exogenous variables), "const" (constant), "trend" (constant and linear trend). The default is "const" for "type = hegyvr" and "none" for all others.
sdtr = *arg*	Seasonal deterministic variables to include in estimation: "none" (no seasonal exogenous), "dum" (seasonal dummies), "const" (constant), "trend" (constant and linear trend).
lagdep	Include lag of dependent variable in computations for Canova-Hansen test ("type = ch").

Lag Difference Options

Specifies the number of lag difference terms to be included in the test equation. Applicable in "Summary", LLC, Breitung, IPS, and Fisher-ADF tests. The default setting is to perform automatic lag selection using the Schwarz criteria ("lagmethod = sic").

lagmethod = *arg* *(default* = "sic"*)*	Method for selecting lag lengths (number of first difference terms) to be included in the Dickey-Fuller test regressions: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn), "tstat" (Ng-Perron first backward significant *t*-statistic).

lag = *arg*	Specified lag length (number of first difference terms) to be included in the regression: *integer* (user-specified common lag length), *vector_name* (user-specific individual lag length, one row per cross-section).
maxlag = *arg*	Maximum lag length to consider when performing automatic lag length selection: *integer* (common maximum lag length), or *vector_name* (individual maximum lag length, one row per cross-section). The default setting produces individual maximum lags of, $$default = \text{int}(\min(12,\ T_i/3) \cdot (\ T_i/100)^{1/4})$$ where T_i is the length of the cross-section.
lagpval = *arg* (*default* = 0.1)	Probability value for use in the *t*-statistic automatic lag selection method (when "lagmethod = tstat").

General options

prompt	Force the dialog to appear from within a program.
p	Print output from the test.

Examples

```
houst.seasuroot(dtr=trend, sdtr=dum)
```

runs a traditional HEGY test using seasonal dummies and trend and automatic AIC lag selection.

```
houst.seasuroot(type=lklhd, dtr=trend, sdtr=trend, info=sic)
```

runs likelihood HEGY test using spectral intercepts and trends and automatic SIC lag selection.

```
airline.seasuroot(type=ch, sdtr=const, rstrct=0 6, lagdep)
```

does Canova-Hansen test for frequency 0 and PI, and add lag of dependent variable as regressor.

Cross-references

See "Seasonal Unit Root Testing" on page 801 of *User's Guide II* for a discussion of seasonal unit root testing.

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 195 of *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 Chapter 18. "Function Reference," on page 679 of the *Command and Programming Reference*.

See "Numeric Expressions" on page 195 of *User's Guide I* for a discussion of rules for forming EViews expressions.

setattr	Series Procs

Set the object attribute.

Syntax

series_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the “revised” attribute in the object A to the string “never”, and extracts the attribute into the string object S.

Cross-references

See “Adding Custom Attributes in the Label View” on page 123 and “Adding Your Own Label Attributes” on page 70 of *User’s Guide I*.

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. 411) or `fetch` (p. 449) in the *Command and Programming Reference* or `Link::linkto` (p. 528)).

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	*High to low conversion methods removing NAs*: “a” (average of the nonmissing observations), “s” (sum of the nonmissing observations), “f” (first nonmissing observation), “l” (last nonmissing observation), “x” (maximum nonmissing observation), “m” (minimum nonmissing observation). *High to low conversion methods propagating NAs*: “an” or “na” (average, propagating missings), “sn” or “ns” (sum, propagating missings), “fn” or “nf” (first, propagating missings), “ln” or “nl” (last, propagating missings), “xn” or “nx” (maximum, propagating missings), “mn” or “nm” (minimum, propagating missings).

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 177 of *User’s Guide I* for a discussion of frequency conversion and the treatment of missing values.

See also `copy` (p. 411) and `fetch` (p. 449) of the *Command and Programming Reference*, and `Link::linkto` (p. 528).

setfillcolor	Series Procs

Set the fill (background) color used in the series spreadsheet using values in the spreadsheet or in a different series.

Syntax

series_name.**setfillcolor**(**t** = *type*) *fill_color_args*

where:

type = *arg*	Type of fill coloring for spreadsheet cells: “single” (single color), “posneg” (positive-negative single threshold), “range” (single range coloring), “hilo” (high-low-median), “custom” (custom coloring).

General Arguments

To specify the series or expression whose values will determine the background color:

- **mapser**(*spec*)

where *spec* is a series name or expression.

To specify the minimum and maximum values where the coloring begins and ends:

- **min**(*color_arg*)
- **max**(*color_arg*)

To set the missing value (NA) background color:

- **naclr**(*color_arg*)

where *color_arg* is described below in “Color definitions” on page 839. If omitted, the color defaults to “white”.

Type-specific Arguments

There are optional type-specific arguments that correspond to each of the type choices:

Single color

To set the single background color:

clr(*color_arg*)

where *color_arg* is described below in “Color definitions” on page 839. If omitted, the color defaults to “white”.

Positive-negative single threshold

You may set the color for both the non-negative (**posclr**) and the negative (**negclr**) values

posclr(*color_arg*)
negclr(*color_arg*)

where *color_arg* is described below in “Color definitions” on page 839. If omitted, the non-negative color defaults to “white” and the negative color defaults to light-red.

Single range

To specify the range, you must specify the range endpoints:

range(*lower_val, upper_val*[, *range_def)*

where *range_def* specifies the range endpoints:

cright	closed on the right only
cboth	closed on both sides
cleft	closed on the left only
oboth	open on both sides

By default, the range will be open on the lower and closed on the upper threshold limits.

You should provide a color specification for the inside range color (**inclr**) and outside range color (**outclr**):

inclr(*color_arg*)
outclr(*color_arg*)

where *color_arg* is described below in "Color definitions" on page 839. If omitted, the interior color defaults to light-red, and the exterior defaults to white.

High-Low-Median

When "type = hilo" you may specify the high, low, and median coloring values:

highclr(*color_arg*)
lowclr(*color_arg*)
medianclr(*color_arg*)

where *color_arg* is described below in "Color definitions" on page 839. If omitted, the colors default to light-red.

Custom

When "type = custom" you may specify custom coloring options.

You may optionally set a base background color, and then add one or more custom threshold or range color specifications. Multiple threshold and range specifications will layer, with the first applied first, followed by the second, and so on.

Custom Base Color

To set the base color (optional):

clr(*color_arg*)

as described below in "Color definitions" on page 839. If omitted, the color defaults to "white".

Custom Threshold

To add a threshold specification:

thresh(**limit***(threshold_value, threshold_spec)*, **lowclr***(below_arg),* **highclr***(above_arg),*
threshold_name])

where *threshold_spec* is one of

cright	closed on the right
cleft	closed on the left

and the *below_arg* and *above_arg* are one of

color_arg	solid color specification
@grad(*color_arg*)	gradient using color specification
@trans	transparent

and *color_arg* are as described below in "Color definitions" on page 839. If omitted, the color defaults to "white".

The optional *threshold_name* argument may be used to attach a name to the corresponding definition.

Custom Range

To add a range specification:

range(**limit***(low_value, high_value, range_spec),* **inclr***(inside_arg),* **outclr***(outside_arg)[, range_name]*)

where *range_spec* is one of

cright	closed on the right only
cboth	closed on both sides
cleft	closed on the left only
oboth	open on both sides

inside_arg is one of

color_arg	solid color specification
@grad(*color_arg1, color_arg2*)	gradient using color specification, where *color_arg1 and color_arg2* are the low and high colors, respectively.
@trans	transparent

outside_arg is one of

color_arg	solid color specification
@grad(*color_arg*)	gradient using color specification
@trans	transparent

color_arg1 and *color_arg2* are as described below in "Color definitions" on page 839.

The optional *range_name* argument may be used to attach a name to the corresponding definition.

Color definitions

color_arg specifies the color to be employed in the arguments above. The color may be specified using predefined color names, by specifying the individual red-green-blue (RGB) components using the special "@RGB" function, or by specifying the individual red-green-blue (RGB) components in hexadecimal using the special "@HEX" function.

The predefined colors are given by the keywords (with their RGB and HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
green	@rgb(0, 128, 0)	@hex(ffa8a8)
black	@rgb(0, 0, 0)	@hex(008000)
white	@rgb(255, 255, 255)	@hex(000000)
purple	@rgb(128, 0, 128)	@hex(ffffff)
orange	@rgb(255, 128, 0)	@hex(800080)
yellow	@rgb(255, 255, 0)	@hex(ff8000)
gray	@rgb(128, 128, 128)	@hex(ffff00)
ltgray	@rgb(192, 192, 192)	@hex(808080)

Examples

To set a gray background color for all cells in the spreadsheet, you may use:

```
myser.setfillcolor(type=single) clr(gray)
```

To set a background color for negative values, you may use

```
myser.setfillcolor(type=posneg) mapser(ser1)
```

which sets the background sheet fill color to white for non-negative values and light red for negative values of SER1.

Similarly,

```
myser.setfillcolor(type=posneg) mapser(ser1) posclr(@rgb(10, 20,
    30)) negclr(purple)
```

sets the background sheet fill color to @rgb(10, 20, 30) for non-negative values and purple for negative values of SER1.

Range coloring may be specified using the "type = range" option. The command

```
myser.setfillcolor(type=range) mapser(ser1) clr(ltgray) range(10,
    20, cleft) inclr(@rgb(128, 0, 128)) outclr(ltred) naclr(green)
```

sets the background fill to @rgb(128, 0, 128) for values between 10 and 20, light-red to values outside of the range 10 to 20, and green, for missing values.

Custom coloring allows you to construct more complex background filling:

```
myser.setfillcolor(type=custom) mapser(ser1) clr(@rgb(10, 0, 0))
    range(limit(-10, 10, oboth), inclr(green), outclr(white)))
    thresh(limit(-1, oleft), highclr(grey), lowclr(@trans))
```

Cross-references

See "Value-Based Text and Fill Coloring" on page 186 of *User's Guide I*.

See also `Series::settextcolor` (p. 844).

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:

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

To specify a format that groups digits into thousands using a comma separator, place a "t" after the format character. For example, to obtain a fixed number of decimal places with commas used to separate thousands, use "ft*[.precision]*".

To use the period character to separate thousands and commas to denote decimal places, use ".." (two periods) when specifying the precision. For example, 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)"), 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 106 of the *Command and Programming Reference*). 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 period 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. 849), `Series::setindent` (p. 843) and `Series::setjust` (p. 844) 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 indentation setttings are taken from the Global Defaults for spreadsheet views ("Spreadsheet Data Display" on page 1019 of *User's Guide I*) at the time the spreadsheet was 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. 849) and `Series::setjust` (p. 844) for details on setting spreadsheet widths and justification.

setjust	Series Procs

Set the horizontal justification for all cells in the spreadsheet view of the series.

Syntax

series_name.**setjust** *format_arg*

where *format_arg* may be set to left, center, right, or auto (strings are left-justified and numbers are right-justified). Default display settings can be set in General Options; see "Spreadsheet Data Display" on page 1019 of *User's Guide I*.

Examples

```
ser1.setjust left
```

left-justifies the cells in the spreadsheet view of the series SER1.

Cross-references

See `Series::setwidth` (p. 849) and `Series::setindent` (p. 843) for details on setting spreadsheet widths and indentation.

settextcolor	Series Procs

Set the text color used in the series spreadsheet using values in the spreadsheet or in a different series.

Syntax

series_name.**settextcolor**(**t** = *type*) *text_color_args*

where:

type = *arg*	Type of fill coloring for spreadsheet cells: "single" (single color), "posneg" (positive-negative single threshold), "range" (single range coloring), "hilo" (high-low-median), "custom" (custom coloring).

General Arguments

To specify the series or expression whose values will determine the background color:

- **mapser**(*spec*)

where *spec* is a series name or expression.

To specify the minimum and maximum values where the coloring begins and ends:

- **min**(*color_arg*)
- **max**(*color_arg*)

To set the missing value (NA) background color:

- **naclr**(*color_arg*)

where *color_arg* is described below in "Color definitions" on page 847. If omitted, the color defaults to "black".

Type-specific Arguments

There are optional type-specific arguments that correspond to each of the type choices:

Single color

To set the single text color:

clr(*color_arg*)

where *color_arg* is described below in "Color definitions" on page 847. If omitted, the color defaults to "black".

Positive-negative single threshold

You may set the color for both the non-negative (**posclr**) and the negative (**negclr**) values

posclr(*color_arg*)
negclr(*color_arg*)

where *color_arg* is described below in "Color definitions" on page 847. If omitted, the non-negative color defaults to "black" and the negative color defaults to "red".

Single range

To specify the range, you must specify the range endpoints:

range(*lower_val, upper_val[, range_def)*

where *range_def* specifies the range endpoints:

cright	closed on the right only
cboth	closed on both sides
cleft	closed on the left only
oboth	open on both sides

By default, the range will be open on the lower and closed on the upper threshold limits.

You should provide a color specification for the inside range color (**inclr**) and outside range color (**outclr**):

inclr(*color_arg*)

outclr(*color_arg*)

where *color_arg* is described below in "Color definitions" on page 847. If omitted, the interior color defaults to light-red, and the exterior defaults to white.

High-Low-Median

When "type = hilo" you may specify the high, low, and median coloring values:

highclr(*color_arg*)
lowclr(*color_arg*)
medianclr(*color_arg*)

where *color_arg* is described below in "Color definitions" on page 839. If omitted, the colors default to light-red.

Custom

When "type = custom" you may specify custom coloring options.

You may optionally set a base text color, and then add one or more custom threshold or range color specifications. Multiple threshold and range specifications will layer, with the first applied first, followed by the second, and so on.

Custom Base Color

To set the base color (optional):

clr(*color_arg*)

as described below in "Color definitions" on page 847. If omitted, the color defaults to "white".

Custom Threshold

To add a threshold specification:

thresh(**limit**(*threshold_value, threshold_spec)*, **lowclr**(*below_arg),* **highclr**(*above_arg),*
threshold_name])

where *threshold_spec* is one of

cright	closed on the right
cleft	closed on the left

and the *below_arg* and *above_arg* are one of

color_arg	solid color specification
@grad(*color_arg*)	gradient using color specification
@trans	transparent

and *color_arg* are as described below in "Color definitions" on page 847. If omitted, the color defaults to "white".

The optional *threshold_name* argument may be used to attach a name to the corresponding definition.

Custom Range

To add a range specification:

range(**limit***(low_value, high_value, range_spec)*, **inclr***(inside_arg)*, **outclr***(outside_arg)[, range_name]*)

where *range_spec* is one of

cright	closed on the right only
cboth	closed on both sides
cleft	closed on the left only
oboth	open on both sides

inside_arg is one of

color_arg	solid color specification
@grad(*color_arg1*, *color_arg2*)	gradient using color specification, where *color_arg1 and color_arg2* are the low and high colors, respectively.
@trans	transparent

outside_arg is one of

color_arg	solid color specification
@grad(*color_arg*)	gradient using color specification
@trans	transparent

color_arg1 and *color_arg2* are as described below in "Color definitions" on page 847.

The optional *range_name* argument may be used to attach a name to the corresponding definition.

Color definitions

color_arg specifies the color to be employed in the arguments above. The color may be specified using predefined color names, by specifying the individual red-green-blue (RGB) components using the special "@RGB" function, or by specifying the individual red-green-blue (RGB) components in hexadecimal using the special "@HEX" function.

The predefined colors are given by the keywords (with their RGB and HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
ltred	@rgb(255, 168, 168)	@hex(ffa8a8)
green	@rgb(0, 128, 0)	@hex(008000)
black	@rgb(0, 0, 0)	@hex(000000)
white	@rgb(255, 255, 255)	@hex(ffffff)
purple	@rgb(128, 0, 128)	@hex(800080)
orange	@rgb(255, 128, 0)	@hex(ff8000)
yellow	@rgb(255, 255, 0)	@hex(ffff00)
gray	@rgb(128, 128, 128)	@hex(808080)
ltgray	@rgb(192, 192, 192)	@hex(c0c0c0)

Examples

To set a gray text color for all cells in the spreadsheet, you may use:

```
myser.settextcolor(type=single) clr(gray)
```

To set a text color for negative values, you may use

```
myser.settextcolor(type=posneg) mapser(ser1)
```

which sets the text color to black for non-negative values and red for negative values of SER1.

Similarly,

```
myser.settextcolor(type=posneg) mapser(ser1) posclr(@rgb(10, 20,
    30)) negclr(purple)
```

sets the text color to @rgb(10, 20, 30) for non-negative values and purple for negative values of SER1.

Range coloring may be specified using the "type = range" option. The command

```
myser.settextcolor(type=range) mapser(ser1) clr(ltgray) range(10,
    20, cleft) inclr(@rgb(128, 0, 128)) outclr(ltred) naclr(green)
```

sets the text to @rgb(128, 0, 128) for values between 10 and 20, light-red to values outside of the range 10 to 20, and green, for missing values.

Custom coloring allows you to construct more complex text coloring:

```
myser.settextcolor(type=custom) mapser(ser1) clr(@rgb(10, 0, 0))
    range(limit(-10, 10, oboth), inclr(green), outclr(white)))
    thresh(limit(-1, oleft), highclr(grey), lowclr(@trans))
```

Cross-references

See "Value-Based Text and Fill Coloring" on page 186 of *User's Guide I*.

See also `Series::setfillcolor` (p. 835).

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. 843) and `Series::setjust` (p. 844) for details on setting spreadsheet indentation and justification.

sheet	Series Views

Spreadsheet view of a series object.

Syntax

series_name.**sheet***(options)*

Options

w	Wide. In a panel this will switch to the unstacked form of the panel (dates along the side, cross-sections along the top).
t	Transpose.
a	All observations (ignore sample)

nl	Do not display labels.
tform = *arg* (*default* = "default"	Display transformed data: value mapped or date formated (when applicable) or raw data, otherwise ("default"), raw data ("level"), one period difference ("dif" or "d"), annual difference ("dify" or "dy"), one period percentage change ("pch" or "pc"), annualized one period percentage change ("pcha" or "pca"), annual percentage change ("pchy" or "pcy"), natural logarithm ("log"), one period difference of logged values ("dlog").
p	Print the spreadsheet view.

Examples

```
ser1.sheet(p)
```

displays and prints the default spreadsheet view of series SER1.

```
ser1.sheet(t, tform=log)
```

displays log values of SER1 in the current sample in a wide spreadsheet.

```
ser1.sheet(nl, tform=diff)
```

shows differenced values of the series without labels.

```
ser1.sheet(a, tform=pc)
```

displays the one period percent changes for all observations in the workfile.

Cross-references

See Chapter 5. "Basic Data Handling," on page 129 of *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 *x* between zero and one for the mean parameter.
d*[,x]*	Double exponential smoothing for series with a trend. You may optionally specify a number *x* between zero and one for the mean parameter.
n*[,x,y] (default)*	Holt-Winters without seasonal component. You may optionally specify numbers *x* and *y* between zero and one for the mean and trend parameters, respectively.
a*[,x,y,z]*	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.
m*[,x,y,z]*	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:

prompt	Force the dialog to appear from within a program.
p	Print a table of forecast statistics.

If you wish to set only some of the damping parameters and let EViews estimate the other parameters, enter the letter "e" where you wish the parameter to be estimated.

If the number of seasons is different from the frequency of the workfile (an unusual case that arises primarily if you are using an undated workfile for data that are not monthly or quarterly), you should enter the number of seasons after the smoothed series name. This optional input will have no effect on forecasts without seasonal components.

Examples

```
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 599 of *User's Guide I* for a discussion of exponential smoothing methods. See also `Series::ets` (p. 787).

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 642 of *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_list*

You should follow the series name with a period, the `statby` keyword, and a name (or a list of names) for the series or groups 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

sum	Display sums.
med	Display medians.
max	Display maxima.
min	Display minima.
quant = *arg* *(default* = .5*)*	Display quantile with value given by the argument.
q = *arg* *(default* = "r"*)*	Compute quantiles using the specified definition: "b" (Blom), "r" (Rankit-Cleveland), "o" (Ordinary), "t" (Tukey), "v" (van der Waerden), "g" (Gumbel).
skew	Display skewness.
kurt	Display kurtosis.
na	Display counts of NAs.

nomean	Do not display means.
nostd	Do not display standard deviations.
nocount	Do not display counts.

Options to control layout

l	Display in list mode (for more than one classifier).
nor	Do not display row margin statistics.
noc	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* = 1000*)*	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.
nolimit	Remove prompt warning for continuing when the total number of cells is very large.

Other options

prompt	Force the dialog to appear from within a program.
p	Print the descriptive statistics table.

Examples

```
wage.statby(max, min) sex race
```

displays the mean, standard deviation, maximum, and minimum of the series WAGE by (possibly binned) values of SEX and RACE.

```
group g1 sex race state
wage.statby(max, min) g1
```

shows the mean, standard deviation, maximum, and minimum of the series WAGE by (possibly binned) values of SEX, race, and STATE.

Cross-references

See "By-Group Statistics" on page 703 of the *Command and Programming Reference* for a list of functions to compute by-group statistics. See also "Stats by Classification" on page 453 of *User's Guide I* for discussion.

See also `Vector::statby` (p. 1258) and `boxplot` (p. 1279).

stats	Series Views

Descriptive statistics.

Computes and displays a table of means, medians, maximum and minimum values, standard deviations, and other descriptive statistics of the 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 450 of *User's Guide I* for a discussion of the descriptive statistics views of series.

See also `boxplot` (p. 1279) and `Series::hist` (p. 798).

stl	Series Procs

Seasonally adjust series using the STL decomposition method.

Unlike other seasonal adjustment methods used by EViews, this procedure works on any time frequency.

Syntax:

series.**stl***(options) seas_name [trend_name]*

You should follow the `stl` keyword with a name for the seasonally adjusted series. Optionally, you may also provide a name for the output trend series.

Options

periodicity = *arg*	Specify the periodicity. Use "w" to expand weekly data to 53 weeks and "d" to expand daily data to 366 (in a 7 day week workfile) or 261 (in a 5 day week workfile) days. Default is the number of periods per year (expanded for weekly and daily).
sp = *integer*	Specify the seasonal polynomial degree. Default is 0.
tp = *integer*	Specify the trend polynomial degree. Default is 1.
fp = *integer*	Specify the filter polynomial degree. Default is 1.
sl = *integer*	Specify the length of the seasonal smoothing window (odd integers only). Default is 35.
tl = *integer*	Specify the length of the trend smoothing window (odd integers only). Default is based upon the seasonal smoothing window length.
fl = *integer*	Specify the length of the filter smoothing window (odd integers only). Default is based upon the data frequency.
inits = *integer*	Specify number of inner iterations. Default is 5.
outits = *integer*	Specify the number of outer iterations. Default is 15.
estsmpl = *arg*	Set the estimation sample.
forclen = *integer*	Set the length of the forecast.
seasdiagnostic	Display seasonality diagnostics graph.

Examples

```
co2.stl co2_sa c02_trend
```

performs STL decomposition on the series C02, saving the adjusted data in the series C02_SA and the trend in C02_TREND.

```
show co2.stl(sl=20, outits=20, seasdiagnostic)
```

performs the same decomposition, but with a seasonal smoothing window of 20, using 20 iterations of the outer loop, and displays the seasonal diagnostics graphs.

testby	Series Views

Test equality of the mean, median, or variance of a series across categories defined by the row values of series or groups.

Syntax

series_name.**testby***(options) arg1 [arg2 arg2 …]*

Follow the `testby` keyword by a list of the names of series or groups to use as classifiers.

By default, `testby` will test for equality of means, but you may specify instead tests of medians or variances as an option, choose whether to use balanced or unbalanced samples, and control binning.

Options

mean *(default)*	Test equality of means.
med	Test equality of medians.
var	Test equality of variances.
dropna *(default)*, keepna	[Drop /Keep] NAs as a classification category.
v = *integer* *(default* = 1000*)*	Bin categories if classification series take 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 on the basis of average cell count.
b = *integer* *(default* = 5*)*	Set maximum number of binned categories.

nolimit	Remove prompt warning for continuing when the total number of cells is very large.
prompt	Force the dialog to appear from within a program.
p	Print the test results.

Examples

```
wage.testby(a=10) state
```

tests equality of means of WAGE across groups classified by state, with binning of the average cell cound is less than 10.

```
wage.testby(med, nov) stores
```

tests equality of medians of WAGE across groups classified by STORES, with no automatic binning on the basis of the number of unique STORES values.

Cross-references

See "Equality Tests by Classification" on page 459 of *User's Guide I* for a discussion of equality tests.

See also `Group::testbtw` (p. 512) and `Series::teststat` (p. 858).

teststat

Series Views

Test simple hypotheses of whether the mean, median, or variance of a series are equal to specific values.

Syntax

series_name.**teststat***(options)*

Specify the type of test and the value under the null hypothesis as an option. You must specify at least one hypothesis.

For tests of means, you may either estimate the variance or specify the variance as an option.

Options

mean = *number*	Test the null hypothesis that the mean equals the specified number.
med = *number*	Test the null hypothesis that the median equals the specified number.
var = *number*	Test the null hypothesis that the variance equals the specified number. The number must be positive.

std = *number*	Test equality of mean conditional on the specified standard deviation. The standard deviation must be positive.
prompt	Force the dialog to appear from within a program.
p	Print the test results.

Examples

```
smpl @all
lwage.teststat(mean=7)
```

tests the null hypothesis that the mean of LWAGE is equal to 7, using an estimated standard deviation.

```
lwage.teststat(mean=7, std=2)
```

tests the null that the mean is 7, using an estimated standard deviation, and also assuming that the standard deviation is known to be 2.

```
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 450 of *User's Guide I* for a discussion of simple hypothesis tests.

See also `Group::testbtw` (p. 512) and `Series::testby` (p. 857).

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 $\underline{n}$ observations and can adjust up to 600 observations where:

$$n = \begin{cases} 36 & \text{for monthly data} \\ \max\{12, 4s\} & \text{for other seasonal data} \end{cases} \tag{1.4}$$

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*	Specify series to save in workfile: you must use one or more from the following key word list: "hat" (forecasts of original series), "lin" (linearized series from Tramo), "pol" (interpolated series from Tramo), "sa" (seasonally adjusted series from Seats), "trd" (final trend component from Seats), "ir" (final irregular component from Seats), "sf" (final seasonal factor from Seats), "cyc" (final cyclical component from Seats). To save more than one series, separate the list of key words with a space. *Do not use commas* within the list of save series. The special key word "*save* = *" will save all series in the key word list. The five key words "*sa*", "*trd*", "*ir*", "*sf*", "*cyc*" will be ignored if "runtype = t".
opt = *arg*	A space delimited list of input namelist. *Do not use commas within the list*. The syntax for the input namelist is explained in the.PDF documentation file. See also "Notes" on page 860.
reg = *arg*	A space delimited list for one line of reg namelist. *Do not use commas within the list.* This option must be used in pairs, either with another "reg = " option or "regname = " option. The reg namelist is available only for Tramo and its syntax is explained in the PDF documentation file. See also "Notes" on page 860.
regname = *arg*	Name of a series or group in the current workfile that contains the exogenous regressors specified in the previous "*reg* = " option. See "Notes" on page 860.
prompt	Force the dialog to appear from within a program.
p	Print the results of the Tramo/Seats procedure.

Notes

The command line interface to Tramo/Seats does very little error checking of the command syntax. EViews simply passes on the options you provide "as is" to Tramo/Seats. If the syntax contains an error, you will most likely 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 546 of *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: "itrad = ".
  7. outlier detection: "iatip = " and "aio = ".
- The command option input string list must be space delimited. *Do not use commas*. Options containing an equals sign should not contain any spaces around the equals; the space will be interpreted as a delimiter by Tramo/Seats.
- If you set "rtype = ts", you are responsible for supplying either "seats = 1" or "seats = 2" in the "opt = " option list. EViews will issue the error message "Seats.itr not found" if the option is omitted. Note that the dialog option **Run Seats after Tramo** sets "seats = 2".
- Each "reg = " or "regname = " option is passed to the input file as a separate line in the order that they appear in the option argument list. Note that these options must come in pairs. A "reg = " option must be followed by another "reg = " option that specifies the outlier or intervention series or by a "regname = " option that provides the name for an exogenous series or group in the current workfile. See the sample programs in the "./Example Files" directory.
- If you specify exogenous regressors with the "reg = " option, you must set the appropriate "ireg = " option (for the total number of exogenous series) in the "opt = " list.
- To use the "regname = " option, the preceding "reg = " list must contain the "user = -1" option and the appropriate "ilong = " option. Do *not* use "user = 1" since EViews will always write data in a separate external file. The "ilong = " option must be at least the number of observations in the current workfile sample *plus* the number of forecasts. The exogenous series should not contain any missing values in this range. *Note that Tramo may increase the forecast horizon, in which case the exogenous series is extended by appending zeros at the end.*

Examples

```
freeze(tab1) 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 543 of *User's Guide I* for discussion. See also the Tramo/Seats documentation that accompanied your EViews distribution.

See `Series::seas` (p. 828) and `Series::x12` (p. 885).

trendtests	Series Views

Perform tests on the existence of a trend in the series.

Calculate the linear t-test, squared F-test, Mann-Kendall test, seasonal Mann-Kendall test, Cox-Stuart test and the Wang, Akritas and Van Keilegom (WAVK) test for time series trends.

Syntax

series_name.**trendtests***(options)*

Options

noboot	Do not calculate bootstrapped p-values.
simple	Use the simple bootstrap. By default, the sieve bootstrap is used.
mle	Calculate the bootstrap AR estimates using MLE. Default is to use the HVK estimates.
iter = *arg*	Number of bootstrap simulations.
seed = *int*	Set the random number generator seed.
rng = *arg*	Set random number generator type. Available types are: 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").
out = *name*	Specify the name of the matrix containing the test statistics and p-values.

Examples

```
gdpc1.trendtests(iter=19999, out=ttstats)
```

Performs a trend test on the GDPC1 series, performing 19,999 bootstraps using HVK estimates in a sieve bootstrap, and saving the test statistics and p-values into a matrix object named TTSTATS.

Cross-references

See "Trend Tests" on page 761 of *User's Guide II* for discussion

uroot	Series Views

Carries out unit root tests on a series or panel structured series.

For ordinary series, computes conventional 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.

For series in a panel structured workfile, computes Levin, Lin and Chu (LLC), Breitung, Im, Pesaran, and Shin (IPS), Fisher - ADF, Fisher - PP, or Hadri panel unit root tests on levels, first, or second differences of the data.

Syntax

series_name.**uroot***(options)*

There are different options for conventional tests on an ordinary series and panel tests for series in panel structured workfiles.

Options for Conventional Unit Root Tests

Basic Specification

You should specify the exogenous variables and order of dependent variable differencing in the test equation using the following options:

exog = *arg* *(default* = "const"*)*	Specification of exogenous trend variables in the test equation: "const" "trend" (include a constant and a linear time trend), "none" (do not include any exogenous regressors).
dif = *integer* *(default* = 0*)*	Order of differencing of the series prior to running the test. Valid values are {0, 1, 2}.

You should specify the test type using one of the following keywords:

adf *(default)*	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.

Note that for backward compatibility, EViews supports older forms of the exogenous specification:

const, c *(default)*	Include a constant in the test equation.
trend, t	Include a constant and a linear time trend in the test equation.
none, n	Do not include a constant or time trend (only available for the ADF and PP tests).

For future compatibility we recommend that you use the "exog = " format.

Spectral Estimation Option

In addition, PP, KPSS, ERS, and NP tests all require the estimation of the long-run variance (frequency zero spectrum). You may specify the method using the "hac = " option. The default setting depends on the selected test.

hac = *arg* *(default* = varies*)*	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). *The default settings are test specific* ("bt" for PP and KPSS tests, "ar" for ERS, "argls" for NP).

Lag Difference Options

Applicable to ADF and DFGLS tests, and for PP, KPSS, ERS, and NP tests that use a AR spectral density estimator ("hac = ar", "hac = ardt", or "hac = argls"). The default lag selection method is based on a comparison of Schwarz criterion values. You may specify a fixed lag using the "lag = " option.

lagmethod = *arg* *(default* = "sic"*)*	Method for selecting lag length (number of first difference terms) to be included in the Dickey-Fuller test regression or number of lags in the AR spectral density estimator: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn), "msaic" (Modified Akaike), "msic" (Modified Schwarz), "mhqc" (Modified Hannan-Quinn), "tstat" (Ng-Perron first backward significant *t*-statistic).
lag = *integer*	Use-specified fixed lag.
maxlag = *integer*	Maximum lag length to consider when performing automatic lag length selection. $$default = \text{int}((12\,T/100)^{0.25})$$
lagpval = *arg* *(default* = 0.1*)*	Probability value for use in the *t*-statistic automatic lag selection method ("lagmethod = tstat").

Kernel Option

Applicable to PP, KPSS, ERS, and NP tests when using kernel estimators of the frequency zero spectrum (where "hac = bt", "hac = pz", or "hac = qs")

band = *arg*, b = *arg* *(default* = "nw"*)*	Method of selecting the bandwidth: "nw" (Newey-West automatic variable bandwidth selection), "a" (Andrews automatic selection), *number* (user specified bandwidth).

General Options

prompt	Force the dialog to appear from within a program.
p	Print output from the test.

Options for Panel Unit Root Tests

Basic Specification

You should specify the exogenous variables, order of dependent variable differencing, and sample handling, in the test equation using the following options:

exog = *arg* *(default* = "const"*)*	Specification of exogenous trend variables in the test equation: "const" (constant),"trend" (include a constant and a linear time trend), "none" (do not include exogenous regressors).
dif = *integer* *(default* = 0*)*	Order of differencing of the series prior to running the test. Valid values are {0, 1, 2}.
balance	Use balanced (across cross-sections or series) data when performing test.

You may use one of the following keywords to specify the test:

sum *(default)*	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.

For backward compatibility, EViews supports older forms of the exogenous specification:

const, c *(default)*	Include a constant in the test equation.
trend, t	Include a constant and a linear time trend in the test equation.
none, n	Do not include a constant or time trend (only available for the ADF and PP tests).

For future compatibility we recommend that you use the "exog = " format.

Lag Difference Options

Specifies the number of lag difference terms to be included in the test equation. Applicable in "Summary", LLC, Breitung, IPS, and Fisher-ADF tests. The default setting is to perform automatic lag selection using the Schwarz criteria ("lagmethod = sic").

lagmethod = *arg* *(default* = "sic"*)*	Method for selecting lag lengths (number of first difference terms) to be included in the Dickey-Fuller test regressions: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn), "tstat" (Ng-Perron first backward significant *t*-statistic).
lag = *arg*	Specified lag length (number of first difference terms) to be included in the regression: *integer* (user-specified common lag length), *vector_name* (user-specific individual lag length, one row per cross-section).
maxlag = *arg*	Maximum lag length to consider when performing automatic lag length selection: *integer* (common maximum lag length), or *vector_name* (individual maximum lag length, one row per cross-section). The default setting produces individual maximum lags of, $$default = \text{int}(\min(12,\ T_i/3) \cdot (T_i/100)^{1/4})$$ where T_i is the length of the cross-section.
lagpval = *arg* *(default* = 0.1*)*	Probability value for use in the *t*-statistic automatic lag selection method (when "lagmethod = tstat").

Kernel Options

Specifies options for computing kernel estimates of the zero-frequency spectrum (long-run covariance). Applicable to "Summary", LLC, Fisher-PP, and Hadri tests.

hac = *arg* *(default* = "bt"*)*	Method of estimating the frequency zero spectrum: "bt" (Bartlett kernel), "pr" (Parzen kernel), "qs" (Quadratic Spectral kernel),
band = *arg*, b = *arg* *(default* = "nw")	Method of selecting the bandwidth: "nw" (Newey-West automatic variable bandwidth selection), "a" (Andrews automatic selection), *number* (user-specified common bandwidth), *vector_name* (user-specified individual bandwidths, one row for each cross-section).

General options

prompt	Force the dialog to appear from within a program.
p	Print output from the test.

Examples

The command:

```
gnp.uroot(adf,exog=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(dfgls,exog=trend,lagmethod=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,exog=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,lagmethod=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.uroot(llc,hac=pr,lagmethod=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 Testing" on page 773 of *User's Guide II* for discussion of standard unit root tests performed on a single series, and "Cross-sectionally Independent Panel Unit Root Testing" on page 811 and "Cross-sectionally Dependent Panel Unit Root Tests" on page 822 of *User's Guide II* for discussion of unit roots tests performed on panel structured workfiles, groups of series, or pooled data.

See also `Series::uroot2` (p. 868), `Group::uroot` (p. 512), `Group::uroot2` (p. 515), `Pool::uroot` (p. 689), `Pool::uroot2` (p. 692).

uroot2	Series Views

Compute dependent (second generation) panel unit root tests on a panel series.

Syntax

ser_name.**uroot2***(options)*

where *ser_name* is the name of a series in a panel structured workfile.

Options

General Options

type = *arg* *(default* = "panic"*)*	Type of unit root test: PANIC - Bai and Ng (2004) ("panic"), CIPS - Pesaran (2007) ("cips"). Note: (1) when performing PANIC testing, factor selection, MQ, ADF lag selection, VAR lag selection (possibly), long-run variance (possibly), and p-value simulation options are relevant. (2) when perform CIPS testing, ADF lag selection options are relevant.
exog = *arg* *(default* = "constant"*)*	Exogenous deterministic variables to include for each cross-section: "none" (no deterministic variables), "constant" (only a constant), "trend" (both a constant and trend).
prompt	Force the dialog to appear from within a program.
p	Print results.

ADF Lag Selection Options

adflagmethod = *arg* *(default* = "sic"*)*	Method for selecting lag length (number of first difference terms) to be included in the Dickey-Fuller test regression or number of lags in the AR spectral density estimator: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn), "msaic" (Modified Akaike), "msic" (Modified Schwarz), "mhqc" (Modified Hannan-Quinn), "tstat" (Ng-Perron first backward significant t-statistic).
adflag = *integer*	Use-specified fixed lag.
adfmaxlag = *integer*	Maximum lag length to consider when performing automatic lag length selection. Note: default is Schwert's rule: let $$L(k) = k(T/100)^{1/4}$$ for $k \in \{2, 4, 6, 8, 10, 12\}$ and let $T^* = \min(N, T)$; then the default maximum lag is given by $$L^* = \max_k\{L(k) \mid L(k) < T^*\}$$
adflagpval = *arg* *(default* = 0.1*)*	Probability value for use in the t-statistic automatic lag selection method ("lagmethod = tstat").

PANIC Number of Factor Selection Options

fsmethod = *arg* *(default* = "bn"*)*	Factor retention selection method: "bn" (Bai and Ng (2002)), "ah" (Ahn and Horenstein (2013)), "simple" (simple eigenvalue methods), "user" (user-specified value). Note the following: (1) If using simple methods, the minimum eigenvalue and cumulative proportions may be specified using "minigen = " and "cproport = ". (2) If setting "fsmethod = user" to provide a user-specified value, you must specify the value with "r = ".
r = *arg (default* = 1*)*	User-specified number of factors to retain (for use when "fsmethod = user").
mineigen = *arg* *(default* = 0*)*	Minimum eigenvalue to retain factor (when "fsmethod = simple").
cproport = *arg* *(default* = 1.0*)*	Cumulative proportion of eigenvalue total to attain (when "fsmethod = simple").
mfmethod = *arg*	Maximum number of factors used by selection methods: "schwert" (Schwert's rule, *default*), "ah" (Ahn and Horenstein's (2013) suggestion), "rootsize" ($\min(\sqrt{N}, \sqrt{T})$), "size" ($\min(N, T)$), "user" (user specified value). (1) For use with all factor retention methods apart from user-specified ("fsmethod = user"). (2) If setting "mfmethod = user", you may specify the maximum number of factors using "rmax = ". (3) Schwert's rule sets the maximum number of factors using the rule: let $$L(k) = k(T/100)^{1/4}$$ for $k \in \{2, 4, 6, 8, 10, 12\}$ and let $T^* = \min(N, T)$; then the default maximum lag is given by $$L^* = \max_k\{L(k) \mid L(k) < T^*\}$$
rmax = *arg* *(default* = all*)*	User-specified maximum number of factors to retain (for use when "mfmethod = user").

fsic = *arg (default* = avg*)*	Factor selection criterion when "fsmethod = bn": "icp1" (ICP1), "icp2" (ICP2), "icp3" (ICP3), "pcp1" (PCP1), "pcp2" (PCP1), "pcp3" (ICP3), "avg" (average of all criteria ICP1 through PCP3). Factor selection criterion when "fsmethod = ah": "er" (eigenvalue ratio), "gr" (growth ratio), "avg" (average of eigenvalue ratio and growth ratio). Factor selection when "fsmethod = simple": "min" (minimum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "max" (maximum of: minimum eigenvalue, cumulative eigenvalue proportion, and maximum number of factors), "avg" (average the optimal number of factors as specified by the min and max rule, then round to the nearest integer).
demeantime	Demeans observations across time prior to factor selection procedures.
sdizetime	Standardizes observations across time prior to factor selection procedures.
demeancross	Demeans observations across cross-sections prior to factor selection procedures.
sdizecross	Standardizes observations across cross-sections prior to factor selection procedures.

PANIC VAR Lag Selection Options

For use when computing a PANIC test with MQ_f statistic.

varlagmethod = *arg (default* = "sic"*)*	Method for selecting lag length (number of first difference terms) to be included in the test statistic VAR: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn), "msaic" (Modified Akaike), "msic" (Modified Schwarz), "mhqc" (Modified Hannan-Quinn), "tstat" (Ng-Perron first backward significant t-statistic).
varlag = *integer*	Use-specified fixed lag.
varmaxlag = *integer*	Maximum lag length to consider when performing automatic lag length selection. Note: default is Schwert's rule: let $$L(k) = k(T/100)^{1/4}$$ for $k \in \{2, 4, 6, 8, 10, 12\}$ and let $T^* = \min(N, T)$; then the default maximum lag is given by $$L^* = \max_k\{L(k) \mid L(k) < T^*\}$$

PANIC Long-run Variance Options

For use when computing a PANIC test using the MQ_c statistic.

Whitening Options

lag = *arg*	Lag specification: *integer* (user-specified number of lags), "a" (automatic selection).
infosel = *arg* *(default* = "aic"*)*	Information criterion for automatic selection: "aic" (Akaike), "sic" (Schwarz), "hqc" (Hannan-Quinn) (if "lag = a").
maxlag = *integer*	Maximum lag-length for automatic selection *(optional)* (if "lag = a"). The default is an observation-based maximum of $T^{1/3}$.

Kernel Options

kern = *arg* *(default* = "bart"*)*	Kernel shape: "none" (no kernel), "bart" (Bartlett), "bohman" (Bohman), "daniell" (Daniel), "parzen" (Parzen), "parzriesz" (Parzen-Riesz), "parzgeo" (Parzen-Geometric), "parzcauchy" (Parzen-Cauchy), "quadspec" (Quadratic Spectral), "trunc" (Truncated), "thamm" (Tukey-Hamming), "thann" (Tukey-Hanning), "tparz" (Tukey-Parzen), "user" (User-specified; see "kernwgt = " below).
kernwgt = *vector*	User-specified kernel weight vector (if "kern = user").
bw = *arg* *(default* = "nwfixed"*)*	Bandwidth: "fixednw" (Newey-West fixed), "andrews" (Andrews automatic), "neweywest" (Newey-West automatic), *number* (User-specified bandwidth).
nwlag = *integer*	Newey-West lag-selection parameter for use in nonparametric bandwidth selection (if "bw = neweywest").
bwoffset = *integer* *(default* = 0*)*	Apply integer offset to bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").
bwint	Use integer portion of bandwidth chosen by automatic selection method ("bw = andrews" or "bw = neweywest").

PANIC p-value Options

mcreps = *integer*	Number of Monte Carlo replications.

asymplen = *integer*	Asymptotic length of series.
seed = *number*	Specifies the random number generator seed
rng = *arg*	Specifies the type of random number generator. The key can be; 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").

Examples

```
oecd_rer.uroot2
```

The line above performs a PANIC unit root test on the series OECD_RER.

```
oecd_rer.uroot2(fsmethod=AH, mq=mqf, varlag=3)
```

The line above performs a PANIC unit root test using Ahn and Horenstein (2013) for factor selection determination and the MQ_f test for the number of common trends using a VAR(3) model.

```
oecd_rer.uroot2(test=cips, exog=trend, adnfosel=sic)
```

The line above performs a CIPS unit root test on the series OECD_RER, with ADF testing performed on each cross-section with a constant and trend, and ADF lag selection using the Schwarz criterion.

Cross-references

See "Unit Root Testing" on page 773 of *User's Guide II* for discussion of standard unit root tests performed on a single series, and "Cross-sectionally Independent Panel Unit Root Testing" on page 811 and "Cross-sectionally Dependent Panel Unit Root Tests" on page 822 of *User's Guide II* for discussion of unit roots tests performed on panel structured workfiles, groups of series, or pooled data.

See also `Series::uroot` (p. 863), `Group::uroot` (p. 512), `Group::uroot2` (p. 515), `Pool::uroot` (p. 689), `Pool::uroot2` (p. 692).

vratio	Series Views

Compute the Lo and MacKinlay (1988) variance ratio test using the original data, or the Wright (2000) rank, rank-score, or sign-based forms of the test.

Multiple comparisons are handled using Wald (Richardson and Smith, 1991) or multiple comparison variance ratio (Chow and Denning, 1993). Significance levels may be computed using the asymptotic distribution, or the wild or permutation bootstrap.

Syntax

Series View: series_name.**vratio***(options) lag_specification*

Series View: series_name.**vratio(grid***[, options]*) *start end [step]*

In the first form of the command, *lag_specification* should contain the lag values to test in the form of a list of integers, scalars, or a vector containing integer values greater than 1.

In the second form of the command, we include the "grid" option and specify a grid of lag values in the form

start end [step]

where *start* is the smallest lag, *end* is the largest required lag, and the optional *step* indicates which intermediate lags to consider. By default, *step* is set to 1 so that all lags from *start* through *end* will be included.

Options

out = *arg* *(default* = "table"*)*	Output type: "table" or "graph" of test results.
data = *arg* *(default* = "level"*)*	Form of data in series: "level" (random walk or martingale), "exp" (exponential random walk or martingale), "innov" (innovations to random walk or martingale).
method = *arg*	Test method: "orig" (Lo-MacKinlay test statistic), "rank" (rank statistic), "rankscore" (score statistic), "sign" (sign variance ratio statistic).
probcalc = *arg* *(default* = "anorm"*)*	Probability calculation: "norm" (asymptotic normal), "wildboot" (wild bootstrap), when "method = orig".
biased	Do not bias correct the variances.
iid	Do not use heteroskedastic robust S.E.
noc	Do not allow for drift / demean the data (for default "data = level").

stack	Compute estimates for stacked panel (in panel workfiles).
rankties = *arg* *(default* = "a"*)*	Tie handling for ranks: "i" (ignore), "a" (average), "r" (randomize).
prompt	Force the dialog to appear from within a program.
p	Print results.

Bootstrap Options

btreps = *integer* *(default* = 1000*)*	Number of bootstrap repetitions
btseed = *positive_integer*	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.
btrnd = *arg* *(default* = "kn" or method previously set using `rndseed` (p. 577) of the *Command and Programming Reference)*	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").
btdist = *arg* *(default* = "twopoint"*)*	Bootstrap distribution: "twopoint", "rademacher", "normal" (when "probcalc = wildboot").

Examples

The commands

```
jp.vratio(data=exp, biased, iid) 2 5 10 30
jp.vratio(out=graph, data=exp, biased, iid) 2 5 10 30
```

compute the Lo-MacKinley and the joint Chow-Denning and Wald tests for the homoskedastic random walk using periods 2, 5, 10, and 30. The results are displayed first in table, then in graph form. The individual test z-statistics use the asymptotic normal distribution and the Chow-Denning statistic uses the asymptotic Studentized Maximum Modulus distribution for evaluating significance.

```
series logjp = log(jp)
logjp.vratio(noc, iid, grid) 2 10 2
```

computes the same tests using periods 2, 4, 6, 8, and 10, with the bias-corrected variances computed without allowing for a mean/drift term.

To compute a heteroskedastic robust version of the last test, we simply remove the "iid" option:

```
logjp.vratio(noc, grid) 2 10 2
```

To compute the significance levels using the wild bootstrap,

```
jp.vratio(data=exp, biased, probcalc=wildboot, btreps=5000,
    btseed=1000, btrng=kn) 2 5 10 30
jp.vratio(data=exp, probcalc=wildboot, btdist=normal, btreps=5000,
    btseed=1000, btrng=kn) 2 5 10 30
```

Both commands produce bootstrap significance levels using 5000 replications with the Knuth generator and a seed of 1000. The second command substitutes bias corrects the variance estimates and changes the bootstrap random number distribution from the default two-step to the normal.

To perform Wright's rank and rank-score based tests,

```
vector(4) periods
periods.fill 2, 5, 10, 30
jp.vratio(data=exp, method=rank, btreps=5000, btseed=1000,
    btrng=kn) periods
jp.vratio(data=exp, method=rankscore, btreps=5000, btseed=1000,
    btrng=mt) periods
```

In panel settings, you may compute the statistic on the individual cross-sections and perform a joint Fisher test

```
exchange.vratio(data=exp, biased, probcalc=wildboot, btreps=5000,
    btseed=1000, btrng=kn) periods
```

or you may compute the statistic on the stacked data

```
series dexch = @dlog(exch)
dexch.vratio(stack, data=innov) periods
```

Cross-references

See "Variance Ratio Test" on page 845 of *User's Guide II* for discussion.

waveanova	Series Views

Perform wavelet variance decomposition of the series.

Syntax

Series View: series_name.**waveanova***(options)*

Options

Basic Options

variance = *arg (default* = "nobias"*)*	Wavelet variance type: "nobias" (unbiased variance), "bias" (biased variance).
ci = *arg (default* = "none"*)*	Confidence interval type: "none" (no CIs computed), "gauss" (asymptotic normal), "chisq" (asymptotic chi-square), "blimit" (band-limited).
cilevel = *arg (default* = 0.95*)*	Confidence interval coverage as a number between 0 and 1.
prompt	Force the dialog to appear from within a program.
p	Print results.

Wavelet Transform Options

transform = *arg (default* = "dwt"*)*	Wavelet transform type: "dwt" (discrete wavelet transform – DWT), "modwt" (maximum overlap DWT – MODWT). Note that when performing DWT, if the series length is not dyadic, a dyadic fix may be set with the "fixlen = " option
fixlen = *arg (default* = "mean"*)*	Fix dyadic lengths in DWT: "zeros" (pad remainder with zeros), "mean" (pad remainder with mean of series), "median" (pad remainder with median of series), "shorten" (cut series length to dyadic length preceding series length).
maxscale = *integer (default* = max possible)	Maximum scale for wavelet transform. The max possible is obtained as follows. Let T denote the series length and decompose T into its dyadic component and a remainder: $T = 2^z + r$, $r \geq 0$. The default *maxscale* m is then set with the following rules: DWT: (1) if $r = 0$ then $m = z$, otherwise (2) if expanding the series, $m = z + 1$ and (3) if contracting the series $m = z$. MODWT: $m = z$.

filter = *arg* *(default* = "h"*)*	Wavelet filter class: "h" (Haar), "d" (Daubechies), "la" (least asymmetric). If "filter = h" or "filter = la", the filter length may be specified using "flen = ". Wavelet filter boundary conditions are specified using the "bound = " option
flen = *integer*	Wavelet filter excess length as an even number between 2 and 20. For use when "filter = d" (*default* = 4) or "filter = la" (*default* = 8).
bound = *arg (default* = "p"*)*	Filter boundary handling: "p" (periodic), "r" (reflective).

Examples

```
dgp.waveanova(maxscale=2)
```

The line above will perform wavelet decomposition of variance of the series DGP using a Haar wavelet filter and the unbiased variance form, up to the second wavelet scale.

```
dgp.waveanova(maxscale=5, ci=gauss)
```

The line above will perform wavelet decomposition of variance using an unbiased variance form. It will also produce a 95% confidence interval using asymptotic Gaussian critical values.

Cross-references

See "Wavelet Analysis" on page 859 and "Wavelet Variance Decomposition" on page 867 of *User's Guide II* for discussion. See also "Wavelet Objects" on page 635 of *User's Guide I*.

See also `Series::wavedecomp` (p. 878), `Series::waveoutlier` (p. 880), `Series::wavethresh` (p. 883), and `Series::makewavelets` (p. 809).

wavedecomp	Series Views

Compute the wavelet transform of the series.

Syntax

Series View: series_name.**wavelet***(options)*

Options

transform = *arg* *(default* = "dwt"*)*	Wavelet transform type: "dwt" (discrete wavelet transform – DWT), "modwt" (maximum overlap DWT – MODWT), "mra" (DWT multiresolution analysis – DWT MRA), or "momra" (MODWT MRA). Note that when performing DWT or MRA, if the series length is not dyadic, a dyadic fix may be set with the "fixlen = " option
fixlen = *arg* *(default* = "mean"*)*	Fix dyadic lengths in DWT and MRA transforms: "zeros" (pad remainder with zeros), "mean" (pad remainder with mean of series), "median" (pad remainder with median of series), "shorten" (cut series length to dyadic length preceding series length).
maxscale = *integer* *(default* = max possible)	Maximum scale for wavelet transform. The max possible is obtained as follows. Let T denote the series length and decompose T into its dyadic component and a remainder: $T = 2^z + r$, $r \geq 0$. The default *maxscale* m is then set with the following rules: DWT: (1) if $r = 0$ then $m = z$, otherwise (2) if expanding the series, $m = z + 1$ and (3) if contracting the series $m = z$. MODWT: $m = z$.
filter = *arg* *(default* = "h"*)*	Wavelet filter class: "h" (Haar), "d" (Daubechies), "la" (least asymmetric). If "filter = h" or "filter = la", the filter length may be specified using "flen = ". Wavelet filter boundary conditions are specified using the "bound = " option
flen = *integer*	Wavelet filter excess length as an even number between 2 and 20. For use when "filter = d" (*default* = 4) or "filter = la" (*default* = 8).
bound = *arg (default* = "p"*)*	Filter boundary handling: "p" (periodic), "r" (reflective).
hidebound	Wavelet filter coefficients affected by the boundary will not be highlighted in the output graphs.
prompt	Force the dialog to appear from within a program.
p	Print results.

Examples

```
dgp.wavelet(maxscale=7)
```

The line above will perform the discrete wavelet transform of the series DGP using a Haar wavelet filter and up to the seventh wavelet scale.

```
dgp.wavelet(transform=modwt, maxscale=3, lter=D)
```

The line above will perform the maximum overlap discrete wavelet transform using a Daubechies wavelet filter of length 4 and up to the third wavelet scale.

```
dgp.wavelet(transform=mra, maxscale=4, lter=la, en=10, xlen=zeros)
```

The line above will perform a DWT multi-resolution analysis of the series DGP using a least asymmetric wavelet filter of length 10 and up to the fourth wavelet scale. It will also fix the non-dyadic length of the series by padding with zeros.

```
dgp.wavelet(transform=momra, maxscale=4, filter=d, flen=12,
    hidebound)
```

The line above will perform a MODWT multi-resolution analysis using a Daubechies wavelet filter of length 12 and up to the fourth wavelet scale. It will also turn off highlighting of wavelet coefficients on the boundary.

Cross-references

See "Wavelet Analysis" on page 859 and "Wavelet Transforms" on page 859 of *User's Guide II* for discussion. See also "Wavelet Objects" on page 635 of *User's Guide I*.

See also `Series::waveanova` (p. 876), `Series::waveoutlier` (p. 880), `Series::wavethresh` (p. 883), and `Series::makewavelets` (p. 809).

waveoutlier	Series Views

Perform wavelet outlier detection for the series.

Syntax

Series View: series_name.**waveoutlier***(options)*

Options

Basic Options

threshtype = *arg* *(default = "soft")*	Wavelet threshold type: "hard" (hard thresholding), "soft" (soft thresholding).
threshlim = *arg* *(default = "universal")*	Wavelet threshold limit type: "universal" (universal), "adaptive" (universal adaptive), "minimax" (minimax), "sureshrink" (SureShrink), "fdr" (false discovery rate). If "threshlim = sureshrink", the grid length may be specified using "ssglen = ". If "threshlim = fdr", the significance level may be specified using "fdrsig = "
mad = *arg* *(default = "gauss")*	Mean/median absolute deviation: "mean" (mean absolute deviation), "gauss" (median absolute deviation with Gaussian adjustment), "median" (median absolute deviation), "meanmedian" (mean median absolute deviation).
sslen = *arg* *(default = 10)*	Grid length used in determining the SureShrink limit.
fdrsig = *arg* *(default = .05)*	Significance level as a number between 0 and 1 for false discovery rate limit determination.
prompt	Force the dialog to appear from within a program.
p	Print results.

Wavelet Transform Options

transform = *arg* *(default* = "dwt"*)*	Wavelet transform type: "dwt" (discrete wavelet transform – DWT), "modwt" (maximum overlap DWT – MODWT). Note that when performing DWT, if the series length is not dyadic, a dyadic fix may be set with the "fixlen = " option
fixlen = *arg* *(default* = "mean"*)*	Fix dyadic lengths in DWT: "zeros" (pad remainder with zeros), "mean" (pad remainder with mean of series), "median" (pad remainder with median of series), "shorten" (cut series length to dyadic length preceding series length).
maxscale = *integer* *(default* = max possible)	Maximum scale for wavelet transform. The max possible is obtained as follows. Let T denote the series length and decompose T into its dyadic component and a remainder: $T = 2^z + r$, $r \geq 0$. The default *maxscale* m is then set with the following rules: DWT: (1) if $r = 0$ then $m = z$, otherwise (2) if expanding the series, $m = z + 1$ and (3) if contracting the series $m = z$. MODWT: $m = z$.
filter = *arg* *(default* = "h"*)*	Wavelet filter class: "h" (Haar), "d" (Daubechies), "la" (least asymmetric). If "filter = h" or "filter = la", the filter length may be specified using "flen = ". Wavelet filter boundary conditions are specified using the "bound = " option
flen = *integer*	Wavelet filter excess length as an even number between 2 and 20. For use when "filter = d" (*default* = 4) or "filter = la" (*default* = 8).
bound = *arg (default* = "p"*)*	Filter boundary handling: "p" (periodic), "r" (reflective).

Examples

```
dgp.wavetoutlier(maxscale=1, threshtype=hard,
    threshlim=meanmedian)
```

The line above will perform the Bilen and Huzurbazar (2002) wavelet outlier detection procedure on a series called DGP. It will use the Haar wavelet filter by default, execute to single scale, will use hard thresholding, and the mean median absolute deviation for the threshold limit. The latter options are those used in the original paper.

Cross-references

See "Wavelet Analysis" on page 859 and "Wavelet Variance Decomposition" on page 867 of *User's Guide II* for discussion. See also "Wavelet Objects" on page 635 of *User's Guide I*.

See also `Series::wavedecomp` (p. 878), `Series::waveoutlier` (p. 880), `Series::wavethresh` (p. 883), and `Series::makewavelets` (p. 809).

wavethresh	Series Views

Perform wavelet thresholding (denoising) of the series.

Syntax

Series View: series_name.**wavethresh***(options)*

Options

Basic Options

threshtype = *arg* *(default = "soft")*	Wavelet threshold type: "hard" (hard thresholding), "soft" (soft thresholding).
threshlim = *arg* *(default = "universal")*	Wavelet threshold limit type: "universal" (universal), "adaptive" (universal adaptive), "minimax" (minimax), "sureshrink" (SureShrink), "fdr" (false discovery rate). If "threshlim = sureshrink", the grid length may be specified using "ssglen = ". If "threshlim = fdr", the significance level may be specified using "fdrsig = "
wavevar = *arg* *(default = "gauss")*	Wavelet coefficient variance method: "mean" (mean absolute deviation), "gauss" (median absolute deviation with Gaussian adjustment), "median" (median absolute deviation), "meanmedian" (mean median absolute deviation).
sslen = *arg* *(default = 10)*	Grid length used in determining the SureShrink limit.
fdrsig = *arg* *(default = .05)*	Significance level as a number between 0 and 1 for false discovery rate limit determination.
prompt	Force the dialog to appear from within a program.
p	Print results.

Wavelet Transform Options

transform = *arg* (*default* = "dwt")	Wavelet transform type: "dwt" (discrete wavelet transform – DWT), "modwt" (maximum overlap DWT – MODWT). Note that when performing DWT, if the series length is not dyadic, a dyadic fix may be set with the "fixlen = " option
fixlen = *arg* (*default* = "mean")	Fix dyadic lengths in DWT: "zeros" (pad remainder with zeros), "mean" (pad remainder with mean of series), "median" (pad remainder with median of series), "shorten" (cut series length to dyadic length preceding series length).
maxscale = *integer* (*default* = max possible)	Maximum scale for wavelet transform. The max possible is obtained as follows. Let T denote the series length and decompose T into its dyadic component and a remainder: $T = 2^z + r$, $r \geq 0$. The default *maxscale* m is then set with the following rules: DWT: (1) if $r = 0$ then $m = z$, otherwise (2) if expanding the series, $m = z + 1$ and (3) if contracting the series $m = z$. MODWT: $m = z$.
filter = *arg* (*default* = "h")	Wavelet filter class: "h" (Haar), "d" (Daubechies), "la" (least asymmetric). If "filter = h" or "filter = la", the filter length may be specified using "flen = ". Wavelet filter boundary conditions are specified using the "bound = " option
flen = *integer*	Wavelet filter excess length as an even number between 2 and 20. For use when "filter = d" (*default* = 4) or "filter = la" (*default* = 8).
bound = *arg* (*default* = "p")	Filter boundary handling: "p" (periodic), "r" (reflective).

Examples

```
dgp.wavethresh(maxscale=3)
```

The line above will perform wavelet thresholding on a series called DGP, using a soft threshold and the universal threshold limit. This procedure is also known as VisuShrink. It will do so up to the third wavelet scale and will use the MAD with a Gaussian correction for the measure of loss.

```
dgp.wavethresh(filter=d, flen=4, maxscale=1, threshtype=hard,
    threshlim=fdr)
```

The line above will perform wavelet thresholding using a Daubechies wavelet filter of length 4, up to the first wavelet scale. Furthermore, it will use a hard threshold and the false discovery rate limit with significance level 0.05.

Cross-references

See "Wavelet Analysis" on page 859 and "Wavelet Threshold (Denoising)" on page 872 of *User's Guide II* for discussion. See also "Wavelet Objects" on page 635 of *User's Guide I*.

See also `Series::wavedecomp` (p. 878), `Series::waveanova` (p. 876), `Series::waveoutlier` (p. 880), and `Series::makewavelets` (p. 809).

x11	Series Procs

Seasonally adjust series using the Census X11.2 method.

Earlier versions of EViews provided (historical) X11 routines as a separate procedure (see but the programs do not run on 64-bit operating systems and are no longer supported.

Syntax

series_name.**x11***(options) adj_name [fac_name]*

Cross-references

See also `Series::seas` (p. 828), `Series::x12` (p. 885), `Series::x13` (p. 890), and `Series::tramoseats` (p. 859).

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 = *arg* *(default* = "m"*)*	Seasonal adjustment method: "m" (multiplicative adjustment; *Series must take only non-negative values*), "a" (additive adjustment), "p" (pseudo-additive adjustment), "l" (log-additive seasonal adjustment; *Series must take only positive values).*
filter = *arg* *(default* = "msr"*)*	Seasonal filter: "msr" (automatic, moving seasonality ratio), "x11" (X11 default), "stable" (stable), "s3x1" (3x1 moving average), "s3x3" (3x3 moving average), "s3x5" (3x5 moving average), "s3x9" (3x9 moving average), "s3x15" (3x15 moving average seasonal filter; *Series must have at least 20 years of data*).
save = "*arg*"	Optionally saved series keyword enclosed in quotes. List the extension (given in Table 6-8, p.71 of the *X12-ARIMA Reference Manual*) for the series you want to save. The created series will use names of the form *basename,* followed by a series keyword specific suffix. Commonly used options and suffixes are: ""d10"" (final seasonal factors, saved with suffix "_sf"), ""d11"" (final seasonally adjusted series using "_sa"), ""d12"" (final trend-cycle component using "_tc"), ""d13"" (final irregular component using "_ir"). All other options are named using the option symbol. For example "save = "d16"" will store a series named *basename*_D16. To save more than two series, separate the list with a space. For example, "save = "d10 d12"" saves the seasonal factors and the trend-cycle series.
tf = *arg*	Transformation for regARIMA: "logit" (Logit transformation), "auto" (automatically choose between no transformation and log transformation), *number* (Box-Cox power transformation using specified parameter; use "tf = 0" for log transformation).
sspan	Sliding spans stability analysis. *Cannot be used along with the "h" option.*
history	Historical record of seasonal adjustment revisions. *Cannot be used along with the* "sspan" *option.*
check	Check residuals of regARIMA.
outlier	Outlier analysis of regARIMA.

x11reg = *arg*	Regressors to model the irregular component in seasonal adjustment. Regressors must be chosen from the predefined list in Table 6-14, p. 88 of the *X12-ARIMA Reference Manual*. To specify more than one regressor, separate by a space within the double quotes.
reg = *arg_list*	Regressors for the regARIMA model. Regressors must be chosen from the predefined list in Table 6-17, pp. 100-101 of the *X12-ARIMA Reference Manual*. To specify more than one regressor, separate by a space within the double quotes.
arima = *arg*	ARIMA spec of the regARIMA model. Must follow the X12 ARIMA specification syntax. *Cannot be used together with the "amdl = " option*.
amdl = f	Automatically choose the ARIMA spec. Use forecasts from the chosen model in seasonal adjustment. *Cannot be used together with the* "arima = " *option and must be used together with the* "mfile = " *option*.
amdl = b	Automatically choose the ARIMA spec. Use forecasts and backcasts from the chosen model in seasonal adjustment. *Cannot be used together with the "arima = " option and must be used together with the "mfile = "* option.
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). *Must be used together with the "amdl = " option*.
modelsmpl = *arg*	Sets the subsample for fitting the ARIMA model. Either specify a sample object name or a sample range. *The model sample must be a subsample of the current workfile sample and should not contain any breaks*.
mfile = *arg*	Specifies the file name (include the extension, if any) that contains a list of ARIMA specifications to choose from. *Must be used together with the "amdl = " option*. The default is the X12A.MDL file provided by the Census.
outsmpl	Use out-of-sample forecasts for automatic model selection. Default is in-sample forecasts. *Must be used together with the "amdl = "* option.
plotspectra	Save graph of spectra for differenced seasonally adjusted series and outlier modified irregular series. The saved graph will be named GR_*seriesname*_SP.
prompt	Force the dialog to appear from within a program.
p	Print X12 procedure results.

Other Options

hma = *integer*	Specifies the Henderson moving average to estimate the final trend-cycle. The X12 default is automatically selected based on the data. To override this default, specify an *odd integer between 1 and 101*.
sigl = *arg*	Specifies the lower sigma limit used to downweight extreme irregulars in the seasonal adjustment. The default is 1.5 and you can specify any positive real number.
sigh = *arg*	Specifies the upper sigma limit used to downweight extreme irregulars in the seasonal adjustment. The default is 2.5 and you can specify any positive real number less than the lower sigma limit.
ea	Nonparametric Easter holiday adjustment (x11easter). *Cannot be used together with the "easter[w]" regressor in the "reg = " or "x11reg = " options*.
f	Appends forecasts up to one year to some optionally saved series. Forecasts are appended only to the following series specified in the "*save* = " option: ""b1"" (original series, adjusted for prior effects), ""d10"" (final seasonal factors), ""d16"" (combined seasonal and trading day factors).
flead = *integer*	Specifies the number of periods to forecast (to be used in the seasonal adjustment procedure). The default is one year and you can specify an integer up to 60.
fback = *integer*	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 = "*. *Must be used together with the "x11reg = " option*.
aicreg	Test (based on AIC) whether to retain the regressors specified in "*reg = "*. *Must be used together with the "reg = " option*.
sfile = *arg*	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 534 of *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. 828), `Series::x11` (p. 885), `Series::x13` (p. 890), and `Series::tramoseats` (p. 859).

x13	Series Procs

Seasonally adjust series using the Census X-13 method.

Census X-13 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.**x13***(options) [@reg(regopts)] [@arima(arimaopts)] [@x11arima(x11arimaopts)] [@tramo(tramopts)] [@x11(x11opts)] [@seats(seatsopts)]*

You should follow the `x13` keyword with general options and optionally, specifications for regression (`@reg`), ARIMA (either manual (`@arima`), X-11 automatic (`@x11arima`), or TRAMO automatic (`@tramo`)), and seasonal adjustment (either X-11 based (`@x11`) or SEATS based (`@seats`)) components.

When using X-13, EViews calls the X-13 executable written by the US Census. Many of the options available in the EViews `x13` command closely mirror those available in the original X-13 executable. As such in the documentation of options that follows, we often make refer-

ence to the original Census documentation, which is included in PDF form with the rest of your EViews documentation.

You should note that while EViews does not offer direct support for the full set of Census X-13, most of the specification statements allow you to directly add Census X-13 options using the `extra` option. For example, although EViews does not directly support the "constant" or "adjust" options of the X-13 Transformation spec (see Section 7.18 of the Census X-13 documentation), you may instruct Census X-13 to use those options by adding the option

```
tfextra="constant adjust"
```

to your EViews X-13 command.

Specification Component Options

The regression, manual, X-11 or TRAMO automatic ARIMA, and X11 or SEATS based seasonal adjustment specification components,

[@reg(regopts)] [@arima(arimaopts)] [@x11arima(x11arimaopts)] [@tramo(tramopts)] [@x11(x11opts)] [@seats(seatsopts)]

each take various options. In this section, we outline the possible settings for each of these components.

Regression Specification (@reg)

Include exogenous variables in the ARIMA regression. If `@arima`, `@x11arima`, and `@tramo` specs are not included, a simple regression without ARIMA is performed. See Section 7.13 of the Census X-13 documentation for details.

		X-13 Equivalent Option
regs = *list*	Quoted, space delimited, list of X-13 built-in variables to use as regressors. For a full list of available variable types, see Table 7.27 of the X-13 documentation.	Variables =
userregs = *list*	Quoted, space delimited, list of series to include as user-variables in the regression. Each member of the list should be a valid series expression (*e.g.* "X" or "log(X)").	User =

usertypes = *list*	Quoted, space delimited, list of user variable types. The number of elements in the list must be the same as the number of elements in the "userregs" list, or should contain only one type, which will apply to *all* variables listed in "userregs". Types can be "constant" (constant), "seasonal" (seasonal), "td" (trading-day), "tdstock" (trading-day stock), "lom" (length of month), "loq" (length of quarter), "lpyear" (leap year), "easter" (Easter), "thanks" (Thanks-giving), "labor" (Labor day), "ao", "ls", "rp", "so" or "tc" (outlier effects), "transitory" (SEATS transitory), "holiday1", "holiday2", "holiday3", "holiday4" or "holiday5" (user-defined holidays), or "user" (none of the above).	Usertype =
aictest = *list*	Quoted, space delimited, list of variables to include in the AIC based variable selection routine. Only certain variable types may be included in this list: "td", "tdnolpyear", "tdstock", "td1coef", "td1nolpyear", "tdstock1coef", "lom", "loq", "lpyear", "easter", "easterstock", and "user". See Table 7.27 of the X-13 documentation for a description of these variables.	AICtest =
chitest	Perform a chi-squared test for inclusion of all user-defined holiday variables.	Chi2test = yes
regextra = *list*	A quoted, space delimited, list of any extra regression options included as part of X-13.	

Manual ARIMA (@arima)

The `@arima` spec allows you to specify manually an ARIMA model to be used. Note that an `@arima` spec cannot be used at the same time as an `@x11arima` or `@tramo` spec. See Section 7.1 of the X-13 documentation for details.

		X-13 Equivalent Option
model = *text*	Set the ARIMA model by specifying the model in standard "(p,d,q)(P,D,Q)" format. The *text* argument must be surrounded by quotes. See the Census X-13 documentation for details on the syntax of *text*.	Model =
ar = *list*	Set the starting values for the AR parameters in the ARIMA model. *list* should be a quoted, comma separated list of AR parameters. A blank space between commas may be used to use the default starting value for a parameter. To fix a parameter at its starting value, you may append the "f" character to the end of the parameter value (*e.g.*, to fix a parameter at 0.7, use "0.7f")	AR =
ma = *list*	Set the starting values for the MA parameters in the ARIMA model. *list* should be a quoted, comma separated list of MA parameters. A blank space between commas may be used to use a default starting value for a parameter. To fix a parameter at its starting value, you may append the "f" character to the end of the parameter value (*e.g.*, to fix a parameter at 0.7, use "0.7f")	MA =

Automatic ARIMA Selection using X-11 (@x11arima)

Use X-11-ARIMA to automatically choose an ARIMA model. Note that an `@x11arima` spec cannot be used at the same time as an `@arima` or `@tramo` spec. See Section 7.12 of the Census X-13 documentation for more details.

		X-13 Equivalent Option
mfile = *file*	Specify a file on disk containing the list of possible ARIMA models to choose from. Note that this option cannot be used with the "max*" options. See the *Details* portion of Section 7.12 of the Census X-13 documentation for details on how to create a valid ARIMA model file.	File =
maxar = *integer*	Set the maximum number of AR terms in models to be selected from. Cannot be used with the "mfile = " option.	
maxdiff = *integer*	Set the maximum differencing order in models to be selected from. Cannot be used with the "mfile = " option.	
maxma = *integer*	Set the maximum number of MA in models to be selected from. Cannot be used with the "mfile = " option.	
maxsar = *integer*	Set the maximum number of seasonal AR terms in models to be selected from. Cannot be used with the "mfile = " option.	
maxsdiff = *integer*	Set the maximum seasonal differencing order in models to be selected from. Cannot be used with the "mfile = " option.	
maxsma = *integer*	Set the maximum number of seasonal MA in models to be selected from. Cannot be used with the "mfile = " option.	
amdl = F	Use only forecasts from the ARIMA model in model evaluation. Without this option, both forecasts and backcasts are used.	Mode = f
outsmpl	Use out of sample forecast errors during model evaluation	Outofsample = yes
best	Model selection tests all possible models and chooses the most likely. Without this option, the model selection routine will chose the first model that matches model selection criteria.	Method = best

flim = *number*	Sets the acceptance threshold for the within-sample forecast error test.	Fcstlim =
blim = *number*	Sets the acceptance threshold for the within-sample backcast error test. Only applies if the amdl = f option is not set.	Bcstlim =
x11aimaextra = *list*	A quoted, space delimited, list of any extra X-11 automatic ARIMA options included as part of X-13.	

Automatic ARIMA selection using TRAMO (@tramo)

Use TRAMO to automatically choose an ARIMA model. Note that an `@tramo` spec cannot be used at the same time as an `@arima` or `@x11arima` spec. See Section 7.2 of the Census X-13 documentation for more details.

		X-13 Equivalent Option
maxorder = "(*int1, int2)"*	Set the maximum order of AR, MA, SAR and SMA terms in candidate models. *int1* sets the maximum for AR and MA terms, and *int2* sets the maximum for seasonal AR and seasonal MA terms.	Maxorder =
maxdiff = "(*int1, int2*)"	Sets the maximum differencing and seasonal differencing in candidate models. *int1* sets the maximum differencing, and *int2* sets the maximum seasonal differencing. Note the "maxdiff" option cannot be used along with the "diff" option.	Maxdiff =
diff = "(*int1, int2*)"	Set a fixed differencing for candidate models – i.e. differencing will not be automatically chosen. *int1* sets differencing, and *int2* sets seasonal differencing. Note that the "diff" option cannot be used along with the "maxdiff" option.	Diff =
nomixed	Do not allow mixed (i.e. models with both AR and MA terms) amongst the candidate models.	Mixed = no
rejectfcst	Test the out-of-sample forecast error of the final three years of data with the identified model to determine if forecast extension should be applied.	Rejectfcst = yes

flim = *number*	Sets the acceptance threshold for the within-sample forecast error test of the final identified model. Only applies if the "rejectfcst" option is set.	Fcstlim =
lbqlim = *number*	Acceptance criterion for confidence coefficient of the Ljung-Box Q statistic.	Ljungboxlimit =
acceptdef	Controls whether the default model is chosen if the Ljung-Box Q statistic for its model residuals is acceptable.	Acceptdefault = yes
nomu	Do not check for significance of the constant term in candidate models	Checkmu = no
tramoextra = *list*	A quoted, space delimited, list of any extra TRAMO automatic ARIMA options included as part of X-13.	

X-11 Seasonal Adjustment (@x11)

Perform an X-11 based seasonal adjustment. Note that an `@x11` spec cannot be included at the same time as an `@seats` spec. See Section 7.19 of the Census X-13 documentation for details.

		X-13 Equivalent Option
mode = *arg*	Sets the mode of seasonal adjustment decomposition: "mult" (multiplicative), "add" (additive), "pseudoadd" (pseudo-additive), or "logadd" (log-additive). The default is "mult".	Mode =
type = *arg*	Sets the type of seasonal adjustment: "sum" (summary), "trend", or "sa" (default). See the Census X-13 documentation for a full description of each.	Type =
filter = *arg*	Specifies the seasonal moving average filter to use: "s3x1" (3x1 moving average), "s3x3", (3x3 moving average), "s3x5" (3x5 moving average), "s3x9" (3x9 moving average), "s3x15" (3x15 moving average), "stable" (Stable seasonal filter), "x11default" (3x3 followed by a 3x5), or "msr" (default). You can set a different filter for each MA term by entering multiple values for *key*, separated by commas and surrounded in quotes (*e.g.*, filter = "s3x1, s3x3, s3x9").	Seasonalma =

fcast	Append forecasted values to certain output series. See the Census X-13 documentation for a list of available series. This option must be used with the "flen = " general option.	appendfcst
bcast	Pre-pend backcasted values to certain output series. See the Census X-13 documentation for a list of available series. This option must be used with the "blen = " general option.	appendbcst
trendma = *integer*	Length of the Henderson moving average to use. *integer* may be any odd integer between 1 and 101.	Trendma =
x11extra = *list*	A quoted, space delimited, list of any extra X-11 seasonal adjustment options included as part of X-13.	

SEATS Seasonal Adjustment (@seats spec)

Perform a SEATS based seasonal adjustment. Note that an `@seats` spec cannot be included at the same time as an `@x11` spec. See Section 7.14 of the Census X-13 documentation for more details.

		X-13 Equivalent Option
fcast	Append forecasted values to certain output series. See the Census X-13 documentation for a list of available series. This option must be used with the "flen = " general option.	appendfcst
bcast	Prepend backcasted values to certain output series. See the Census X-13 documentation for a list of available series. This option must be used with the "blen = " general option.	appendbcst
hp	Decompose the trend-cycle component into a long-term component using the Hodrick-Prescott filter.	Hpcycle = yes
nostat	Do not accept any stationary seasonal ARIMA models, and convert the seasonal part to (0, 1, 1).	Statseas = no
qmax = *integer*	Sets a limit for the Ljung-Box Q statistic, which is used to determine if the model provided to SEATS is of acceptable quality.	Qmax =
seatsextra = *list*	A quoted, space delimited, list of any extra SEATS seasonal adjustment options included as part of X-13.	

Force Annual Totals (@force spec)

Force the annual totals of the adjusted series to match those of the original series. See Section 7.6 of the Census X-13 documentation for additional details.

		X-13 Equivalent Option
type = *arg*	Change the forcing algorithm. *Arg* can be "denton" or "regress." Default is "denton."	type
mode = *arg*	Change the mode of the algorithm. *Arg* can be "ratio" or "diff." Default is "ratio."	mode
target = *arg*	Specify the target for matching the annual totals. *Arg* can be "cal" (Calendar Adjusted Series), "perm" (Original series adjusted for permanent prior adjustment factors), or "both." If this option is not used, the original series is used as the target.	target
round	Use rounded data for matching.	round
rho = *val*	Specify the value for rho.	rho
lambda = *val*	Specify the value for lambda.	lambda

Options

General Options

savespec = *name*	Save a copy of the X-13 spec file as a text object in the workfile. This can be useful as a template when making your own spec file to use with the "spec = " option.
save = *list*	Output series to save from the seasonal adjustment routine. *list* should be space delimited, in quotes, and contain the list of *small* identifiers from Table 7.46 (if doing X-11) or Table 7.30 (if doing SEATS) of the Census X-13 documentation. If this option is omitted, EViews will save the seasonally adjusted series (D11 for X-11, and S11 for SEATS).
errlog = *name*	Save a copy of the error log as a text object in the workfile. The error log will only be saved if the X-13 executable created an error message.

spec = *name*	User supplied X-13 spec file. Either a file on disk, or a text object in the workfile. Note that this option overrides all other options apart from "prompt", "save", "savespec" and "errlog". Note you can use the "savespec" option to generate a spec file for editing. If your spec file contains a SERIES specification, EViews will use it. If it does not, EViews will generate one. In general we recommend letting EViews generate the SERIES part of your spec file.
html	Generate html formatted output. If not specified, text formatted output is the default.
prompt	Force dialog to show in program
p	Print output from the procedure.

Transformation Options:

Sets options for the transformation of data used. See the Transformation section, 7.18, of the Census X-13 documentation for more details.

tf = *arg*	Employ data transformation: "logit" (logistic), "auto" (choose between log or none), "log" (natural log), or *number* (where *number* is a Box-Cox power parameter. for the Box-Cox transformation).
tfextra = *list*	A quoted, space delimited, list of any extra transformation options included as part of X-13. The full set of possible options is provided in Section 7.18 of the X-13 documentation.

Automatic Outlier Options

Sets options for automatic outlier detection. Note that specific outliers can be included in the optional `@reg` spec. See the Outlier section, 7.11, of the Census X-13 documentation for more details.

		X-13 Equivalent Option
outcrit = *arg*	Value to which the absolute values of the outlier *t*-statistics are compared to detect outliers in automatic outlier detection.	Critical =
outls = *arg*	Compute *t*-statistics to test the null hypotheses that each run of 2,..., *outls* successive level shifts cancel to form a temporary level shift.	Lsrun =
outall	Sets the outlier detection method to all at once (as opposed to one at a time).	Method = addall

outtype = *list*	List of types of outliers to include in outlier detection (quoted and space delimited). Members of the list can include "ao" (additive outlier), "ls" (level shift), "tc" (temporary change), "so" (seasonal outliers). You may use the special unquoted keyword "all" to include all types, as in "outtype = all".	Types =
outspan = *arg*	Set the dates to search between. *arg* should be two dates, surrounded in quotes, of the format "*YYYY.MON YYYY.MON*" (for monthly data) or "*YYYY.Q YYYY.Q*" (for quarterly), where MON is a three letter month abbreviation, and Q is an integer representing the quarter.	Span =
outextra = *list*	A quoted, space delimited, list of any extra outlier options included as part of X-13.	

Estimation Options

Sets estimation options for the ARIMA/Regression estimation. Only relevant if a `@reg`, `@arima`, `@x11arima`, or a `@tramo` spec are included. See the Estimation section, 7.5, of the Census X-13 documentation for more details.

		X-13 Equivalent Option
tol = *number*	Set the convergence tolerance	Tol =
iter = *integer*	Set the maximum number of iterations	Maxiter =
exact = *arg*	Specifies the use of an exact or a conditional likelihood for estimation:. "arma" (use exact likelihood for both AR and MA terms), "ma" (use conditional likelihood for AR and exact likelihood for MA terms), and "none" (use conditional likelihood for both sets of terms).	Exact =
arimasmpl = *arg*	Set the estimation sample. *arg* should be two dates, surrounded in quotes, of the format "*YYYY.MON YYYY.MON*" (for monthly data) or "*YYYY.Q YYYY.Q*" (for quarterly), where MON is a three letter month abbreviation, and Q is an integer representing the quarter.	Modelspan = (in the SERIES spec, section 7.15).
estextra = *list*	A quoted, space delimited, list of any extra estimation options included as part of X-13.	

Forecast Options

Sets forecast options for the ARIMA/Regression estimation. Only relevant if a `@reg`, `@arima`, `@x11arima`, or a `@tramo` spec are included. See the Forecast section, 7.7, of the Census X-13 documentation for more details.

		X-13 Equivalent Option
flen = *integer*	Length of forecast to perform. May be between 0 and 60. Note that if performing SEATS seasonal adjustment, the forecast length will be adjusted upwards to 2 years (24 months or 8 quarters).	Maxlead =
blen = *integer*	Length of backcast to perform. May be between 0 and 60.	Maxback =
forclognorm	Adjust forecasts to reflect that forecasts are generated from a log-normal distribution.	Lognormal
forcextra = *list*	A quoted, space delimited, list of any extra forecast options included as part of X-13.	

Examples

As an example of using X-13, we will seasonally adjust some data obtained from FRED. The workfile, "X13 Macro.wf1" contains monthly non-seasonally adjusted US unemployment data from January 2005 to June 2012 in a series called UNRATENSA.

The command:

```
unratensa.x13(save="d12 d10 d13 d11") @x11arima @x11
```

performs X-11 based seasonal adjustment using X-11-ARIMA to automatically select the ARIMA model, using the default set of candidate models. We save the final seasonally adjusted series (D11), the final trend series (D12), the final seasonal factors (D10), and the irregular component (D13) as series in the workfile.

The command:

```
unratensa.x13(save="s12s10s13s11 afd", outtype="all", flen=24)
    @tramo(maxdiff="(2,1)", maxorder="(2,1)")
    @seats(fcast, seatsextra="signifsc=0.5")
```

Performs SEATS based seasonal adjustment, where TRAMO is used to automatically detect the best ARIMA model (with a maximum AR and MA order of 2, a maximum SAR and SMA order of 1, maximum differencing of 2, and a maximum seasonal differencing of 1), automatic outlier detection is included, with all types of outliers detected, and 24 periods of forecasted values are kept. Note that we use the `seatsextra` option to specify the non-included signifsc SEATS option. We save the final seasonally adjusted series (s11), the final trend

series (s12), the final seasonal factors (s10), the irregular component (s13), and the forecasted seasonally adjusted values (afd) as series in the workfile.

Examples

See "Census X-13" on page 515 of *User's Guide I* for a discussion of the Census X-13 program. The full documentation for the Census program, *X-13ARIMA-SEATS Reference Manual*, can be found in the "docs" subdirectory of your EViews installation directory in the PDF file "X-13 Reference Manual.pdf".

See also `Series::seas` (p. 828), `Series::x11` (p. 885), `Series::x12` (p. 885), and `Series::tramoseats` (p. 859).

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.

Sspace

State space object. Estimation and evaluation of state space models using the Kalman filter.

Sspace Declaration

To declare a sspace object, use the `sspace` keyword, followed by a valid name.

Sspace Method

Sspace Views

Sspace Procs

makefilter..... make new Kalman Filter (p. 916).
makegrads..... make group containing the gradients of the log likelihood (p. 916).
makemodel ..... make a model object containing equations in sspace (p. 917).
makesignals ..... make group containing signal and residual series (p. 918).
makestates ..... make group containing state series (p. 919).
olepush ..... push updates to OLE linked objects in open applications (p. 921).
setattr..... set the value of an object attribute (p. 924).
updatecoefs ..... update coefficient vector(s) from sspace (p. 930).

Sspace Data Members

Scalar Values

@coefcov(i,j)..... covariance of coefficients i and j.
@coefs(i) ..... coefficient i.
@eqncoef(k) ..... number of estimated coefficients in equation k.
@eqregobs(k)..... number of observations in signal equation k.
@linecount ..... scalar containing the number of lines in the Sspace object.
@sddep(k) ..... standard deviation of the signal variable in equation k.
@ssr(k)..... sum-of-squared standardized one-step ahead residuals for equation k.
@stderrs(i)..... standard error for coefficient i.
@tstats(i) ..... t-statistic value for coefficient i.

Scalar Values (system level data)

@aic ..... Akaike information criterion for the system.
@hq..... Hannan-Quinn information criterion for the system.
@logl ..... value of the log likelihood function.
@ncoefs ..... total number of estimated coefficients in the system.
@neqns..... number of equations for observable variables.
@regobs..... number of observations in the system.
@sc..... Schwarz information criterion for the system.
@totalobs..... sum of "@eqregobs" from each equation.

Vectors and Matrices

@coefcov ..... covariance matrix for coefficients of equation.
@coefs ..... coefficient vector.
@final_state ..... matrix of final states.
@final_statecov.... (sym) covariance matrix of final state covariances.
@init_state..... matrix of initial states.
@init_statecov..... (sym) covariance matrix of initial state covariances.
@pvals..... vector containing the coefficient probability values.

@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_signalse.....matrix or vector of the standard errors of the one-step ahead predicted signals.
@pred_err.....matrix or vector of one-step ahead prediction errors.
@pred_errcov .....matrix where every row is the @vech of the one-step ahead prediction error covariance.
@pred_errcovinv...matrix where every row is the @vech of the inverse of the one-step ahead prediction error covariance.
@pred_errse.....matrix or vector of the standard errors of the one-step ahead prediction errors.
@pred_errstd .....matrix or vector of standardized one-step ahead prediction errors.
@pred_state.....matrix or vector of one-step ahead predicted states.
@pred_statecov.....matrix where each row is the @vech of the one-step ahead predicated state covariance.
@pred_statese.....matrix or vector of the standard errors of the one-step ahead predicted states.
@pred_stateerr.....matrix or vector of one-step ahead predicted state errors.
@curr_err .....matrix or vector of filtered error estimates.
@curr_gain.....matrix or vector where each row is the @vec of the Kalman gain.
@curr_state .....matrix or vector of filtered states.
@curr_statecov .....matrix where every row is the @vech of the filtered state covariance.
@curr_statese .....matrix or vector of the standard errors of the filtered state estimates.
@sm_signal .....matrix or vector of smoothed signal estimates.

@sm_signalcov .... matrix where every row is the @vech of the smoothed signal covariance.
@sm_signalse..... matrix or vector of the standard errors of the smoothed signals.
@sm_signalerr ..... matrix or vector of smoothed signal error estimates.
@sm_signalerrcov matrix 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_state..... matrix or vector of smoothed states.
@sm_statecov ..... matrix where each row is the @vech of the smoothed state covariances.
@sm_statese..... matrix or vector of the standard errors of the smoothed state.
@sm_stateerr ..... matrix or vector of the smoothed state errors.
@sm_stateerrcov .. matrix where each row is the @vech of the smoothed state error covariance.
@sm_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.

String Values

@attr("*arg*") ..... string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@command..... full command line form of the state space estimation command. Note this is a combination of `@method` and `@options`.
@description ..... string containing the Sspace object's description (if available).
@detailedtype ..... returns a string with the object type: "SSPACE".
@displayname..... returns the Sspace object's display name. If the Sspace has no display name set, the name is returned.
@line(i) ..... returns a string containing the i-th line of the Sspace object.
@name ..... returns the Sspace's name.
@options..... command line form of sspace estimation options.
@remarks ..... string containing the sspace object's remarks (if available).
@smpl ..... sample used for estimation.
@svector ..... returns an Svector where each element is a line of the Sspace object.
@svectornb ..... same as `@svector`, with blank lines removed.
@type ..... returns a string with the object type: "SSPACE".
@updatetime..... returns a string representation of the time and date at which the Sspace was last updated.

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.

Cross-references

See "Specifying a State Space Model in EViews" on page 1156 of *User's Guide II* for a discussion of specification syntax.

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

ind = *arg*	Specifies whether and how to draw the individual coefficient intervals. The default is "ind = line" which plots the individual coefficient intervals as dashed lines. "ind = none" does not plot the individual intervals, while "ind = shade" plots the individual intervals as a shaded rectangle.
size = *number* (*default* = 0.95)	Set the size (level) of the confidence ellipse. You may specify more than one size by specifying a space separated list enclosed in double quotes.
dist = *arg*	Select the distribution to use for the critical value associated with the ellipse size. The default depends on estimation object and method. If the parameter estimates are least-squares based, the $F(2, n-2)$ distribution is used; if the parameter estimates are likelihood based, the $\chi^2(2)$ distribution will be employed. "dist = f" forces use of the F-distribution, while "dist = c" uses the χ^2 distribution.
prompt	Force the dialog to appear from within a program.
p	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 Intervals and Confidence Ellipses" on page 203 of *User's Guide II* for discussion.

See also `Sspace::wald` (p. 930).

clearhist	Sspace Procs

Clear the contents of the history attribute for sspace objects.

Removes the sspace's history attribute, as shown in the label view of the sspace.

Syntax

sspace_name.**clearhist**

Examples

```
s1.clearhist
s1.label
```

The first line removes the history from the sspace S1, and the second line displays the label view of S1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Sspace::label` (p. 914).

clearremarks	Sspace Procs

Clear the contents of the remarks attribute.

Removes the sspace's remarks attribute, as shown in the label view of the sspace.

Syntax

sspace_name.**clearremarks**

Examples

```
s1.clearremarks
s1.label
```

The first line removes the remarks from the sspace S1, and the second line displays the label view of S1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Sspace::label` (p. 914).

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. 26) and `Sspace::spec` (p. 925).

copy	Sspace Procs

Creates a copy of the sspace.

Creates either a named or unnamed copy of the sspace.

Syntax

sspace_name.**copy**

sspace_name.**copy** *dest_name*

Examples

```
s1.copy
```

creates an unnamed copy of the sspace S1.

```
s1.copy s2
```

creates S2, a copy of the sspace S1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

display	Sspace Views

Display table, graph, or spool output in the sspace object window.

Display the contents of a table, graph, or spool in the window of the sspace object.

Syntax

sspace_name.**display** *object_name*

Examples

```
sspace1.display tab1
```

Display the contents of the table TAB1 in the window of the object SSPACE1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

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 123 of *User's Guide I* for a discussion of labels and display names.

See also `Sspace::label` (p. 914).

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. 915) and `Sspace::sspace` (p. 926).

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 wildcard 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 forecast-

ing 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

i = *arg* (*default* = "o")	State initialization options: "*o*" (one-step), "*e*" (EViews computed), "*u*" (user-specified), "*s*" (smoothed).
m = *arg* (*default* = "d")	Basic forecasting method: "*n*" (*n*-step ahead forecasting), "*s*" (smoothed forecasting), "*d*" (dynamic forecasting.
mprior = *vector_name*	Name of state initialization (use if option "*i* = *u*" is specified).
n = *arg* (*default* = 1)	Number of *n*-step forecast periods (only relevant if *n*-step forecasting is specified using the *method* option).
vprior = *sym_name*	Name of state covariance initialization (use if option "*i* = *u*" is specified).
prompt	Force the dialog to appear from within a program.
p	Print view.

Examples

The following command performs *n*-step forecasting of the signals and states from a sspace object:

```
ss1.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

State space forecasting is described in Chapter 51. "State Space Models and the Kalman Filter," on page 1151 of *User's Guide II*. For additional discussion of wildcards, see Appendix A. "Wildcards," on page 1227 of the *Command and Programming Reference*.

See also `Sspace::makemodel` (p. 917).

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. 916).

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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User's Guide I* for a discussion of labels.

See also `Sspace::displayname` (p. 911).

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

```
ss1.makeendog grp_v1
```

creates a group named GRP_V1 that contains the endogenous series in SS1.

Cross-references

See also `Sspace::endog` (p. 912) and `Model::makegroup` (p. 627).

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

```
ss1.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 51. "State Space Models and the Kalman Filter," on page 1151 of *User's Guide II* for details on state space models.

See also `Sspace::makesignals` (p. 918) and `Sspace::makestates` (p. 919).

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

n = *arg*	Name of group object to contain the series.

Examples

```
ss1.grads(n=out)
```

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

```
ss1.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. 913).

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 52. "Models," on page 1177 of *User's Guide II* for a discussion of specifying and solving models in EViews.

See also `Sspace::append` (p. 907), `Sspace::makefilter` (p. 916), and `Model::solve` (p. 640).

makeresids	Sspace Procs

`makeresids` is no longer supported for the sspace object—see `Sspace::makesignals` (p. 918) for more general 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 `makesignal` 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

t = *output_type* *(default* = "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").
n = *group_name*	Name of group to hold newly created series.
prompt	Force the dialog to appear from within a program.

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 51. "State Space Models and the Kalman Filter," on page 1151 of *User's Guide II* for details on state space models. For additional discussion of wildcards, see Appendix A. "Wildcards," on page 1227 of the *Command and Programming Reference*.

See also `Sspace::forecast` (p. 912), `Sspace::makefilter` (p. 916), and `Sspace::makestates` (p. 919).

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

t = *output_type* *(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").
n = *group_name*	Name of group to hold newly created series.
prompt	Force the dialog to appear from within a program.

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 51. "State Space Models and the Kalman Filter," on page 1151 of *User's Guide II* for details on state space models. For additional discussion of wildcards, see Appendix A. "Wildcards," on page 1227 of the *Command and Programming Reference*.

See also `Sspace::forecast` (p. 912), `Sspace::makefilter` (p. 916) and `Sspace::makesignals` (p. 918).

ml	Sspace Method

Maximum likelihood estimation of state space models.

Syntax

sspace_name.**ml***(options)*

Options

optmethod = *arg*	Optimization method: "bfgs" (BFGS); "newton" (Newton-Raphson), "opg" or "bhhh" (OPG or BHHH), "legacy" (EViews legacy). BFGS is the default method.
optstep = *arg*	Step method: "marquardt" (Marquardt); "dogleg" (Dog-leg); "linesearch" (Line search). Marquardt is the default method.
cov = *arg*	Covariance method: "ordinary" (default method based on inverse of the estimated information matrix), "huber" or "white" (Huber-White sandwich methods).,
covinfo = *arg*	Information matrix method: "opg" (OPG); "hessian" (observed Hessian). (Applicable when non-legacy "optmethod = ".)
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.
prompt	Force the dialog to appear from within a program.
p	Print basic estimation results.

Examples

```
bvar.ml
```

estimates the sspace object BVAR by maximum likelihood.

Cross-references

See Chapter 51. "State Space Models and the Kalman Filter," on page 1151 of *User's Guide II* for a discussion of user specified state space models.

olepush — Sspace Procs

Push updates to OLE linked objects in open applications.

Syntax

sspace_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

output — Sspace Views

Display estimation output.

`output` changes the default object view to display the estimation output (equivalent to using `Sspace::results` (p. 924)).

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. 924).

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. 922) and `Sspace::makeresids` (p. 918).

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. 922) and `Sspace::makeresids` (p. 918).

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 51. “State Space Models and the Kalman Filter,” on page 1151 of *User’s Guide II* for a discussion of state space models.

See also `Sspace::makeresids` (p. 918).

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 51. “State Space Models and the Kalman Filter,” on page 1151 of *User’s Guide II* for a discussion of state space models.

setattr	Sspace Procs

Set the object attribute.

Syntax

sspace_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the “revised” attribute in the object A to the string “never”, and extracts the attribute into the string object S.

Cross-references

See “Adding Custom Attributes in the Label View” on page 123 and “Adding Your Own Label Attributes” on page 70 of *User’s Guide I*.

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

t = *output_type* *(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).
prompt	Force the dialog to appear from within a program.

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 51. "State Space Models and the Kalman Filter," on page 1151 of *User's Guide II* for a discussion of state space models.

See also `Sspace::stategraphs` (p. 927), `Sspace::makesignals` (p. 918) and `Sspace::makestates` (p. 919).

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. 907).

See "Specifying a State Space Model in EViews" on page 1156 of *User's Guide II* for a discussion of specification syntax.

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 51. "State Space Models and the Kalman Filter," on page 1151 of *User's Guide II* for a discussion of state space models.

Sspace::append (p. 907) may be used to add lines to an existing sspace object. See also Sspace::ml (p. 920) 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

```
ss1.statefinal(c)
```

displays a view containing the final state covariances (the one-step ahead covariances for the first out-of-(estimation) sample period.

Cross-references

See Chapter 51. "State Space Models and the Kalman Filter," on page 1151 of *User's Guide II* for a discussion of state space models.

See also `Sspace::stateinit` (p. 928).

stategraphs	Sspace Views

Display graphs of a set of state series computed using the Kalman filter.

Syntax

sspace_name.**stategraph***(options)*

Options

t = *output_type* (*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).
prompt	Force the dialog to appear from within a program.

Other options

p	Print the view.

Examples

```
ss1.stategraphs(t=filt)
```

displays a graph view containing the filtered state values.

Cross-references

See Chapter 51. "State Space Models and the Kalman Filter," on page 1151 of *User's Guide II* for a discussion of state space models.

See also `Sspace::signalgraphs` (p. 925), `Sspace::makesignals` (p. 918) and `Sspace::makestates` (p. 919).

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

c	Display the covariance matrix.
p	Print the view.

Examples

```
ss1.stateinit
```

displays a view containing the initial state values (the one-step ahead predictions for the first period).

Cross-references

See Chapter 51. "State Space Models and the Kalman Filter," on page 1151 of *User's Guide II* for a discussion of state space models.

See also `Sspace::statefinal` (p. 927).

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

```
ss1.structure
```

displays a system transition matrices.

```
ss1.structure 1993q4
```

displays the transition matrices evaluated at 1993Q4.

Cross-references

See Chapter 51. "State Space Models and the Kalman Filter," on page 1151 of *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. 26).

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 210 of *User's Guide II* for a discussion of Wald tests.

See also `Sspace::cellipse` (p. 907).

Spool

Spool object. Container for output objects.

Spool Declaration

spool ..... create spool object (p. 954).

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

display ..... display the contents of the spool (p. 938).
label..... label information for the spool object (p. 944).

Spool Procs

addfolder..... adds an object folder to a spool (p. 934).
append ..... append objects to a spool (p. 935).
clearhist ..... clear the contents of the history attribute (p. 935).
clearremarks ..... clear the contents of the remarks attribute (p. 936).
comment ..... assign a comment to an object in a spool (p. 937).
copy..... creates a copy of the spool (p. 937).
displayname ..... assign a display name to an object in a spool (p. 938).
extract..... extract a copy of an object in a spool (p. 939).
flatten ..... remove tree hierarchy from the spool or specified embedded spool (p. 939).
graphmode..... set the display mode for graphs in the spool (p. 940).
hide ..... hides objects of specified type (p. 941).
horizindent ..... sets the horizontal indentation for the spool (p. 941).
insert ..... insert objects into a spool (p. 942).
leftmargin ..... sets the left margin of the spool or a specified embedded spool (p. 941).
move..... move an object in the spool (p. 946).
name..... rename an object in a spool (p. 947).
olepush ..... push updates to OLE linked objects in open applications (p. 947).
options..... set display options for a spool (p. 948).
print..... print an object in a spool (p. 949).
remove..... remove objects from a spool (p. 949).

save .....save spool object to disk as an a tab-delimited ASCII text, CSV, RTF or LaTeX file (p. 950).
setattr .....set the value of an object attribute (p. 951).
setfont.....set the font for title and comments (p. 952).
setzoom .....set the zoom level of the spool default display view (p. 953).
show.....displays objects of specified type (p. 953).
tablemode .....set the display mode for tables and text objects in the spool (p. 954).
title.....assign or change the title of a spool (p. 955).
topmargin .....sets the top margin of the spool or a specified embedded spool (p. 955).
vertindent .....sets the vertical indentation for the spool (p. 956).
vertspacing .....sets the amount of vertical spacing between objects in the spool (p. 957).
width.....change or reset the width of an object in the spool (p. 957).

Spool Data Members

String Values

@attr("*arg*").....string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@description.....string containing the Spool's description (if available).
@detailedtype.....string with the object type: "SPOOL".
@displayname .....string containing the Spool's display name. If the Spool has no display name set, the name is returned.
@name.....string containing the Spool's name.
@objname(i).....string containing name of the *i*-th object in the spool.
@objtype(i) .....string containing type of the *i*-th object in the spool ("graph", "table", "text", "spool").
@remarks.....string containing the Spool's remarks (if available).
@type.....string with the object type: "SPOOL".
@updatetime .....string representation of the time and date at which the Spool was last updated.

Scalar Values

@count.....number of base objects in the spool.
@totalcount .....number of objects in a flattened version of the 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.

addfolder	Spool Procs

Adds an object folder to a spool.

Syntax

spool_name.**addfolder***(options) folder_name*

Should an object or folder with the same name as *folder_name* already exist, the next available number will be appended to the *folder_name*.

Note: folders cannot be added to spool objects that are nested inside other spool objects.

Options

The options below are used to specify a point of insertion for the folder being added. If a point of insertion is not specified, the folder is appended to the bottom of the spool.

loc = *arg*	*arg* 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 *arg*, as specified by the offset option below.
offset = *arg*, (default = "before")	*arg* indicates where the object should be inserted relative to the object specified in the "loc = " option above. *arg* may be "before" (default) or "after".

Examples

```
spool01.addfolder tables
```

adds a folder named TABLES to the end of the SPOOL01 object.

Given that the object GR1 already exists in SPOOL1,

```
spool01.addfolder(loc=gr1) output
```

inserts a folder named OUTPUT in the current location of GR1.

Cross-references

See `Spool::move` (p. 946) for adding and removing objects to and from a folder. For additional discussion of spools see Chapter 18. "Spool Objects," on page 953 in *User's Guide I*.

append	Spool Procs

Append objects to a spool.

Syntax

spool_name.**append***(options) 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.

Options

name = *arg*	Set the names of the objects in the spool. *arg* is a space delimited list of new names for the list of objects being added to the spool. If this option is omitted, the names will be UNTITLED01, UNTITLED02, *etc.*
mode = *overwrite*	Will remove any existing objects with the same name when used in conjunction with the name = *arg* option.

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
```

To replace a preexisting object X in SPOOL01 with the line graph view of series SER1:

```
spool01.append(name=X, mode=overwrite) ser1.line
```

Cross-references

For additional discussion of spools see Chapter 18. "Spool Objects," on page 953 in *User's Guide I*. See also `Spool::insert` (p. 942) and `Spool::remove` (p. 949).

clearhist	Spool Procs

Clear the contents of the history attribute for spool objects.

Removes the spool's history attribute, as shown in the label view of the spool.

Syntax

spool_name.**clearhist**

Examples

```
s1.clearhist
s1.label
```

The first line removes the history from the spool S1, and the second line displays the label view of S1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Spool::label` (p. 944).

clearremarks	Spool Procs

Clear the contents of the remarks attribute.

Removes the spool's remarks attribute, as shown in the label view of the spool.

Syntax

spool_name.**clearremarks**

Examples

```
s1.clearremarks
s1.label
```

The first line removes the remarks from the spool S1, and the second line displays the label view of S1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Spool::label` (p. 944).

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 123 of *User's Guide I* for a discussion of labels and display names.

See also `Spool::label` (p. 944).

copy	Spool Procs

Creates a copy of the spool.

Creates either a named or unnamed copy of the spool.

Syntax

spool_name.**copy**

spool_name.**copy** *dest_name*

Examples

```
s1.copy
```

creates an unnamed copy of the spool S1.

```
s1.copy s2
```

creates S2, a copy of the spool S1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

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 Chapter 18. "Spool Objects," on page 953 in *User's Guide I*.

displayname	Spool Procs

Assign a display name to an object in the spool.

Syntax

spool_name.**displayname(options)** *object_arg new_name*

where *new_name* specifies the display name for the object in *object_arg*, and *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*.

Options

d	Default - sets the display name for the spool object. NOTE - *object_arg* is not applicable in this case.

Examples

```
spool01.displayname state/tab1 "Unemployment Rate"
```

assigns the “Unemployment Rate” displayname to the object TAB1, which is a child of the STATE spool.

```
spool01.displayname(d) "Hours worked"
```

attaches a display name “Hours Worked” to the spool object SPOOL1.

Cross-references

See “Labeling Objects” on page 123 of *User’s Guide I* for a discussion of labels and display names. See also `Spool::label` (p. 944).

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 Chapter 18. “Spool Objects,” on page 953 in *User’s Guide I*.

See also `Spool::print` (p. 949), `Spool::insert` (p. 942) and `Spool::remove` (p. 949).

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 MYSPOOL1.

Cross-references

For additional discussion of spools see Chapter 18. “Spool Objects,” on page 953 in *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 Chapter 18. “Spool Objects,” on page 953 in *User’s Guide I*.

See also `Spool::tablemode` (p. 954).

hide	Spool Procs

Hides objects of specified type.

Syntax

spool_name.hide(object_list)

where object_list is a list of object types that will be made hidden in addition to what is already hidden. Object_list may be one or more of the following values: “graphs,” “tables,” and “text.”

Examples

```
spool01.hide(graphs, text)
```

hides all the graph and text objects in the spool object SPOOL01.

Cross-references

For additional discussion of spools see Chapter 18. “Spool Objects,” on page 953 in *User’s Guide I*.

horizindent	Spool Procs

Change the horizontal indentation size for objects in the spool.

Syntax

spool_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 Chapter 18. "Spool Objects," on page 953 in *User's Guide I*.

See also `Spool::leftmargin` (p. 945), `Spool::topmargin` (p. 955), `Spool::vertspacing` (p. 957).

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

loc = *arg*	*arg* 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 *arg*, 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 = *arg*, *(default* = "before"*)*	*arg* indicates that the object should be inserted relative to the object specified in the "loc = " option above. *arg* may be "before" or "after" *(default* = "before"*)*. In addition, if the location specified by the "loc = " option corresponds to a spool object, *arg* may be "first" or "last", where the object will be inserted as the first or last object in the spool object specified *(default* = "last"*)*.
name = *arg*	Set the names of the objects in the spool. *arg* is a space delimited list of new names for the list of objects being added to the spool. If this UNTITLED01, UNTITLED02, *etc.*
mode = *over-write*	Will remove any existing objects with the same name when used in conjunction with the name = *arg* option.

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 SPOOL1.

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 SPOOL1, which in turn contains a graph G2:

```
spool01.insert(loc=sp1/g2) tab1
```

inserts TAB1 before graph G2 in spool SPOOL1, and moves the remaining objects down.

To replace a preexisting object X in SPOOL01 with the line graph view of series SER1:

```
spool01.insert(name=X, mode=overwrite) ser1.line
```

Cross-references

For additional discussion of spools see Chapter 18. "Spool Objects," on page 953 in *User's Guide I*.

See also `Spool::append` (p. 935) and `Spool::remove` (p. 949).

label	Spool Views | 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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User's Guide I* for a discussion of labels.

See also `Spool::displayname` (p. 938).

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

obj = *arg*	where *arg* 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 Chapter 18. "Spool Objects," on page 953 in *User's Guide I*.

See also `Spool::horizindent` (p. 941), `Spool::topmargin` (p. 955), and `Spool::vertindent` (p. 956).

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 = *arg*	*arg* 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 *arg*, 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 = *arg*	*arg* indicates that the object should be inserted relative to the object specified in the "loc = " option above. *arg* may be "before" or "after" *(default* = "before"*)*. In addition, if the location specified by the "loc = " option corresponds to a spool object, *arg* may be "first" or "last", where the object will be inserted as the first or last object in the spool object specified *(default* = "last"*)*.

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 Chapter 18. "Spool Objects," on page 953 in *User's Guide I*.

See also `Spool::insert` (p. 942).

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. 938) command.

Examples

```
spool01.name untitled01 tab1
```

renames the object UNTITLED01 to TAB1.

Cross-references

For additional discussion of spools see Chapter 18. "Spool Objects," on page 953 in *User's Guide I*.

See also `Spool::displayname` (p. 938).

olepush	Spool Procs

Push updates to OLE linked objects in open applications.

Syntax

spool_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

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 Chapter 18. "Spool Objects," on page 953 in *User's Guide I*.

See also `Spool::name` (p. 947), `Spool::displayname` (p. 938) and `Spool::label` (p. 944).

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 Chapter 18. “Spool Objects,” on page 953 in *User’s Guide I*.

See also `print` (p. 566) and `Spool::extract` (p. 939).

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 Chapter 18. “Spool Objects,” on page 953 in *User’s Guide I*.

See also `Spool::append` (p. 935) and `Spool::insert` (p. 942).

save	Spool Procs

Save spool object to disk as a tab-delimited ASCII text, RTF, CSV, PDF, LaTeX, or MD file.

Syntax

spool_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.

The MD (Markdown) setting uses very basic syntax and should be usable in most editors.

Options

t = *file_type* (*default* = "txt")	Specifies the file type, where *file_type* may be: "rtf" (Rich-text format), "txt" (tab-delimited text), "csv" (comma-separated values format, CSV), "pdf" (Portable Document Format, PDF), "tex" (LaTeX), or "md" (Markdown). Files will be saved with the ".rtf", ".txt", ".csv", "pdf", "tex", or "md" extensions, respectively. If you specify a text or CSV file, graphs in the spool will not be written to the file.
title	Include object titles.
comment	Include object comments.
prompt	Force the dialog to appear from within a program.

PDF Options

mode = *arg*	Multiple object handling: "i" (each object on individual page), "c" (continuous), or "f" (fit to page)
landscape	Save in landscape mode (the default is to save in portrait mode).
size = *arg* (*default* = "letter")	Page size: "letter", "legal", "a4", and "custom".
width = *number* (*default* = 8.5)	Page width in inches if "size = custom".
height = *number* (*default* = 11)	Page height in inches if "size = custom".

leftmargin = *number (default* = 0.5*)*	Left margin width in inches.
rightmargin = *number (default* = 0.5*)*	Right margin width in inches.
topmargin = *number (default* = 1*)*	Top margin width in inches.
bottommargin = *number (default* = 1*)*	Bottom margin width in inches.

LaTeX Options

texspec / -texspec	[Include / Do not include] the full LaTeX documentation specification in the LaTeX output. The default behavior is taken from the global default settings.

Examples

```
spool01.save(t=rtf, title) c:\temp\spool01
```

saves SPOOL01 to an RTF file named "spool01.rtf" in the "C:\TEMP" directory, and precedes each object in the spool with its title.

```
spool01.save(comment) spool01.txt
```

saves SPOOL01 to a text file named "spool01.txt" in the current directory, and precedes each object in the spool with its associated comment if one exists.

Cross-references

For additional discussion see "Saving a Spool," on page 971 in *User's Guide I*.

setattr	Spool Procs

Set the object attribute.

Syntax

spool_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the “revised” attribute in the object A to the string “never”, and extracts the attribute into the string object S.

Cross-references

See “Adding Custom Attributes in the Label View” on page 123 and “Adding Your Own Label Attributes” on page 70 of *User’s Guide I*.

setfont	Spool Procs

Set the font for title and comments.

Syntax

spool_name.**setfont** *font_args*

where *font_args* may include one or more of the options listed below.

Options

[face], [pt], [+/-b], [+/-i], [+/-u], [+/-s]	Set characteristics of the font for the spool titles and comments. The font name (face), size (pt), and characteristics are all optional. *Face* should be a valid font name, enclosed in double quotes. *pt* 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.

Examples

```
spool1.setfont “Times New Roman” +i
```

sets the title and comment font to Times New Roman italic.

```
spool1.setfont 8pt
```

changes the font to 8 point.

```
spool1.setfont +b -i
```

removes the italic, and adds boldface.

```
spool1.setfont -s +u 14pt
```

changes the point size to 14, removes strikethrough, and adds underscoring.

```
spool1.setfont “Batang” 14pt +u
```

changes the typeface to Batang, and adds underscoring.

Cross-references

For additional discussion of spools see Chapter 18. “Spool Objects,” on page 953 in *User’s Guide I*.

setzoom	Spool Procs

Sets the zoom level of the spool default display view.

Syntax

spool_name.**setzoom** *zoom_arg*

where *zoom_arg* specifies the zoom level in percent. The *zoom_arg* must be one of the following values: 25, 50, 75, 100, 125, 150, 200.

Examples

```
spool1.setzoom 150
```

sets the zoom level of the default spool view to 150% for spool SPOOL1.

Cross-references

For additional discussion of spools see Chapter 18. “Spool Objects,” on page 953 in *User’s Guide I*.

show	Spool Procs

Displays objects of specified type.

Syntax

spool_name.**show(object_list)**

where object_list is a list of object types that will be made visible in addition to what is already visible. Object_list may be one or more of the following values: “graphs,” “tables,” and “text.”

Examples

```
spool01.show(tables, text)
```

makes visible all the table and text objects in the spool object SPOOL01.

Cross-references

For additional discussion of spools see Chapter 18. “Spool Objects,” on page 953 in *User’s Guide I*.

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 Chapter 18. “Spool Objects,” on page 953 in *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 Chapter 18. “Spool Objects,” on page 953 in *User’s Guide I*.

See also `Spool::graphmode` (p. 940).

title	Spool Procs

Assign or change the title of a spool.

Syntax

spool_name.**title** *title_arg*

where *title_arg* is a case sensitive string which may contain spaces and carriage returns.

Examples

```
spool1.title Estimated Models\nfor 2017
```

sets a two line title for SPOOL1 where the first line is "Estimated Models" and the second line is "for 2017."

```
spool1.title
```

clears the SPOOL1 title.

Cross-references

See also `Spool::displayname` (p. 938) and `Spool::label` (p. 944).

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 = *arg*	where *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 Chapter 18. “Spool Objects,” on page 953 in *User’s Guide I*.

See also `Spool::vertindent` (p. 956), `Spool::vertspacing` (p. 957), and `Spool::horizindent` (p. 941).

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 Chapter 18. “Spool Objects,” on page 953 in *User’s Guide I*.

See also `Spool::topmargin` (p. 955), `Spool::vertspacing` (p. 957), and `Spool::horizindent` (p. 941).

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_arg*	where *object_arg* 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 Chapter 18. “Spool Objects,” on page 953 in *User’s Guide I*.

See also `Spool::vertindent` (p. 956), and `Spool::topmargin` (p. 955).

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 `Spool::graphmode` (p. 940) and `Spool::tablemode` (p. 954), 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 = *arg*	where *arg* 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 Chapter 18. “Spool Objects,” on page 953 in *User’s Guide I*.

String

String object. String objects may be used in standard EViews expressions in place of string literals.

String Declaration

string.....declare string object (p. 965).

To declare a string object, use the keyword `string`, followed by a name, an " = " sign and a text string.

String Views

display.....display table, graph, or spool in object window (p. 961).
label.....label view (p. 962).
list.....list view display of the string (p. 963).
sheet.....display the string (p. 964).

String Procs

clearhist.....clear the contents of the history attribute (p. 960).
clearremarks.....clear the contents of the remarks attribute (p. 960).
copy.....creates a copy of the string (p. 961).
displayname.....set display name (p. 962).
olepush.....push updates to OLE linked objects in open applications (p. 964).
setattr.....set the value of an object attribute (p. 964).

String Data Members

String values

@attr("*arg*").....string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@description.....string containing the String object's description (if available).
@detailedtype.....string with the object type: "STRING".
@displayname.....string containing the String object's display name. If the String has no display name set, the name is returned.
@name.....string containing the String object's name.
@remarks.....string containing the String object's remarks (if available).
@type.....string with the object type: "STRING".
@updatetime.....string representation of the time and date at which the String was last updated.

String Examples

You can declare a string and examine its contents:

```
string st="Hello world"
```

```
show st
```

String Entries

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

clearhist	String Procs

Clear the contents of the history attribute.

Removes the string's history attribute, as shown in the label view of the string.

Syntax

string_name.**clearhist**

Examples

```
s1.clearhist
s1.label
```

The first line removes the history from the string S1, and the second line displays the label view of S1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `String::label` (p. 962).

clearremarks	String Procs

Clear the contents of the remarks attribute.

Removes the string's remarks attribute, as shown in the label view of the string.

Syntax

string_name.**clearremarks**

Examples

```
s1.clearremarks
s1.label
```

The first line removes the remarks from the string S1, and the second line displays the label view of S1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `String::label` (p. 962).

copy	String Procs

Creates a copy of the string.

Creates either a named or unnamed copy of the string.

Syntax

string_name.**copy**

string_name.**copy** *dest_name*

Examples

```
s1.copy
```

creates an unnamed copy of the string S1.

```
s1.copy s2
```

creates S2, a copy of the string S1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

display	String Views

Display table, graph, or spool output in the string object window.

Display the contents of a table, graph, or spool in the window of the string object.

Syntax

string_name.**display** *object_name*

Examples

```
string1.display tab1
```

Display the contents of the table TAB1 in the window of the object STRING1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

displayname	String Views

Display name for the string objects.

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

Syntax

string_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

```
str1.displayname Patagonian Toothfish Name
str1.label
```

The first line attaches a display name "Patagonian Toothfish Name" to the string object STR1, and the second line displays the label view of STR1, including its display name.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names.

See also `String::label` (p. 962).

label	String Views

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

Syntax

string_name.**label**

string_name.**label***(options) text*

Options

To modify the label, you should specify one of the following options along with optional text. If there is no text provided, the specified field will be cleared:

c	Clears all text fields in the label.
d	Sets the description field to *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of the string S1 with "Name of Dependent Variable from EQ3":

```
s1.label(r)
s1.label(r) Name of Dependent Variable EQ3
```

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels.

list

String Views

List view of a string object.

Syntax

string_name.**list***(options)*

Options

p	Print the list view.

Examples

```
s01.list
```

displays the text of the string in S01 in list format with one word per line.

Cross-references

See `String::sheet` (p. 964) for an alternative formatted view of the string contents.

olepush	String Procs

Push updates to OLE linked objects in open applications.

Syntax

string_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

setattr	String Procs

Set the object attribute.

Syntax

string_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the "revised" attribute in the object A to the string "never", and extracts the attribute into the string object S.

Cross-references

See "Adding Custom Attributes in the Label View" on page 123 and "Adding Your Own Label Attributes" on page 70 of *User's Guide I*.

sheet	String Views

Spreadsheet view of a string object.

Syntax

string_name.**string***(options)*

Options

p	Print the spreadsheet view.

Examples

```
s01.string
```

displays the text of the string in S01.

Cross-references

See `String::list` (p. 963) for an alternative formatted view of the string contents.

string — String Declaration

Declare a string object.

The `string` command declares a string object and optionally assigns text.

Syntax

string *string_name[= assignment]*

The `string` 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 null.

Examples

```
string alpha
```

declares a string object named ALPHA containing no text.

You may also create a string that includes quotes:

```
string lunch = "Apple Tuna Cookie"
string dinner = """Chicken Marsala"" ""Beef Stew"" Hamburger"
```

creates the string objects LUNCH and DINNER, each containing the corresponding string literal. We have used the double quote character in the DINNER string as an escape character for double quotes.

Cross-references

See "Strings" on page 85 and "String Objects" on page 101 of the *Command and Programming Reference* for a discussion of strings and string objects.

Svector

String vector object.

Svector Declaration

svector ..... declare svector object (p. 986).

To declare an svector object, use the keyword `svector`, followed by a name.

Svector Views

display ..... display table, graph, or spool in object window (p. 970).
freq ..... one-way tabulation (p. 975).
label ..... label view (p. 982).
sheet ..... spreadsheet view of the scalar (p. 985).

Svector Procs

clearcollabels ..... clear the column labels in a svector object (p. 967).
clearhist ..... clear the contents of the history attribute (p. 968).
clearremarks ..... clear the contents of the remarks attribute (p. 968).
clearrowlabels ..... clear the row labels in a svector object (p. 969).
copy ..... creates a copy of the svector (p. 969).
displayname ..... set display name (p. 970).
export ..... export svector as Excel 2007 XLSX, CSV, tab-delimited ASCII text, RTF, HTML, Enhanced Metafile, PDF, TEX, or MD file on disk (p. 971).
fill ..... fill the svector with the specified values (p. 974).
import ..... imports data from a foreign file into the svector object (p. 976).
olepush ..... push updates to OLE linked objects in open applications (p. 983).
resize ..... resize the svector object (p. 983).
setattr ..... set the value of an object attribute (p. 984).
setcollabels ..... set the column label in a svector object (p. 984).
setrowlabels ..... set the row labels in a svector object (p. 985).
showlabels ..... displays the custom row and column labels of an svector spreadsheet (p. 986).

Svector Data Members

String values

@attr("*arg*") ..... string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@collabels ..... string containing the column label of the svector.
@description ..... string containing the svector object's description (if available).

@detailedtype .....string with the object type: “SVECTOR”.
@displayname .....string containing the svector object’s display name. If the svector has no display name set, the name is returned.
@name .....string containing the svector object’s name.
@remarks .....string containing the svector object’s remarks (if available).
@rowlabels.....string containing the row labels of the svector.
@type.....string with the object type: “SVECTOR”.
@updatetime .....string representation of the time and date at which the svector was last updated.

Scalar values

@rows.....number of rows in the svector.

Svector values

@droprow(*arg*) .....Returns the svector with rows defined by *arg* removed. *arg* may be an integer, vector of integers, string, or svector of strings. Integers correspond to row numbers so that, for example, *arg* = 2 specifies the second row. Strings correspond to row labels so that *arg* = "2" specifies the first row labeled “2”.
@irow(*arg*) .....Returns the indices for the rows defined by *arg* where *arg* is a string or svector of strings. The strings correspond to row labels so that *arg* = "2" specifies the first row labeled “2”.
@row(*arg*) .....Returns the svector with rows defined by *arg*. *arg* may be an integer, vector of integers, string, or svector of strings. Integers correspond to row numbers so that, for example, *arg* = 2 specifies the second row. Strings correspond to row labels so that *arg* = "2" specifies the first row labeled “2”.

Svector Entries

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

clearcollabels	Svector Procs

Clear the column label in a svector object.

Syntax

svector_name.**clearcollabels**

Examples

```
svec1.clearcollabels
```

clears the custom column label from the svector SVEC1.

Cross-references

See also `Svector::clearrowlabels` (p. 969).

clearhist	Svector Procs

Clear the contents of the history attribute.

Removes the svector history attribute, as shown in the label view of the svector.

Syntax

svector_name.**clearhist**

Examples

```
s1.clearhist
s1.label
```

The first line removes the history from the svector S1, and the second line displays the label view of S1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names.

See also `Svector::label` (p. 982).

clearremarks	Svector Procs

Clear the contents of the remarks attribute.

Removes the svector's remarks attribute, as shown in the label view of the svector.

Syntax

svector_name.**clearremarks**

Examples

```
s1.clearremarks
s1.label
```

The first line removes the remarks from the svector S1, and the second line displays the label view of S1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names.

See also `Svector::label` (p. 982).

clearrowlabels	Svector Procs

Clear the row labels in a vector object.

Syntax

svector_name.**clearrowlabels**

Examples

```
svec1.clearrowlabels
```

clears the custom row labels from the svector SVEC1.

Cross-references

See also `Svector::clearcollabels` (p. 967).

copy	Svector Procs

Creates a copy of the svector.

Creates either a named or unnamed copy of the svector.

Syntax

svector_name.**copy**

svector_name.**copy** *dest_name*

Examples

```
s1.copy
```

creates an unnamed copy of the svector S1.

```
s1.copy s2
```

creates S2, a copy of the svector S1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

display	Svector Views

Display table, graph, or spool output in the svector object window.

Display the contents of a table, graph, or spool in the window of the svector object.

Syntax

svector_name.**display** *object_name*

Examples

```
svector1.display tab1
```

Display the contents of the table TAB1 in the window of the object SVECTOR1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

displayname	Svector Views

Display name for the svector objects.

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

Syntax

svector_name.**displayname** *display_name*

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

Examples

```
svec1.displayname List of Names
svec1.label
```

The first line attaches a display name "List of Names" to the svector object SVEC1, and the second line displays the label view of SVEC1, including its display name.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names.

See also `Svector::label` (p. 982).

export	Svector Procs

Export svector to disk as an Excel 2007 XLSX, CSV, tab-delimited ASCII text, RTF, HTML, Enhanced Metafile, LaTeX, PDF, or Markdown file.

Syntax

svector_name.**export***(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.

The base syntax for writing Excel 2007 files is:

svector_name.**export**(*options*) *[path\]file_name [table_description]*

where the *table_description* may contain:

- "range = *arg*", where arg is top left cell of the destination 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.

Options

t = *file_type* (*default* = "csv")	Specifies the file type, where *file_type* may be one of: "excelxml" (Excel 2007 (xml)),"csv" (CSV - comma-separated), "rtf" (Rich-text format), "txt" (tab-delimited text), "html" (HTML - Hypertext Markup Language), "emf" (Enhanced Metafile), "pdf" (PDF - Portable Document Format), "tex" (LaTeX), or "md" (Markdown). Files will be saved with the ".xlsx", ".csv", ".rtf", ".txt", ".htm", ".emf", ".pdf", ".tex", or ".md" extensions, respectively.
s = *arg*	Scale size, where *arg* is from 5 to 200, representing the percentage of the original table size (only valid for HTML or RTF files).
n = *string*	Replace all cells that contain NA values with the specified string. "NA" is the default.
h / -h	Include(/do not include) column and row headers. The default is to not include the headers
prompt	Force the dialog to appear from within a program.

PDF Options

landscape	Save in landscape mode (the default is to save in portrait mode).
size = *arg* (*default* = "letter")	Page size: "letter", "legal", "a4", and "custom".
width = *number* (*default* = 8.5)	Page width in inches if "size = custom".
height = *number* (*default* = 11)	Page height in inches if "size = custom".
leftmargin = *number* (*default* = 0.5)	Left margin width in inches.
rightmargin = *number* (*default* = 0.5)	Right margin width in inches.
topmargin = *number* (*default* = 1)	Top margin width in inches.
bottommargin = *number* (*default* = 1)	Bottom margin width in inches.

LaTeX Options

texspec / -texspec	[Include / Do not include] the full LaTeX documentation specification in the LaTeX output. The default behavior is taken from the global default settings.

Excel Options

mode = *arg*	Specify whether to create a new file, overwrite an existing file, or update an existing file. arg may be "create" (create new file only; error on attempt to overwrite) or "update" (update an existing file, only overwriting the area specified by the range = *table_description*). If the "mode = " option is not used, EViews will create a new file, unless the file already exists in which case it will overwrite it. Note that the "mode = update" option is only available for Excel in 1) Excel versions through 2003, if Excel is installed, and 2) Excel 2007 (xml). Note: Excel does not need to be installed for Excel 2007 writing.

Excel 2007 Options

cellfmt = *arg*	Specify whether to use EViews, pre-existing, or remove cell formatting (colors, font, number formatting when possible, column widths and row heights) for the written range. *arg* may be "eviews" (replace current formatting in the file with the same cell formatting in EViews), "preserve" (leave current cell formatting already in the Excel file), or "clear" (remove current formatting and do not replace).
strlen = *arg* (default = 256)	Specify the maximum the number of characters written for cells containing text. Strings in cells which are longer the max, will be truncated.

Examples

The command:

```
svector1.export mysvector
```

exports data in SVECTOR1 to a CSV file named "mysvector.CSV" in the default directory.

```
svector1.export(h,t=csv, n="NaN") mysvector
```

saves the contents of SVECTOR1 along with the column and row headers to a CSV (comma separated value) file named "mysvector.CSV" and writes all NA values as "NaN".

```
svector1.export(h,t=html, s=50) mysvector
```

writes the data of SVECTOR1 along with the column and row headers to a HTML file named "mysvector.HTM" at half of the original size.

```
svector1.export(n=".", r=B) mysvector
```

exports the data in the second column to a CSV file named "mysvector.CSV", and writes all NA values as ".".

```
svector1.export(t=excelxml, cellfmt=clear, mode=update) mysvector
    range=Country!b5
```

writes the data in SVECTOR1 to the preexisting "mysvector.XLSX" Excel file to the "Country" sheet at cell B5, where all cell formatting is cleared.

Cross-references

See Svector::import (p. 976).

fill	Svector Procs

Fill a svector with the specified values.

Syntax

svector_name.**fill***(options) s1[s2 s3 …]*

Follow the keyword with a list of strings to place in the svector object. *Each value should be surrounded by double quotes if necessary, and values should separated by a space*. 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 to enable looping. If, however, you list more values than the object can hold, EViews will return an error message.

Options

l	Loop repeatedly over the list of values as many times as it takes to fill the vector.
o = *integer* *(default* = 1*)*	Fill the svector starting from the specified element. Default is the first element.

Examples

```
sv1.fill a B c
```

sets the first element of SV1 to "a", the second to "B" and the third to "c".

```
sv1.fill(o=2) a "Hello World" name
```

sets the second element of SV1 to "a", the third to "Hello World" and the fourth to "name".

```
sv1.fill(o=4, l) first "" Last
```

sets the fourth element of SV1 to “first”, the fifth to be an empty string, and the sixth value to “Last”, then repeats the same three values for the remaining rows, so that the seventh element is set to “first”, the eight element is empty, the ninth set to “Last” and so on.

```
sv1.fill(l) ""
```

clears all of the values in the svector SV1.

An alternative approach to filling the elements of an svector uses the `@sfill` function to create a new svector in a single line, as in

```
svector vs2 = @fill("a", "Hello World", "name")
```

Cross-references

See Chapter 11. “Matrix Language,” on page 279 of the *Command and Programming Reference* for a detailed discussion of vector and matrix manipulation in EViews.

freq	Svector Views

Compute frequency tables.

`freq` performs a one-way frequency tabulation.

Frequencies are computed for all of the rows in the svector. Rows with missing values (blanks) are dropped unless included by option. You may use options to control the order of the entries of the table.

Syntax

svector_name.**freq***(options)*

Options

dropna *(default)* / keepna	[Drop/Keep] NA as a category.
n, obs, count *(default)*	Display frequency counts.
nocount	Do not display frequency counts.
prompt	Force the dialog to appear from within a program.
p	Print the table.
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.
sort = *arg* *(default* = "lohi"*)*	Sort order for entries in the frequency table: high data value to low ("hilo"), low data value to high ("lohi" – *default*), high frequency to low ("freqhilo"), low frequency to high ("freqlohi").
prompt	Force the dialog to appear from within a program.
p	Print the table.

Examples

```
svec1.freq
```

tabulates each value of SVEC1 in ascending alphabetical order with counts, percentages, and cumulatives.

```
svec1.freq(keepna)
```

tabulates the values of SVEC1 including NAs.

```
svec1.freq(sort=hilo)
```

tabulates SVEC1 with the table rows ordered from values with highest frequency to lowest.

Cross-references

See "One-Way Tabulation" on page 467 of *User's Guide I* for a discussion of frequency tables

import	Svector Procs

Imports data from a foreign file into the svector object.

Syntax

svector_name.**import***([type =]) source_description import_specification*

- *source_description* should contain a description of the file from which the data is to be imported. The specification of the description is usually just the path and file name of the file, however you can also specify more precise information. See `wfopen` (p. 640) of the *Command and Programming Reference* for more details on the specification of *source_description*.
- The optional "type = " option may be used to specify a source type. For the most part, you should not need to specify a "type = " option as EViews will automatically determine the type from the filename. The following table summaries the various source formats and along with the corresponding "type = " keywords:

	Option Keywords
Excel (through 2003)	"excel"
Excel 2007 (xml)	"excelxml"
HTML	"html"
Text / ASCII	"text"

- *import_specification* can be used to provide additional information about the file to be read. The details of import_specification will depend upon the type of file being imported.

Excel Files

The syntax for reading Excel files is:

svector_name.**import(type = excel***[xml]*) *source_description [table_description] [variables_description]*

The following *table_description* elements may be used when reading Excel data:

- "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.

- "byrow", transpose the incoming data. This option allows you to read files where the series are contained in rows (one row per series) rather than columns.

The optional *variables_description* may be formed using the elements:

- "colhead = *int*", number of table rows to be treated as column headers.
- "types = ("*arg1*","*arg2*",…)", user specified data types of the series. If types are provided they will override the types automatically detected by EViews. You may use any of the following format keywords: "a" (character data), "f" (numeric data), "d" (dates), or "w" (EViews automatic detection). This option is rarely required.
- "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 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 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.

Excel Examples

```
svec_obj.import "c:\data files\data.xls"
```

loads the active sheet of DATA.XLSX into the SVEC_NAME svector object.

```
svec_obj.import "c:\data files\data.xls" range="GDP data"
```

reads the data contained in the "GDP data" sheet of "Data.XLS" into the SVEC_OBJ object.

HTML Files

The syntax for reading HTML pages is:

svector_name.**import(type = html)** *source_description [table_description] [variables_description]*

The following *table_description* elements may be used when reading an HTML file or page:

- "table = *arg*", where *arg* specifies which HTML 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.
- "byrow", transpose the incoming data. This option allows you to import files where the series are contained in rows (one row per series) rather than columns.

The optional *variables_description* may be formed using the elements:

- "colhead = *int*", number of table rows to be treated as 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.

HTML Examples

```
svec1.import "c:\data.html"
```

loads into the SVEC1 svector object the data located in the HTML file "Data.HTML" located on the C:\ drive

```
svec1.import(type=html) "http://www.tradingroom.com.au/apps/mkt/
    forex.ac" colhead=3
```

loads into an svector object called SVEC1 the data with the given URL located on the web-site site "http://www.tradingroom.com.au". The column header is set to three rows.

Text and Binary Files

The syntax for reading text or binary files is:

svector_name.**import(type** = *arg*) *source_description [table_description] [variables_description]*

If a *table_description* is not provided, EViews will attempt to read the file as a free-format text file. The following *table_description* elements may be used when reading a text or binary file:

- "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.

- “byrow”, transpose the incoming data. This option allows you to import files where the series are contained in rows (one row per series) rather than columns.
- “lastcol”, include implied last column. For lines that end with a delimiter, this option adds an additional column.

  When importing a CSV file, lines which have the delimiter as the last character (for example: ‘name,description,date,’), EViews normally determines the line to have 3 columns. With the above option, EViews will determine the line to have 4 columns. Note this is not the same as a line containing ‘name,description,date’. In this case, EViews will always determine the line to have 3 columns regardless if the option is set.

A central component of the *table_description* element is the format statement. You may specify the data format using the following table descriptors:

- Fortran Format:

  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 Format:

  rformat = "*[n1]Type[Width][.Precision]*, *[n2]Type[Width][.Precision]*, ...)"

  where optional type is “$” for string or “*#*” for number, and *n1*, *n2*, *n3*, *n4*, *etc*. are the range of columns containing the data.

- C printf/scanf Format:

  cformat = "*fmt*"

  where *fmt* follows standard C language (printf/scanf) format rules.

The optional *variables_description* may be formed using the elements:

- “colhead = *int*”, number of table rows to be treated as column headers.

- "types = ("*arg1*","*arg2*",...)", user specified data types of the series. If types are provided they will override the types automatically detected by EViews. You may use any of the following format keywords: "a" (character data), "f" (numeric data), "d" (dates), or "w" (EViews automatic detection). This option is rarely used.
- "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.

Text and Binary File Examples (.txt, .csv, etc.)

```
svec2.import c:\data.csv skip=5
```

reads "Data.CSV" into SVEC2, skipping the first 5 rows.

```
svec01.import(type=text) c:\date.txt delim=comma
```

loads the comma delimited data DATE.TXT into the SVEC01 svector object.

Cross-references

See `Svector::export` (p. 971).

label	Svector Views

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

Syntax

svector_name.**label**

svector_name.**label***(options) text*

Options

To modify the label, you should specify one of the following options along with optional text. If there is no text provided, the specified field will be cleared:

c	Clears all text fields in the label.
d	Sets the description field to *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of the string S1 with "Name of Dependent Variable from EQ3":

```
s1.label(r)
s1.label(r) Name of Dependent Variable EQ3
```

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels.

olepush — Svector Procs

Push updates to OLE linked objects in open applications.

Syntax

svector_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

resize — Svector Procs

Resize the svector object.

Syntax

svector_name.**resize** *rows*

Examples

```
svec1.resize 20
```

resizes the svector SVEC1 to 20 rows, retaining the contents of any existing elements and initializing new elements to the empty string "".

Cross-references

See also `Svector::fill` (p. 974).

setattr	Svector Procs

Set the object attribute.

Syntax

svector_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the "revised" attribute in the object A to the string "never", and extracts the attribute into the string object S.

Cross-references

See "Adding Custom Attributes in the Label View" on page 123 and "Adding Your Own Label Attributes" on page 70 of *User's Guide I*.

setcollabels	Svector Procs

Set the column label in a svector object.

Syntax

svector_name.**setcollabels** *label*

Follow the `setcollabels` command with the column label. Note that the column label should not contain spaces unless it is enclosed in quotes.

Examples

```
svec1.setcollabels MyResults
```

sets the column label to "MyResults".

Cross-references

See also `Svector::setrowlabels` (p. 985).

setrowlabels	Svector Procs

Set the row labels in a svector object.

Syntax

svector_name.**setrowlabels** *label1 label2 label3....*

Follow the `setrowlabels` command with a space delimited list of row labels. Note that each row label should not contain spaces unless it is enclosed in quotes. If you provide fewer labels than there are rows, EViews will keep the corresponding default row names ("R11", "R12", etc...).

Examples

```
svec1.setrowlabels USA UK FRANCE
```

sets the row label for the first row in svector SVEC1 to USA, the second to UK, and the third to FRANCE.

Cross-references

See also `Svector::setcollabels` (p. 984).

sheet	Svector Views

Spreadsheet view of a string vector object.

Syntax

svector_name.**sheet***(options)*

Options

p	Print the spreadsheet view.

Examples

```
s01.sheet
```

displays the spreadsheet view of S01.

showlabels	Svector Procs

Displays the custom row and column labels of an svector spreadsheet.

Syntax

svector_name.**showlabels** *mode*

where *mode* is either 0 or 1 where 0 displays the default row and column labels and 1 displays the custom row and column labels (if present).

Examples

```
s1.showlabels 1
```

displays the custom row and column labels for the S1 spreadsheet. If custom labels have not been set the default labels will be displayed.

```
s1.showlabels 0
```

displays the default row and column labels for the S1 spreadsheet.

Cross-references

See `Svector::setcollabels` (p. 984) and `Svector::setrowlabels` (p. 985).

svector	Svector Declaration

Declare a string vector object.

The `svector` command declares a string vector object.

Syntax

svector*(n) stringvector_name*

The `svector` keyword should be followed by a valid name. *n* is an optional length for the vector. If *n* is not provided, the resulting svector will be one element long.

Examples

```
svector alphavec
```

declares a string vector object named ALPHAVEC containing no text.

```
svector(20) alphavec
```

declares a 20 element svector.

Cross-references

See "Strings" on page 85 and "String Vectors" on page 102 of the *Command and Programming Reference* for a discussion of strings and string vectors.

Sym

Symmetric matrix (symmetric two-dimensional array).

Sym Declaration

sym .....declare sym object (p. 1025).

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

cor .....correlation matrix by columns (p. 995).
cov .....covariance matrix by columns (p. 998).
eigen.....eigenvalues calculation for a symmetric matrix (p. 1002).
label .....label information for the symmetric matrix (p. 1015).
sheet.....spreadsheet view of the symmetric matrix (p. 1023).
stats.....descriptive statistics by column (p. 1024).

Sym Procs

clearcollabels .....clear the column labels in a svector object (p. 993).
clearhist.....clear the contents of the history attribute (p. 993).
clearremarks .....clear the contents of the remarks attribute (p. 994).
clearrowlabels.....clear the row labels in a vector object (p. 994).
copy.....creates a copy of the sym (p. 994).
displayname.....set display name (p. 1002).
export .....save sym matrix as Excel 2007 XLSX, CSV, tab-delimited ASCII text, RTF, HTML, Enhanced Metafile, PDF, TEX, or MD file on disk (p. 1005).
fill.....fill the elements of the matrix (p. 1008).
import.....imports data from a foreign file into the sym object (p. 1009).
label .....label information for the symmetric matrix (p. 1015).
olepush.....push updates to OLE linked objects in open applications (p. 1016).
read .....(deprecated) import data from disk (p. 1017).
resize.....resize the sym object (p. 1019).
setattr .....set the value of an object attribute (p. 1019).
setcollabels .....set the column labels in a sym object (p. 1020).
setformat.....set the display format for the sym spreadsheet (p. 1020).

setindent ..... set the indentation for the sym spreadsheet (p. 1021).
setjust ..... set the horizontal justification for all cells in the spreadsheet view of the sym object (p. 1022).
setrowlabels ..... set the row labels in a sym object (p. 1022).
setwidth ..... set the column width in the sym spreadsheet (p. 1023).
showlabels ..... displays the custom row and column labels of a sym spreadsheet (p. 1024).
write ..... export data to disk (p. 1025).

Sym Graph Views

Graph creation views are discussed in detail in "Graph Creation Command Summary" on page 1267.

area..... area graph of the columns of the matrix (p. 1269).
band ..... area band graph (p. 1272).
bar ..... bar graph of each column against the row index (p. 1275).
boxplot..... boxplot graph (p. 1279).
distplot..... distribution graph (p. 1283).
dot ..... dot plot graph (p. 1290).
errbar..... error bar graph view (p. 1294).
hilo ..... high-low(-open-close) chart (p. 1296).
line ..... line graph of each column against the row index (p. 1298).
mixed..... mixed-type graph (p. 1301).
pie ..... pie chart view (p. 1304).
qqplot ..... quantile-quantile graph (p. 1306).
spike ..... spike graph (p. 1323).

Sym Data Members

String values

@attr("*arg*") ..... string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@collabels..... string containing the column labels of the sym.
@description ..... string containing the Sym object's description (if available).
@detailedtype ..... string with the object type: "SCALAR".
@displayname..... string containing the Sym object's display name. If the Sym has no display name set, the name is returned.
@name ..... string containing the Sym object's name.
@remarks ..... string containing the Sym object's remarks (if available).
@rowlabels ..... string containing the row labels of the sym.
@type ..... string with the object type: "SCALAR".

@updatetime .....string representation of the time and date at which the Sym was last updated.

Scalar values

(i,j) .....(*i,j*)-th element of the sym. Simply append "(i,j)" to the *sym* name (without a ".").

@cols .....number of columns in the sym.

@rows.....number of rows in the sym.

Matrix values

@col(*arg*) .....Returns the columns defined by *arg*. *arg* may be an integer, vector of integers, string, or svector of strings. Integers correspond to column numbers so that, for example, *arg* = 2 specifies the second column. Strings correspond to column labels so that *arg* = "2" specifies the first column labeled "2".

@diag.....vector containing the diagonal elements of the sym.

@drop(*arg*) .....Returns the sym with the rows and columns defined by *arg* removed. *arg* may be an integer, vector of integers, string, or svector of strings. Integers correspond to row and column numbers so that, for example, *arg* = 2 specifies the second row and column. Strings correspond to row and column labels so that *arg* = "2" specifies the first row and column labeled "2".

@dropboth(*arg1, arg2*)Returns the matrix with the rows defined by *arg1* and columns defined by *arg2* removed. The *args* may be integers, vectors of integers, strings, or svectors of strings. Integers correspond to row or column numbers so that, for example, *arg1* = 2 specifies the second row. Strings correspond to row or column labels so that *arg2* = "2" specifies the first column labeled "2".

@dropcol(*arg*).....Returns the matrix with the columns defined by *arg* removed. *arg* may be an integer, vector of integers, string, or svector of strings. Integers correspond to column numbers so that, for example, *arg* = 2 specifies the second column. Strings correspond to column labels so that *arg* = "2" specifies the first column labeled "2".

@droprow(*arg*) .....Returns the matrix with rows defined by *arg* removed. *arg* may be an integer, vector of integers, string, or svector of strings. Integers correspond to row numbers so that, for example, *arg* = 2 specifies the second row. Strings correspond to row labels so that *arg* = "2" specifies the first row labeled "2".

@icol(*arg*) .....Returns the indices for the columns defined by *arg* where *arg* is a string or svector of strings. The strings correspond to column labels so that *arg* = "2" specifies the first column labeled "2".

@irow(*arg*) ..... Returns the indices for the rows defined by *arg* where *arg* is a string or svector of strings. The strings correspond to row labels so that *arg* = "2" specifies the first row labeled "2".

@row(*arg*) ..... Returns the rows defined by *arg*. *arg* may be an integer, vector of integers, string, or svector of strings. Integers correspond to row numbers so that, for example, *arg* = 2 specifies the second row. Strings correspond to row labels so that *arg* = "2" specifies the first row labeled "2".

@sub(*arg*)..... Returns the sym with rows and columns defined by *arg*. *arg* may be an integer, vector of integers, string, or svector of strings. Integers correspond to row and column numbers so that, for example, *arg* = 2 specifies the second row and column. Strings correspond to row and column labels so that *arg* = "2" specifies the first row and column labeled "2".

@sub(*arg1*, *arg2*).. Returns the matrix with rows defined by *arg1* and columns defined by *arg2*. The *args* may be integers, vectors of integers, strings, or svectors of strings. Integers correspond to row or column numbers so that, for example, *arg1* = 2 specifies the second row. Strings correspond to row or column labels so that *arg2* = "2" specifies the first column labeled "2".

@t ..... Returns transpose (copy) of the sym.

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.

clearcollabels	Sym Procs

Clear the column label in a sym object.

Syntax

sym_name.**clearcollabels**

Examples

```
sym1.clearcollabels
```

clears the custom column label from the sym SYM1.

Cross-references

See also `Sym::clearrowlabels` (p. 994).

clearhist	Sym Procs

Clear the contents of the history attribute.

Removes the sym's history attribute, as shown in the label view of the sym.

Syntax

sym_name.**clearhist**

Examples

```
s1.clearhist
s1.label
```

The first line removes the history from the sym S1, and the second line displays the label view of S1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Sym::label` (p. 1015).

clearremarks	Sym Procs

Clear the contents of the remarks attribute.

Removes the sym's remarks attribute, as shown in the label view of the sym.

Syntax

sym_name.**clearremarks**

Examples

```
s1.clearremarks
s1.label
```

The first line removes the remarks from the sym S1, and the second line displays the label view of S1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Sym::label` (p. 1015).

clearrowlabels	Sym Procs

Clear the row labels in a sym object.

Syntax

sym_name.**clearrowlabels**

Examples

```
sym1.clearrowlabels
```

clears the custom row labels from the sym SYM1.

Cross-references

See also `Sym::clearcollabels` (p. 993).

copy	Sym Procs

Creates a copy of the sym.

Creates either a named or unnamed copy of the sym.

Syntax

sym_name.**copy**

sym_name.**copy** *dest_name*

Examples

```
s1.copy
```

creates an unnamed copy of the sym S1.

```
s1.copy s2
```

creates S2, a copy of the sym S1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

cor	Sym Views

Compute covariances, correlations, and other measures of association for the columns in a matrix.

You may compute measures related to Pearson product-moment (ordinary) covariances and correlations, Spearman rank covariances, or Kendall's tau along with test statistics for evaluating whether the correlations are equal to zero.

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 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 `Sym::cor` is equivalent to the `Sym::cov` (p. 998) command with a different default setting.

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.
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

wgt = *name (optional)*	Name of vector containing weights. The number of rows of the weight vector should match the number of rows in the original matrix.
wgtmethod = *arg (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.
pairwise	Compute using pairwise deletion of observations with missing cases (pairwise samples).
df	Compute covariances with a degree-of-freedom correction to account for estimated means (for centered specifications), and any partial conditioning variables.
multi = *arg (default* = "none"*)*	Adjustment to p-values for multiple comparisons: none ("none"), Bonferroni ("bonferroni"), Dunn-Sidak ("dunn").
outfmt = *arg* (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").
prompt	Force the dialog to appear from within a program.
p	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 `Sym::cov` (p. 998). For simple forms of the calculation, see `@cor` (p. 766), and `@cov` (p. 767) in the *Command and Programming Reference*.

cov	Sym Views

Compute covariances, correlations, and other measures of association for the columns in a matrix.

You may compute measures related to Pearson product-moment (ordinary) covariances and correlations, Spearman rank covariances, or Kendall's tau along with test statistics for evaluating whether the correlations are equal to zero.

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 or rows should match the original matrix.

You may specify keywords from one of the four sets (Pearson correlation, Spearman rank 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 `Sym::cov` is equivalent to the `Sym::cor` (p. 995) command with a different default setting.

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.
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

wgt = *name (optional)*	Name of vector containing weights. The number of rows of the weight vector should match the number of rows in the original matrix.
wgtmethod = *arg (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.
pairwise	Compute using pairwise deletion of observations with missing cases (pairwise samples).
df	Compute covariances with a degree-of-freedom correction to account for estimated means (for centered specifications), and any partial conditioning variables.

multi = *arg* *(default* = "none"*)*	Adjustment to p-values for multiple comparisons: none ("none"), Bonferroni ("bonferroni"), Dunn-Sidak ("dunn").
outfmt = *arg* *(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").
prompt	Force the dialog to appear from within a program.
p	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 `Sym::cor` (p. 995). For simple forms of the calculation, see `@cor` (p. 766), and `@cov` (p. 767) in the *Command and Programming Reference*.

display	Sym Views

Display table, graph, or spool output in the sym object window.

Display the contents of a table, graph, or spool in the window of the sym object.

Syntax

sym_name.**display** *object_name*

Examples

```
sym1.display tab1
```

Display the contents of the table TAB1 in the window of the object SYM1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

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 123 of *User's Guide I* for a discussion of labels and display names.

See also `Sym::label` (p. 1015).

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

out = *arg* *(default* = "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.e.*, if you specify "cproport", a scree plot will not be provided unless requested).
n = *integer*	Maximum number of components to retain when presenting table ("out = table") or eigenvalue graph ("out = graph") results. The default is to set n to the number of variables. EViews will retain the minimum number satisfying any of: "n = ", "mineig = " or "cproport = ".
mineig = *arg* *(default* = 0*)*	Minimum eigenvalue threshold value: we retain components with eigenvalues that are greater than or equal to the threshold. EViews will retain the minimum number satisfying any of: "n = ", "mineig = " or "cproport = ".

cproport = arg *(default = 1)*	Cumulative proportion threshold value: we retain k, the number of components required for the sum of the first k eigenvalues exceeds the specified value for the cumulative variance explained proportion. EViews will retain the minimum number satisfying any of: "n = ", "mineig = " or "cproport = ".
eigval = *vec_name*	Specify name of vector to hold the saved the eigenvalues in workfile.
eigvec = *mat_name*	Specify name of matrix to hold the save the eigenvectors in workfile.
prompt	Force the dialog to appear from within a program.
p	Print results.

Graph Options

scale = *arg*, *(default* = "normload"*)*	Diagonal matrix scaling of the loadings: normalize loadings ("normload"), normalize scores ("normscores"), symmetric weighting ("symmetric"), user-specified power (*arg* = *number*).
mult = *arg* *(default* = "first"*)*	Multiple series handling: plot first against remainder ("first"), plot as x-y pairs ("pair"), lower-triangular plot ("lt").
nocenter	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 685 of *User's Guide I* for a discussion of principal components analysis on a group of series, which describes a superset of the tools for eigenvalue calculations offered by the sym matrix.

export	Sym Procs

Export sym to disk as an Excel 2007 XLSX, CSV, tab-delimited ASCII text, RTF, HTML, Enhanced Metafile, LaTeX, PDF, or Markdown file.

Syntax

sym_name.**export***(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.

The base syntax for writing Excel 2007 files is:

sym_name.**export**(*options*) *[path\]file_name [table_description]*

where the *table_description* may contain:

- "range = *arg*", where arg is top left cell of the destination 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.

Options

t = *file_type* (*default* = "csv")	Specifies the file type, where *file_type* may be one of: "excelxml" (Excel 2007 (xml)),"csv" (CSV - comma-separated), "rtf" (Rich-text format), "txt" (tab-delimited text), "html" (HTML - Hypertext Markup Language), "emf" (Enhanced Metafile), "pdf" (PDF - Portable Document Format), "tex" (LaTeX), or "md" (Markdown). Files will be saved with the ".xlsx", ".csv", ".rtf", ".txt", ".htm", ".emf", ".pdf", ".tex", or ".md" extensions, respectively.
s = *arg*	Scale size, where *arg* is from 5 to 200, representing the percentage of the original table size (only valid for HTML or RTF files).
n = *string*	Replace all cells that contain NA values with the specified string. "NA" is the default.
h / -h	Include(/do not include) column and row headers. The default is to not include the headers
prompt	Force the dialog to appear from within a program.

PDF Options

landscape	Save in landscape mode (the default is to save in portrait mode).
size = *arg* (*default* = "letter")	Page size: "letter", "legal", "a4", and "custom".
width = *number* (*default* = 8.5)	Page width in inches if "size = custom".
height = *number* (*default* = 11)	Page height in inches if "size = custom".
leftmargin = *number* (*default* = 0.5)	Left margin width in inches.
rightmargin = *number* (*default* = 0.5)	Right margin width in inches.
topmargin = *number* (*default* = 1)	Top margin width in inches.
bottommargin = *number* (*default* = 1)	Bottom margin width in inches.

LaTeX Options

texspec / -texspec	[Include / Do not include] the full LaTeX documentation specification in the LaTeX output. The default behavior is taken from the global default settings.

Excel Options

mode = *arg*	Specify whether to create a new file, overwrite an existing file, or update an existing file. arg may be "create" (create new file only; error on attempt to overwrite) or "update" (update an existing file, only overwriting the area specified by the range = *table_description*). If the "mode = " option is not used, EViews will create a new file, unless the file already exists in which case it will overwrite it. Note that the "mode = update" option is only available for Excel in 1) Excel versions through 2003, if Excel is installed, and 2) Excel 2007 (xml). Note: Excel does not need to be installed for Excel 2007 writing.

Excel 2007 Options

mode = *arg*	Specify whether to create a new file, overwrite an existing file, or update an existing file. arg may be "create" (create new file only; error on attempt to overwrite) or "update" (update an existing file, only overwriting the area specified by the range = *table_description*). If the "mode = " option is not used, EViews will create a new file, unless the file already exists in which case it will overwrite it. Note that the "mode = update" option is only available for Excel in 1) Excel versions through 2003, if Excel is installed, and 2) Excel 2007 (xml). Note: Excel does not need to be installed for Excel 2007 writing.
cellfmt = *arg*	Specify whether to use EViews, pre-existing, or remove cell formatting (colors, font, number formatting when possible, column widths and row heights) for the written range. *arg* may be "eviews" (replace current formatting in the file with the same cell formatting in EViews), "preserve" (leave current cell formatting already in the Excel file), or "clear" (remove current formatting and do not replace).
strlen = *arg* (default = 256)	Specify the maximum the number of characters written for cells containing text. Strings in cells which are longer the max, will be truncated.

Examples

The command:

```
sym1.export mysym
```

exports the data in SYM1 to a CSV file named "mysym.CSV" in the default directory.

```
sym1.export(h, t=csv, n="NaN") mysym
```

saves the contents of SYM1 along with the column and row headers to a CSV (comma separated value) file named "mysym.CSV" and writes all NA values as "NaN".

```
sym1.export(h, t=html, s=50) mysym
```

exports the data in SYM1 along with the column and row headers to a HTML file named "mysym.HTM" at half of the original size.

```
sym1.export(n=".", r=B) mysym
```

saves the data in the second column to a CSV file named "mysym.CSV", and writes all NA values as ".".

```
sym1.export(t=excelxml, cellfmt=clear, mode=update) mysym
    range=Country!b5
```

writes the data in SYM1 to the preexisting "mysym.XLSX" Excel file to the "Country" sheet at cell B5, where all cell formatting is cleared.

Cross-references

See also `Sym::import` (p. 1009).

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:

$$m1 = \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix} \tag{1.5}$$

Cross-references

See Chapter 11. "Matrix Language," on page 279 of the *Command and Programming Reference* for a detailed discussion of vector and matrix manipulation in EViews.

import	Sym Procs

Imports data from a foreign file into the sym object.

Syntax

sym_name.**import***([type =]) source_description import_specification*

- *source_description* should contain a description of the file from which the data is to be imported. The specification of the description is usually just the path and file name of the file, however you can also specify more precise information. See `wfopen` (p. 640) of the *Command and Programming Reference* for more details on the specification of *source_description*.
- The optional "type = " option may be used to specify a source type. For the most part, you should not need to specify a "type = " option as EViews will automatically determine the type from the filename. The following table summaries the various source formats and along with the corresponding "type = " keywords:

	Option Keywords
Excel (through 2003)	"excel"
Excel 2007 (xml)	"excelxml"

HTML	"html"
Text / ASCII	"text"

- *import_specification* can be used to provide additional information about the file to be read. The details of import_specification will depend upon the type of file being imported.

Excel Files

The syntax for reading Excel files is:

sym_name.**import(type = excel***[xml]*) *source_description [table_description] [variables_description]*

The following *table_description* elements may be used when reading Excel data:

- "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.

- "byrow", transpose the incoming data. This option allows you to read files where the series are contained in rows (one row per series) rather than columns.

The optional *variables_description* may be formed using the elements:

- "colhead = *int*", number of table rows to be treated as 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). *Note: If a "range = " argument is not specified, then EViews will only scan the first five rows of data to try and determine the data format for each column. Likewise, if the "na = " argument is not specified, EViews will also try to determine possible NA values by looking for repeated values in the same rows. If the first five rows are not enough to correctly determine the data format, use the "scan = " argument to instruct EViews to look at more rows. In addition, you may want to specify a the "na = " value to override any dynamic NA value that EViews may determine on its own.*

- "firstobs = *int*", first observation to be imported from the 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 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.

Excel Examples

```
sym_obj.import "c:\data files\data.xls"
```

loads the active sheet of DATA.XLSX into the SYM_NAME sym object.

```
sym_obj.import "c:\data files\data.xls" range="GDP data"
```

reads the data contained in the "GDP data" sheet of "Data.XLS" into the SYM_OBJ object.

HTML Files

The syntax for reading HTML pages is:

sym_name.**import(type = html)** *source_description [table_description] [variables_description]*

The following *table_description* elements may be used when reading an HTML file or page:

- "table = *arg*", where *arg* specifies which HTML 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.
- "byrow", transpose the incoming data. This option allows you to import files where the series are contained in rows (one row per series) rather than columns.

The optional *variables_description* may be formed using the elements:

- "colhead = *int*", number of table rows to be treated as 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). *Note: If a "range = " argument is not specified, then EViews will only scan the first five rows of data to try and determine the data format for each column. Likewise, if the "na = " argument is not specified, EViews will*

also try to determine possible NA values by looking for repeated values in the same rows. If the first five rows are not enough to correctly determine the data format, use the "scan = " argument to instruct EViews to look at more rows. In addition, you may want to specify a the "na = " value to override any dynamic NA value that EViews may determine on its own.

- "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.

HTML Examples

```
sym01.import 01"c:\data.html"
```

loads into the SYM01 matrix object the data located in the HTML file "Data.HTML" located on the C:\ drive

Text and Binary Files

The syntax for reading text or binary files is:

sym_name.**import(type** = *arg***)** *source_description [table_description] [variables_description]*

If a *table_description* is not provided, EViews will attempt to read the file as a free-format text file. The following *table_description* elements may be used when reading a text or binary file:

- "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.
- "byrow", transpose the incoming data. This option allows you to import files where the series are contained in rows (one row per series) rather than columns.
- "lastcol", include implied last column. For lines that end with a delimiter, this option adds an additional column.

  When importing a CSV file, lines which have the delimiter as the last character (for example: 'name,description,date,'), EViews normally determines the line to have 3 columns. With the above option, EViews will determine the line to have 4 columns. Note this is not the same as a line containing 'name,description,date'. In this case, EViews will always determine the line to have 3 columns regardless if the option is set.

A central component of the *table_description* element is the format statement. You may specify the data format using the following table descriptors:

- Fortran Format:

  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 Format:

  rformat = "*[n1]Type[Width][.Precision]*, *[n2]Type[Width][.Precision]*, ...)"

  where optional type is "$" for string or "#" for number, and *n1*, *n2*, *n3*, *n4*, *etc*. are the range of columns containing the data.

- C printf/scanf Format:

  cformat = "*fmt*"

  where *fmt* follows standard C language (printf/scanf) format rules.

The optional *variables_description* may be formed using the elements:

- "colhead = *int*", number of table rows to be treated as column headers.
- "types = ("*arg1*","*arg2*",...)", user specified data types of the series. If types are provided they will override the types automatically detected by EViews. You may use any of the following format keywords: "a" (character data), "f" (numeric data), "d" (dates), or "w" (EViews automatic detection). This option is rarely used.
- "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). *Note: If a "range = " argument is not specified, then EViews will only scan the first five rows of data to try and determine the data format for each column. Likewise, if the "na = " argument is not specified, EViews will also try to determine possible NA values by looking for repeated values in the same rows. If the first five rows are not enough to correctly determine the data format, use the "scan = " argument to instruct EViews to look at more rows. In addition, you may want to specify a the "na = " value to override any dynamic NA value that EViews may determine on its own.*
- "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.

Text and Binary File Examples (.txt, .csv, etc.)

```
sym2.import c:\data.csv skip=5
```

reads "Data.CSV" into SYM2, skipping the first 5 rows.

```
sym01.import(type=text) c:\date.txt delim=comma
```

loads the comma delimited data "Date.TXT" into the SYM01 matrix object.

Cross-references

See `Sym::export` (p. 1005).

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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User's Guide I* for a discussion of labels.

See also `Sym::displayname` (p. 1002).

olepush

Sym Procs

Push updates to OLE linked objects in open applications.

Syntax

sym_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

read	Sym Procs

Import data from a foreign disk file into a symmetric matrix.

(This is a deprecated method of importing into a sym. See `Sym::import` (p. 1009) for the currently supported method.)

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

prompt	Force the dialog to appear from within a program.

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 *default* 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.
skipcol = *integer*	Number of columns to skip. Must be used with the "*rect*" option.
skiprow = *integer*	Number of rows to skip. Must be used with the "rect" option.
comment = *symbol*	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.
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

t	Read data organized by column (transposed). Default is to read by row.
letter_number (default = "b2"*)*	Coordinate of the upper-left cell containing data.
s = *sheet_name*	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 152 of *User's Guide I* for a discussion and examples of importing data from external files.

See also `Sym::export` (p. 1005).

resize	Sym Procs

Resize the sym object.

Syntax

sym_name.**resize** *rows/cols*

Examples

```
sym1.resize 20
```

resizes the sym SYM1 to 20 rows/columns, retaining the contents of any existing elements and initializing new elements to 0.

setattr	Sym Procs

Set the object attribute.

Syntax

sym_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the "revised" attribute in the object A to the string "never", and extracts the attribute into the string object S.

Cross-references

See "Adding Custom Attributes in the Label View" on page 123 and "Adding Your Own Label Attributes" on page 70 of *User's Guide I*.

setcollabels	Sym Procs

Set the column labels in a sym object.

Syntax

sym_name.**setcollabels** *label1 label2 label3....*

Follow the `setcollabels` command with a space delimited list of column labels. Note that each column label should not contain spaces unless it is enclosed in quotes. If you provide fewer labels than there are columns, EViews will keep the corresponding default column names ("C11", "C12", etc...).

Examples

```
sym1.setcollabels USA UK FRANCE
```

sets the column label for the first column in symmetric matrix SYM1 to USA, the second to UK, and the third to FRANCE.

Cross-references

See also `Sym::setrowlabels` (p. 1022).

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:

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

To specify a format that groups digits into thousands using a comma separator, place a "t" after the format character. For example, to obtain a fixed number of decimal places with commas used to separate thousands, use "ft*[.precision]*".

To use the period character to separate thousands and commas to denote decimal places, use ".." (two periods) when specifying the precision. For example, 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)"), 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. 1023), `Sym::setindent` (p. 1021) and `Sym::setjust` (p. 1022) 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 indentation setttings are taken from the Global Defaults for spreadsheet views ("Spreadsheet Data Display" on page 1019 of *User's Guide I*) at the time the spreadsheet was 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. 1023) and `Sym::setjust` (p. 1022) for details on setting spreadsheet widths and justification.

setjust	Sym Procs

Set the horizontal justification for all cells in the spreadsheet view of the sym object.

Syntax

sym_name.**setjust** *format_arg*

where *format_arg* may be set to left, center, right, or auto (strings are left-justified and numbers are right-justified). Default display settings can be set in General Options; see "Spreadsheet Data Display" on page 1019 of *User's Guide I*.

Examples

```
sym1.setjust left
```

left-justifies the cells in the spreadsheet view of the sym SYM1.

Cross-references

See `Sym::setwidth` (p. 1023) and `Sym::setindent` (p. 1021) for details on setting spreadsheet widths and indentation.

setrowlabels	Sym Procs

Set the row labels in a sym object.

Syntax

sym_name.**setrowlabels** *label1 label2 label3….*

Follow the `setrowlabels` command with a space delimited list of row labels. Note that each row label should not contain spaces unless it is enclosed in quotes. If you provide fewer labels than there are rows, EViews will use the corresponding default row names ("R11", "R12", etc...).

Examples

```
sym1.setrowlabels USA UK FRANCE
```

sets the row label for the first row in sym SYM1 to USA, the second to UK, and the third to FRANCE.

Cross-references

See also `Sym::setcollabels` (p. 1020).

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. 1021) and `Sym::setjust` (p. 1022) 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.

showlabels	Sym Procs

Displays the custom row and column labels of a sym spreadsheet.

Syntax

sym_name.**showlabels** *mode*

where *mode* is either 0 or 1 where 0 displays the default row and column labels and 1 displays the custom row and column labels (if present).

Examples

```
s1.showlabels 1
```

displays the custom row and column labels for the S1 spreadsheet. If custom labels have not been set the default labels will be displayed.

```
s1.showlabels 0
```

displays the default row and column labels for the S1 spreadsheet.

Cross-references

See also `Sym::setcollabels` (p. 1020) and `Sym::setrowlabels` (p. 1022).

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 450 and "Descriptive Statistics" on page 667 of *User's Guide I* for a discussion of the descriptive statistics views.

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 279 of the *Command and Programming Reference* for a discussion of matrix objects in EViews.

See also `Matrix::matrix` (p. 583).

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.

This routine should realistically only be used in the oft-hand chance that you wish to write into a Lotus file. Improved Excel, text, and other format writing is available in `Sym::export` (p. 1005).

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

prompt	Force the dialog to appear from within a program.

Other options are used to specify the format of the output file.

File type

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

If you omit the "*t* = " option, EViews will determine the type based on the file extension. Unrecognized extensions will be treated as ASCII files. For Lotus and Excel spreadsheet files specified without the "*t* = " option, EViews will automatically append the appropriate extension if it is not otherwise specified.

ASCII text files

na = *string*	Specify text string for NAs. Default is "NA".
d = *arg*	Specify delimiter *(default* is tab*)*: "s" (space), "c" (comma).
t	Write by column (transpose the data). Default is to write by row.

Spreadsheet (Lotus, Excel) files

letter_number	Coordinate of the upper-left cell containing data.
t	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 171 of *User's Guide I* for a discussion.

See also `Sym::export` (p. 1005) and `Sym::read` (p. 1017).

System

System of equations for estimation.

System Declaration

system .....declare system object (p. 1064).

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. 604).

System Methods

3sls.....three-stage least squares (p. 1032).
arch .....estimate generalized autoregressive conditional heteroskedasticity (GARCH) models (p. 1034).
fiml.....full information maximum likelihood (p. 1047).
gmm .....generalized method of moments (p. 1049).
ls .....ordinary least squares (p. 1054).
sur.....seemingly unrelated regression (p. 1063).
tsls.....two-stage least squares (p. 1065).
wls .....weighted least squares (p. 1067).
wtsls.....weighted two-stage least squares (p. 1068).

System Views

cellipse .....confidence ellipses for coefficient restrictions (p. 1039).
coefcov .....coefficient covariance matrix (p. 1041).
correl.....display graphs or tables of residual autocorrelations and cross-correlations (p. 1042).
derivs.....derivatives of the system equations (p. 1043).
display.....display table, graph, or spool in object window (p. 1044).
endog.....table or graph of endogenous variables (p. 1045).
estcov .....display the covariance matrix used in estimation (p. 1046).
garch .....conditional variance/covariance of (G)ARCH estimation (p. 1048).
grads.....examine the gradients of the objective function (p. 1051).
jbera .....multivariate residual normality test (p. 1052).
label .....label information for the system object (p. 1053).
output.....table of estimation results (p. 1058).
qstats .....multivariate residual autocorrelation Portmanteau tests (p. 1059).
representations.....text showing specification of the system (p. 1060).

residcor ..... residual correlation matrix (p. 1060).
residcov ..... residual covariance matrix (p. 1061).
resids ..... residual graphs or spreadsheets (p. 1061).
results ..... table of estimation results (p. 1062).
spec ..... text representation of system specification (p. 1063).
wald ..... Wald coefficient restriction test (p. 1066).

System Procs

append ..... add a line of text to the system specification (p. 1033).
autospec ..... automatically create system specification text (p. 1038).
clearhist ..... clear the contents of the history attribute (p. 1040).
clearremarks ..... clear the contents of the remarks attribute (p. 1041).
copy ..... creates a copy of the system (p. 1042).
displayname ..... set display name (p. 1045).
makeendog ..... make group of endogenous series (p. 1055).
makegarch ..... generate conditional variance series (p. 1055).
makeloglike ..... create and save log likelihood contribution from system (ARCH estimation) (p. 1056).
makemodel ..... create a model from the estimated system (p. 1057).
makeresids ..... make series containing residuals from system (p. 1057).
olepush ..... push updates to OLE linked objects in open applications (p. 1058).
setattr ..... set the value of an object attribute (p. 1062).
updatecoefs ..... update coefficient vector(s) from system (p. 1066).

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.
@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 or z-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).
@linecount.....scalar containing the number of lines in the System object.
@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.
@estcov.....(sym) residual covariance matrix used in estimation (see `System::estcov` (p. 1046) in *Object Reference*).
@pvals.....vector containing the coefficient probability values.
@residcov.....(sym) covariance matrix of the residuals.
@stderrs.....vector of standard errors for coefficients.
@tstats.....vector of *t*-statistic or *z*-statistic values for coefficients.

String values

@attr("*arg*").....string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@command.....full command line form of the estimation command. Note this is a combination of `@method` and `@options`.
@description.....string containing the System object's description (if available).
@detailedtype.....returns a string with the object type: "SYSTEM".
@displayname.....returns the System's display name. If the System has no display name set, the name is returned.
@line(i).....returns a string containing the *i*-th line of the System object.
@method.....command line form of estimation method type ("ARCH", "LS", *etc.*...).
@name.....returns the System's name.

@options..... command line form of estimation options.
@remarks ..... string containing the system object's remarks (if available).
@smpl ..... sample used for estimation.
@svector..... returns an Svector where each element is a line of the System object.
@svectornb..... same as `@svector`, with blank lines removed.
@type ..... returns a string with the object type: "SYSTEM".
@updatetime..... returns a string representation of the time and date at which the System was last updated.

System Examples

To estimate a system using GMM and to create residual series for the estimated system:

```
sys1.gmm(i,m=7,c=.01,b=v)
sys1.makeresids consres incres saveres
```

To test coefficients using a Wald test:

```
sys1.wald c(1)=c(4)
```

To save the coefficient covariance matrix:

```
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 *(default)*	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 = *integer*	Maximum number of iterations.
c = *number*	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
l = *number*	Set maximum number of iterations on the first-stage coefficient estimation to get the one-step weighting matrix.
numericderiv / -numericderiv	[Do / do not] use numeric derivatives only. If omitted, EViews will follow the global default.
fastderiv / -fastderiv	[Do / do not] use fast derivative computation. If omitted, EViews will follow the global default.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
prompt	Force the dialog to appear from within a program.
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 43. “System Estimation,” on page 897 of *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 macro1
macro1.append cons=c(1)+c(2)*gdp+c(3)*cons(-1)
macro1.append inv=c(4)+c(5)*tb3+c(6)*d(gdp)
macro1.append gdp=cons+inv+gov
macro1.append inst tb3 gov cons(-1) gdp(-1)
macro1.gmm
show macro1.results
```

The first line declares a system. The next three lines append the specification of each endogenous variable in the system. The fifth line appends the list of instruments to be used in estimation. The last two lines estimate the model by GMM and display the estimation results.

Cross-references

For details, see "How to Create and Specify a System" on page 900 of *User's Guide II*.

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 VECH Argument Options

c(*arg*)	where *arg* may be "scalar", "diag" (diagonal), "rank1" (rank one), "fullrank", "indef" (indefinite - default), or "vt" (variance target).
arch(*n, arg*)	where *n* indicates the order of the term, and *arg* may be "scalar", "diag" (diagonal), "rank1" (rank one), "fullrank", or "indef" (indefinite - default).
garch(*n, arg*)	where *n* indicates the order of the term, and *arg* may be "scalar", "diag" (diagonal), "rank1" (rank one), "fullrank", or "indef" (indefinite - default).

tarch(*n, arg*)	where *n* indicates the order of the term, and *arg* may be "scalar", "diag" (diagonal), "rank1" (rank one), "fullrank", or "indef" (indefinite - default).
exog(*series, arg*)	where *series* indicates a series name, and *arg* 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 Constant Conditional Correlation model:

system_name.**arch***(options)* **@ccc c***(arg) [***arch***(n[, arg])] [***tarch***(n[, arg])] [***garch***(n[, arg])] [***exog***(series, arg)]*

Indicate a Constant Conditional 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.

Constant Conditional Correlation Argument Options

c(*arg*)	where *arg* may be "scalar" (default) or "vt" (variance target).
arch(*n[, arg]*)	where *n* indicates the order of the term, and the optional *arg* may be "scalar" (default).
garch(*n[, arg]*)	where *n* indicates the order of the term, and the optional *arg* may be "scalar" (default).
tarch(*n[, arg]*)	where *n* indicates the order of the term, and the optional *arg* may be "scalar" (default).
exog(*series, arg*)	where *series* indicates a series name, and *arg* 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(*arg*)	where *arg* may be "scalar", "diag" (diagonal), "rank1" (rank one), "fullrank", "indef" (indefinite - default), or "vt" (variance target).
arch(*n[, arg]*)	where *n* indicates the order of the term, and the optional *arg* may be "diag" (diagonal - default).
garch(*n[, arg]*)	where *n* indicates the order of the term, and the optional *arg* may be "diag" (diagonal - default).
tarch(*n[, arg]*)	where *n* indicates the order of the term, and the optional *arg* may be "diag" (diagonal - default).
exog(*series, arg*)	where *series* indicates a series name, and *arg* 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 *t*-distribution (the default is the conditional normal distribution).
optmethod = *arg*	Optimization method: "bfgs" (BFGS); "newton" (Newton-Raphson), "opg" or "bhhh" (OPG or BHHH), "legacy" (EViews legacy). "bfgs" is the default for new equations.
optstep = *arg*	Step method: "marquardt" (Marquardt - default); "dogleg" (Dogleg); "linesearch" (Line search). (Applicable when "optmethod = bfgs", "optmethod = newton" or "optmethod = opg".)
b	Use Berndt-Hall-Hall-Hausman (BHHH) as maximization algorithm. The default is Marquardt. (Applicable when "optmethod = legacy".)
cov = *arg*	Covariance method: "ordinary" (default method based on inverse of the estimated information matrix), "huber" or "white" (Huber-White sandwich method), "bollerslev" (Bollerslev-Wooldridge method).
covinfo = *arg*	Information matrix method: "opg" (OPG); "hessian" (observed Hessian), " (Applicable when non-legacy "optmethod = " with "cov = ordinary".)

h	Bollerslev-Wooldridge robust quasi-maximum likelihood (QML) covariance/standard errors. (Applicable for "optmethod = legacy" when estimating assuming normal errors.)
m = *integer*	Set maximum number of iterations.
c = *scalar*	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.
s	Use the current coefficient values in "C" as starting values (see also `param` (p. 564) of the *Command and Programming Reference*).
numericderiv / -numericderiv	[Do / do not] use numeric derivatives only. If omitted, EViews will follow the global default.
fastderiv / -fastderiv	[Do / do not] use fast derivative computation. If omitted, EViews will follow the global default.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
coef = *arg*	Specify the name of the coefficient vector of a system's variance component; the default behavior is to use the "C" coefficient vector.
backcast = *n*	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.
prompt	Force the dialog to appear from within a program.
p	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 27. "ARCH and GARCH Estimation," on page 273 of *User's Guide II* for a discussion of ARCH models.

See also `System::makegarch` (p. 1055) and `Equation::arch` (p. 64).

autospec	System Procs

Automatically create system specification text.

Syntax

system_name.**autospec***(options) y1 y2 y3 ...* **@reg** *x1 x2 x3 ... [***@eqreg** *w1 w2 ...] [***@inst** *z1 z2 ...] [***@eqinst** *z3 z4 ...]*

Defines the specification of the system. The **@reg** list consists of regressors with common coefficients in the system. The **@eqreg** list consists of regressors with different coefficients in each equation. The list of variables that follow **@inst** are the common instruments. The list of variables that follow **@eqinst** are the equation specific instruments.

Options

ytrans = *arg*	Dependent variable transformation: none *(default)*, log ("log"), difference ("d"), difference of logs ("dlog"), one percentage change in decimal ("pch"), one-period percentage change—annualized, in percent ("pcha"), one-year percentage change in decimal ("pchy").
prompt	Force the dialog to appear from within a program.

Examples

```
system sys1
sys1.autospec @regs y1 y2 y3 @regs x1 x2 c @inst z1 z2 z3
```

creates a system named SYS1 with the series Y1, Y2 and Y3 as the dependent variables and a common intercept and coefficients on X1 and X2, with common instruments Z1, Z2, and Z3.

Cross-references

See Chapter 43. "System Estimation," on page 897 of *User's Guide II* for a discussion of system objects in EViews.

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

ind = *arg*	Specifies whether and how to draw the individual coefficient intervals. The default is "ind = line" which plots the individual coefficient intervals as dashed lines. "ind = none" does not plot the individual intervals, while "ind = shade" plots the individual intervals as a shaded rectangle.
size = *number* *(default* = 0.95*)*	Set the size (level) of the confidence ellipse. You may specify more than one size by specifying a space separated list enclosed in double quotes.
dist = *arg*	Select the distribution to use for the critical value associated with the ellipse size. The default depends on estimation object and method. If the parameter estimates are least-squares based, the $F(2, n-2)$ distribution is used; if the parameter estimates are likelihood based, the $\chi^2(2)$ distribution will be employed. "dist = f" forces use of the F-distribution, while "dist = c" uses the χ^2 distribution.
prompt	Force the dialog to appear from within a program.
p	Print the graph.

Examples

The two commands:

```
sys1.cellipse c(1), c(2), c(3)
sys1.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).

```
sys1.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 Intervals and Confidence Ellipses" on page 203 of *User's Guide II* for discussion.

See also `System::wald` (p. 1066).

clearhist	System Procs

Clear the contents of the history attribute.

Removes the system's history attribute, as shown in the label view of the system.

Syntax

system_name.**clearhist**

Examples

```
s1.clearhist
s1.label
```

The first line removes the history from the system S1, and the second line displays the label view of S1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names.

See also `System::label` (p. 1053).

clearremarks	System Procs

Clear the contents of the remarks attribute.

Removes the system's remarks attribute, as shown in the label view of the system.

Syntax

system_name.**clearremarks**

Examples

```
s1.clearremarks
s1.label
```

The first line removes the remarks from the system S1, and the second line displays the label view of S1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names.

See also `System::label` (p. 1053).

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. 26) and System::spec (p. 1063).

copy	System Procs

Creates a copy of the system.

Creates either a named or unnamed copy of the system.

Syntax

system_name.**copy**

system_name.**copy** *dest_name*

Examples

```
s1.copy
```

creates an unnamed copy of the system S1.

```
s1.copy s2
```

creates S2, a copy of the system S1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

correl	System Views

Display graphs or tables of residual autocorrelations and cross-correlations.

Displays the auto and cross-correlation functions of the estimated system residals.

Syntax

system_name.**correl**(*n, options)*

You must specify the largest lag *n* to use in the computations. The default is to display a graphical view of the auto and cross-correlations.

Options

graph *(default)*	Display correlograms (graphs).
bylag	Display table of results grouped by lag.
bser	Display table of results grouped by series.

factor = chol	Factorization by the inverse of the Cholesky factor of the residual covariance matrix (if estimated by ARCH).
factor = cor	Factorization by the inverse square root of the residual correlation matrix (if estimated by ARCH; Doornik and Hansen, 1994).
factor = cov	Factorization by the inverse square root of the residual covariance matrix (if estimated by ARCH; Urzua, 1997).
name = *arg*	Save matrix of results.
prompt	Force the dialog to appear from within a program.
p	Print the correlograms.

Examples

```
sys.correl(24)
```

Displays the correlograms of the SER1 series for up to 24 lags.

Cross-references

See "Correlogram" on page 488 and "Cross Correlations and Correlograms" on page 706 of *User's Guide I* for related discussion of autocorrelation and cross-correlation functions, respectively. See also "Residual Tests" on page 955 for related testing in a VAR context.

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" on page 1537 of *User's Guide II* for details on the computation of derivatives.

See also `Equation::makederivs` (p. 190) for additional routines for examining derivatives, and `System::grads` (p. 1051), and `Equation::makegrads` (p. 194) for corresponding routines for gradients.

display	System Views

Display table, graph, or spool output in the system object window.

Display the contents of a table, graph, or spool in the window of the system object.

Syntax

system_name.**display** *object_name*

Examples

```
system1.display tab1
```

Display the contents of the table TAB1 in the window of the object SYSTEM1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

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 123 of *User's Guide I* for a discussion of labels and display names.

See also `System::label` (p. 1053).

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. 1055), `System::system` (p. 1064).

estcov	System Views

Displays the covariance matrix used in estimation.

The estimation covariance contains:

1. the identity matrix for OLS and TSLS
2. a diagonal matrix with equation variances used to compute WOLS and WTSLS
3. the residual covariance matrix used to compute SUR and 3SLS
4. the residual covariance matrix for unrestricted FIML; diagonal residual covariance matrix for diagonal FIML; user-specified covariance for user-covariance FIML
5. the long-run covariance of the moments used to compute the weighting matrix for GMM estimates
6. a matrix of missing values for ARCH

Syntax

system_name.**estcov***(options)*

Options

p	Print the estimation covariance.

Examples

```
sys1.estcov
```

displays the estimation covariance.

Cross-references

See also "System Views" on page 913 of *User's Guide II*.

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

rcov = *arg*	Restricted residual covariance matrix in estimation: "diag" (non-zero diagonal and zero off-diagonal elements), "usercov" (fully specified user covariance matrix), "user-factor" user provides the matrix P, such that PP' equals the fully specified covariance matrix). Note that system objects estimated using a restricted FIML estimator are not backward compatible with earlier versions of EViews, and will be dropped from the workfile if opened in a version prior to 9.5.
rcovname = *arg*	Name of the matrix for determining the user specified residual covariance matrix. (Applicable when "rcov = usercov" or "rcov = userfactor".)
optmethod = *arg*	Optimization method: "bfgs" (BFGS); "newton" (Newton-Raphson), "opg" or "bhhh" (OPG or BHHH), "legacy" (EViews legacy). "bfgs" is the default for new equations.
optstep = *arg*	Step method: "marquardt" (Marquardt - default); "dogleg" (Dogleg); "linesearch" (Line search). (Applicable when "optmethod = bfgs", "optmethod = newton" or "optmethod = opg".)
b	Use Berndt-Hall-Hall-Hausman (BHHH) as maximization algorithm. The default is Marquardt. (Applicable when "optmethod = legacy".)
cov = *arg*	Covariance method: "ordinary" (default method based on inverse of the estimated information matrix), "huber" or "white" (Huber-White sandwich method).

covinfo = *arg*	Information matrix method: “opg” (OPG); “hessian” (observed Hessian), “ (Applicable when non-legacy “optmethod = ” with “cov = ordinary”.)
m = *integer*	Maximum number of iterations.
c = *number*	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
numericderiv / -numericderiv	[Do / do not] use numeric derivatives only. If omitted, EViews will follow the global default.
fastderiv / -fastderiv	[Do / do not] use fast derivative computation. If omitted, EViews will follow the global default.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
prompt	Force the dialog to appear from within a program.
p	Print estimation results.

Examples

```
sys1.fiml
```

estimates SYS1 by FIML using the default settings.

```
sys1.fiml(rcov=diag)
```

estimates SYS1 by FIML with the off-diagonal residual covariances set to zero.

```
sys1.fiml(rcov=user, rcovname=mycov)
```

estimates a FIML model using MYCOV as the residual covariance matrix.

Cross-references

See Chapter 43. “System Estimation,” on page 897 of *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).
prompt	Force the dialog to appear from within a program.
p	Print the graph

Examples

```
sys1.garch(cor)
```

displays the conditional correlation graph of SYS1.

Cross-references

ARCH estimation is described in Chapter 27. “ARCH and GARCH Estimation,” on page 273 of *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

m = *integer*	Maximum number of iterations.
c = *number*	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
l = *number*	Set maximum number of iterations on the first-stage iteration to get the one-step weighting matrix.
numericderiv / -numericderiv	[Do / do not] use numeric derivatives only. If omitted, EViews will follow the global default.
fastderiv / -fastderiv	[Do / do not] use fast derivative computation. If omitted, EViews will follow the global default.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
w	Use White's diagonal weighting matrix (for cross section data).
b = *arg* *(default* = "nw"*)*	Specify the bandwidth: "nw" (Newey-West fixed bandwidth based on the number of observations), *number* (user specified bandwidth), "v" (Newey-West automatic variable bandwidth selection), "a" (Andrews automatic selection).
q	Use the quadratic kernel. Default is to use the Bartlett kernel.
n	Prewhiten by a first order VAR before estimation.
i	Iterate simultaneously over the weighting matrix and the coefficient vector.
s	Iterate sequentially over the weighting matrix and coefficient vector.
o *(default)*	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.
prompt	Force the dialog to appear from within a program.
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 simultaneously over the weight and coefficient vectors until convergence.

Cross-references

See Chapter 21. "Additional Regression Tools," on page 23 and Chapter 43. "System Estimation," on page 897 of *User's Guide II* 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. 1043).

jbera	System 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).
name = *arg*	Save the test statistics in a named matrix object. See below for a description of the statistics contained in the stored matrix.
prompt	Force the dialog to appear from within a program.
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_1^2 statistics for the third moments, the third column contains the fourth moments, and the fourth column holds the χ_1^2 statistics for the fourth moments. The sum of the second and fourth columns are the Jarque-Bera statistics reported in the last output table.

The 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_k^2 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

```
sys01.jbera(factor=cor,name=jb)
```

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 44. "Vector Autoregression (VAR) Models," on page 939 of *User's Guide II* for a discussion of the test in the context of 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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User's Guide I* for a discussion of labels.

See also `System::displayname` (p. 1045).

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. 564)).
numericderiv / -numericderiv	[Do / do not] use numeric derivatives only. If omitted, EViews will follow the global default.
fastderiv / -fastderiv	[Do / do not] use fast derivative computation. If omitted, EViews will follow the global default.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
prompt	Force the dialog to appear from within a program.
p	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 20. "Basic Regression Analysis," on page 5 and Chapter 21. "Additional Regression Tools," on page 23 of *User's Guide II* discuss the various regression methods in greater depth.

See Chapter 13. "Special Expression Reference," on page 323 of the *Command and Programming Reference* for special terms that may be used in system `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. 1045) and `Model::makegroup` (p. 627).

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

cor	Generate conditional correlation.
cov *(default)*	Generate conditional variance and covariance.
var	Generate conditional variance.
mat	Output as a matrix (default is to output as a series).
name = *arg*	Base name or matrix name of the data to be saved.
date = *arg*	Date to return conditional covariance value (used only with the mat option).
pre	Include presample data (used only with the mat option).
prompt	Force the dialog to appear from within a program.

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 27. "ARCH and GARCH Estimation," on page 273 of *User's Guide II* for a discussion of GARCH models.

See also `System::arch` (p. 1034), `System::arch` (p. 1034), `Equation::archtest` (p. 70), and `System::garch` (p. 1048).

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 logl
```

creates a series of log likelihood contribution for the system and saves it in the series LOG1.

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 52. "Models," on page 1177 of *User's Guide II* for a discussion of specifying and solving models in EViews.

See also `System::append` (p. 1033), `Model::merge` (p. 628) and `Model::solve` (p. 640).

makeresids — System Procs

Create residual series.

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

Syntax

system_name.**makeresids***(options) [residual_names]*

Follow the system 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

n = *arg*	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 = *arg*	Base name used to generate the name of the residual series.
prompt	Force the dialog to appear from within a program.

Examples

```
sys1.makeresids 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 `System::resids` (p. 1061).

olepush — System Procs

Push updates to OLE linked objects in open applications.

Syntax

system_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

output — System Views

Display estimation output.

`output` changes the default object view to display the estimation output (equivalent to using `System::results` (p. 1062)).

Syntax

system_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:

```
sys1.output
```

displays the estimation output for system SYS1.

Cross-references

See `System::results` (p. 1062).

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

maxlag = *arg*	Maximum lag in system specification (default = 0).
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).
prompt	Force the dialog to appear from within a program.
p	Print the Portmanteau test results.

Examples

```
show sys1.qstats(l0)
```

displays the portmanteau tests for lags up to 10.

Cross-references

See "Diagnostic Views" on page 954 of *User's Guide II* for a discussion of the Portmanteau tests and other VAR diagnostics.

See `Var::arlm` (p. 1140) for a related multivariate residual serial correlation LM test.

representations	System Views

Display text of specification for system objects.

Syntax

system_name.**representation***(options)*

Options

p	Print the representation text.

Examples

```
sys1.representations
```

displays the specifications of the equations in SYS1.

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. 1061) and `System::makeresids` (p. 1057).

residcov	System Views

Residual covariance matrix.

Displays the covariances of the residuals from each equation in the system.

Syntax

system_name.**residcov***(options)*

Options

p	Print the covariance matrix.

Examples

```
sys1.residcov
```

displays the residual covariance matrix of SYS1.

Cross-references

See also `System::residcor` (p. 1060) and `System::makeresids` (p. 1057).

resids	System Views

Display residuals.

`resids` displays multiple graphs or a spreadsheet of the residuals. Each graph will contain the residuals for each equation in the system.

Syntax

system_name.**resids***(options)*

Options

sheet	Display residuals in spreadsheet.
p	Print the table/graph.

Examples

```
sys1.resids
```

displays a graph of the residual series in system SYS1.

Cross-references

See also `System::makeresids` (p. 1057).

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.

setattr	System Procs

Set the object attribute.

Syntax

system_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the “revised” attribute in the object A to the string “never”, and extracts the attribute into the string object S.

Cross-references

See “Adding Custom Attributes in the Label View” on page 123 and “Adding Your Own Label Attributes” on page 70 of *User’s Guide I*.

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. 1033).

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 *(default)*	Iterate only on the coefficient vector with one step of the weighting matrix.
c	One step iteration on the coefficient vector after one step of the weighting matrix.
m = *integer*	Maximum number of iterations.

c = *number*	Set convergence criterion.The criterion will be set to the nearest value between 1e-24 and 0.2.
l = *number*	Set maximum number of iterations on the first-stage iteration to get one-step weighting matrix.
numericderiv / -numericderiv	[Do / do not] use numeric derivatives only. If omitted, EViews will follow the global default.
fastderiv / -fastderiv	[Do / do not] use fast derivative computation. If omitted, EViews will follow the global default.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
prompt	Force the dialog to appear from within a program.
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 43. "System Estimation," on page 897 of *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 mysys
```

creates a system named MYSYS.

Cross-references

Chapter 43. "System Estimation," on page 897 of *User's Guide II* provides a full discussion of system objects.

See `System::append` (p. 1033) 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

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.
numericderiv / -numericderiv	[Do / do not] use numeric derivatives only. If omitted, EViews will follow the global default.
fastderiv / -fastderiv	[Do / do not] use fast derivative computation. If omitted, EViews will follow the global default.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
i	Iterate on the weighting matrix and coefficient vector simultaneously.
s	Iterate on the weighting matrix and coefficient vector sequentially.
o *(default)*	Iterate only on the coefficient vector with one step of the weighting matrix.
c	One step iteration of the coefficient vector after one step of the weighting matrix.

l = *number*	Set maximum number of iterations on the first-stage iteration to get one-step weighting matrix.
prompt	Force the dialog to appear from within a program.
p	Print estimation results.

Examples

```
sys1.tsls
```

estimates the system object using TSLS.

Cross-references

See "Two-Stage Least Squares" on page 899 of *User's Guide II* for details on two-stage least squares estimation in systems.

See also `System::ls` (p. 1054). For estimation of weighted TSLS in systems, see `System::wtsls` (p. 1068).

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. 26).

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

p	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 210 of *User's Guide II* for a discussion of Wald tests.

See also `System::cellipse` (p. 1039), `testdrop` (p. 607), `testadd` (p. 606).

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 (*default)*	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 = *integer*	Maximum number of iterations.

c = *number*	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
l = *number*	Set maximum number of iterations on the first-stage coefficient estimation to get one-step weighting matrix.
numericderiv / -numericderiv	[Do / do not] use numeric derivatives only. If omitted, EViews will follow the global default.
fastderiv / -fastderiv	[Do / do not] use fast derivative computation. If omitted, EViews will follow the global default.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
prompt	Force the dialog to appear from within a program.
p	Print the estimation results.

Examples

```
sys1.wls
```

estimates the system of equations in SYS1 by weighted least squares.

Cross-references

See Chapter 43. "System Estimation," on page 897 of *User's Guide II* for a discussion of system estimation.

See also the available options for weighted least squares in `System::ls` (p. 1054).

wtsls — 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 *(default)*	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 = *integer*	Maximum number of iterations.
c = *number*	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients. The criterion will be set to the nearest value between 1e-24 and 0.2.
l = *number*	Set maximum number of iterations on the first-stage iteration to get the one-step weighting matrix.
numericderiv / -numericderiv	[Do / do not] use numeric derivatives only. If omitted, EViews will follow the global default.
fastderiv / -fastderiv	[Do / do not] use fast derivative computation. If omitted, EViews will follow the global default.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
prompt	Force the dialog to appear from within a program.
p	Print estimation results.

Examples

```
sys1.wtsls
```

estimates the system of equations in SYS1 by weighted two-stage least squares.

Cross-references

See "Weighted Two-Stage Least Squares" on page 899 of *User's Guide II* for further discussion.

See also `System::tsls` (p. 1065) 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. 457).
table ..... create table object (p. 1108).

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

display ..... display table, graph, or spool in object window (p. 1080).
label ..... label information for the table object (p. 1085).
sheet ..... view the table (p. 1107).
table ..... view the table (p. 1108).

Table Procs

clearhist ..... clear the contents of the history attribute (p. 1073).
clearremarks ..... clear the contents of the remarks attribute (p. 1073).
comment ..... adds or removes a comment in a table cell (p. 1074).
copy ..... creates a copy of the table (p. 1075).
copyrange ..... copies a portion of the table to another table (p. 1075).
copytable ..... copies the entire table to another table (p. 1076).
deletecells ..... delete cells from a table (p. 1077).
deletecol ..... remove columns from a table (p. 1078).
deleterow ..... remove rows from a table (p. 1079).
displayname ..... set display name (p. 1080).
fixcol ..... fixes a set of columns to left of the spreadsheet view so that the leading columns are always in view (p. 1081).

Table Data Members

String values

(i,j).....the (i,j)-th element of the table, formatted as a string.
@attr("*arg*").....string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@description.....string containing the Table object's description (if available).
@detailedtype.....string with the object type: "TABLE".
@displayname.....string containing the Table object's display name. If the Table has no display name set, the name is returned.

@find("*arg*") ..... string containing the cell identifiers meeting the boolean argument. This argument can be a mathematical or string expression. For example:

```
string s = tab1.@find("[b1:c15]>0.3")
```

returns a list of cells between B1 and C15 greater than 0.3;

```
string s = tabl1.@find("[a1:e67]== ""1949q4""")
```

returns a list of cells between A1 and E67 matching the string "1949q4" (string comparisons in `@find` ignore case);

```
string s = tab1.@find("[@all]==0.5")
```

returns a list of cells in the table equal to 0.5;

```
string s = t.@find("@instr([@all],""in"")")
```

returns a list of cells in the table containing the substring 'in';

```
string s = t.@find("[b3:c5]<[d5]")
```

returns a list of cells between B3 and C5 less than cell D5;

```
string s = t.@find("@left([@all],1)==""cp""")
```

returns a list of cells starting with "cp".

(See "Conditional Table Cells" on page 68 in *Command and Programming Reference* for syntax discussion.)

@name ..... string containing the Table object's name.

@remarks ..... string containing the Table object's remarks (if available).

@title ..... string containing the Table object's title (if available).

@type ..... string with the object type: "TABLE".

@updatetime ..... string representation of the time and date at which the Table was last updated.

Scalar values

@cols ..... number of columns in the table.

@colwidth(i) ..... the column width of the i-th column in the table.

@rowheight(i) ..... the row height of the i-th row in the table.

@rows ..... number of rows in the table.

@val(i,j) ..... the numerical value of the (i,j)-th element of the table.

Table Commands

setcell ..... format and fill in a table cell (p. 585).

setcolwidth ..... set width of a table column (p. 586).

setline ..... place a horizontal line in table (p. 588).

tabplace ..... insert a table into another table (p. 605).

All of the these commands are in the *Command and Programming Reference*. Note that with the exception of `tabplace`, these commands are supported primarily for backward compat-

iblity. There is a more extensive set of table procs for working with and customizing tables. See "Table Procs," on page 1070.

Table Examples

```
table(5,5) mytable
%strval = mytable(2,3)
mytable(4,4) = "R2"
mytable(4,5) = @str(eq1.@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.

clearhist	Table Procs

Clear the contents of the history attribute.

Removes the table's history attribute, as shown in the label view of the table.

Syntax

table_name.**clearhist**

Examples

```
t1.clearhist
t1.label
```

The first line removes the history from the table T1, and the second line displays the label view of T1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names.

See also `Table::label` (p. 1085).

clearremarks	Table Procs

Clear the contents of the remarks attribute.

Removes the table's remarks attribute, as shown in the label view of the table.

Syntax

table_name.**clearremarks**

Examples

```
t1.clearremarks
t1.label
```

The first line removes the remarks from the table T1, and the second line displays the label view of T1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names.

See also `Table::label` (p. 1085).

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:

cell	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 *number* (*e.g.*, "R1C2").
row[,] *col*	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 table commands see Chapter 3. "Working with Tables and Spreadsheets" in the *Command and Programming Reference*. See also `Table::setlines` (p. 1101) and `Table::setmerge` (p. 1102).

copy	Table Procs

Creates a copy of the table.

Creates either a named or unnamed copy of the table.

Syntax

table_name.**copy**

table_name.**copy** *dest_name*

Examples

```
t1.copy
```

creates an unnamed copy of the table T1.

```
t1.copy t2
```

creates T2, a copy of the table T1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

copyrange	Table Procs

Copies a portion of the table to the specified location in another table.

Syntax

table_name.**copyrange(options)** *s1 s2 destname d1*

table_name.**copyrange(options)** *sr1 sc1 sr2 sc2 destname dr1 dc1*

Options

Options are specified in parentheses after the keyword and are used to specify the format of the data in the destination table.

t	Places a transpose of the selected data into the destination area. The default is not transposed.

The copyrange command can be specified either using coordinates where columns are signified with a letter, and rows by a number (for example "A3" represents the first column, third row), or by row number and column number.

The first syntax represents coordinate form, where *s1* specifies the upper-left coordinate portion of the section of the source table to be copied, *s2* specifies the bottom-right coordinate, *destname* specifies the name of the table to copy to, and *d1* specifies the upper-left coordinate of the destination table.

The second syntax represents the row/column number form, where *sr1* specifies the source table upper row number, *sc1* specifies the source table left most column number, *sr2* specifies the source table bottom row number, *sc2* specifies the source table right most column number. *destname* specifies the name of the table to copy to, and *dr1* and *dr2* specify the upper and left most row and column of the destination table, respectively.

Examples

```
table1.copyrange B2 D4 table2 A1
```

places a copy of the data from cell range B2 to D4 in TABLE1 to TABLE2 at cell A1

```
table1.copyrange(t) 1 1 1 5 table2 1 3
```

copies 5 rows of data in the first column of data in table1 and places a transpose of the data at the top of the 3rd column of TABLE2.

Cross-references

See also `Table::copytable` (p. 1076).

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets," on page 57 of the *Command and Programming Reference*.

See also Chapter 17. "Table and Text Objects," on page 937 of *User's Guide I* for a discussion and examples of table formatting in EViews.

copytable — Table Procs

Copies the entire table to the specified location in another table.

Syntax

table_name.**copytable** *destname d1*

table_name.**copytable** *destname dr1 dc1*

The copytable command can be specified either using coordinates where columns are signified with a letter, and rows by a number (for example "A3" represents the first column, third row), or by row number and column number.

The first syntax represents coordinate form, where *destname* specifies the name of the table to copy to, and *d1* specifies the upper-left coordinate of the destination table.

The second syntax represents the row/column number form, where *destname* specifies the name of the table to copy to, and *dr1* and *dr2* specify the upper and left most row and column of the destination table, respectively.

Examples

```
table1.copytable table2 A10
```

copies all of the data in TABLE1 to the 1st column and 10th row of TABLE2.

```
table1.copytable table2 1 5
```

copies all of the data in TABLE1 to the 5th column and first row of TABLE2.

Cross-references

See also `Table::copyrange` (p. 1075).

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets," on page 57 of the *Command and Programming Reference*.

See also Chapter 17. "Table and Text Objects," on page 937 of *User's Guide I* for a discussion and examples of table formatting in EViews.

deletecells — Table Procs

Delete cells from a table.

Syntax

table_name.**deletecells**(*cell_range*) *delete_arg*

where *delete_arg* specifies what to delete or how the surrounding cells should be moved.

The *cell_range* argument defines the cells to be modified. See `Table::setformat` for the syntax.

delete_arg may be one of the following:

row	Delete row.
column	Delete column.

left	Shift cells to right of *cell_range* to the left after cells have been deleted.
up	Shift cells below *cell_range* up after cells have been deleted.

Examples

```
tab1.deletecells(2:5) row
```

deletes rows 2 through 5 from the table TAB1.

```
tab1.deletecells(B2:E5) left
```

deletes cells B2 through E5 from the table TAB1 and moves all the cells right of and including F2 and F5 to the left in place of B2 and B5 respectively.

```
tab1.deletecells(B2:E5) up
```

deletes cells B2 through E5 from the table TAB1 and moves all the cells below and including B5 and F5 up in place of B2 and E5 respectively.

You may delete cells conditionally. For example, to delete the cell e17 if cell b3 is less than c4 and shift the remaining cells in the row to the left:

```
tab1.deletecells(e17 if [b3]<[c4]) left
```

Or to delete the range of cells from c7 to d17 if cell b3 is greater than c4 and shift the remaining cells to the left:

```
tab1.deletecells(c7:d17 if [b3]>[c4]) left
```

Cross-references

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets" in the *Command and Programming Reference*.

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.deletecol(d) 2
```

removes two columns beginning at the "d" or fourth column.

Cross-references

For other row and columns operations, see `Table::deleterow` (p. 1079), `Table::insertcol` (p. 1083), and `Table::insertrow` (p. 1084).

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets," on page 57 of the *Command and Programming Reference*.

See also Chapter 17. "Table and Text Objects," on page 937 of *User's Guide I* for a discussion and examples of table formatting in EViews.

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 other row and columns operations, see `Table::deletecol` (p. 1078), `Table::insertcol` (p. 1083), and `Table::insertrow` (p. 1084).

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets," on page 57 of the *Command and Programming Reference*.

See also Chapter 17. "Table and Text Objects," on page 937 of *User's Guide I* for a discussion and examples of table formatting in EViews.

display	Table Views

Display table, graph, or spool output in the table object window.

Display the contents of a table, graph, or spool in the window of the table object.

Syntax

table_name.**display** *object_name*

Examples

```
table1.display tab1
```

Display the contents of the table TAB1 in the window of the object TABLE1.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names. See also `Table::label` (p. 1085).

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 123 of *User's Guide I* for a discussion of labels and display names.

See also `Table::label` (p. 1085).

fixcol	Table Procs

Fixes a set of columns to left of the spreadsheet view of a table object so that the leading columns are always in view.

Syntax

table_name.**fixcol** *cols*

where *cols* is the number of columns to be fixed

Example

```
tab1.fixcol 3
```

fixes the first 3 columns of the table TAB1 such that they are always in view despite the horizontal scroll position.

```
tab1.fixcol 0
```

removes any fixed columns in table TAB1.

Cross-references

See also `Table::fixrow` (p. 1081) and `Table::fixrowcol` (p. 1082).

fixrow	Table Procs

Fixes a set of rows at the top of the spreadsheet view of a table object so that the leading rows are always in view.

Syntax

table_name.**fixrow** *rows*

where *rows* is the number of rw to be fixed

Example

```
tab1.fixrow 2
```

fixes the first 2 rows of the table TAB1 such that they are always in view despite the vertical scroll position.

```
tab1.fixrow 0
```

removes any fixed rows in table TAB1.

Cross-references

See also `Table::fixcol` (p. 1081) and `Table::fixrowcol` (p. 1082).

fixrowcol	Table Procs

Fixes a set of rows at the top and a set of columns to left of a spreadsheet view of a table object so that the leading rows and columns are always in view.

Syntax

table_name.**fixrowcol** *rows cols*

where rows is the number of rows to be fixed and cols is the number of columns to be fixed.

Example

```
tab1.fixrowcol 1 4
```

fixes the first row and the first 4 columns of the table TAB1 such that they are always in view despite the horizontal and vertical scroll position of the table.

```
tab1.fixrowcol 0 0
```

removes all fixed rows and columns in table TAB1.

```
tab1.fixrowcol 0 4
```

in table TAB1 removes all fixed rows but fixes the first 4 columns.

Cross-references

See also `Table::fixcol` (p. 1081) and `Table::fixrow` (p. 1081).

insertcells	Table Procs

Inserts cells into a table.

Syntax

table_name.**insertcells**(*cell_range*) *insert_arg*

where *insert_arg* specifies what to be inserted or how the surrounding cells should be shifted.

The *cell_range* argument defines the cells to be modified. See `Table::setformat` for the syntax.

insert_arg may be one of the following:

row	Insert row.
column	Insert column.
right	Shift cells in *cell_range* to the right before inserting new cells.
down	Shift cells below *cell_range* down before inserting new cells.

Examples

```
tab1.insertcells(2:5) row
```

inserts four new rows starting at row 2 in the table TAB1.

```
tab1.insertcells(B2:E5) right
```

shifts the cells to the right of and including B2 through E5 in table TAB1 by four columns.

```
tab1.insertcells(B2:E5) down
```

shifts the cells below and including B2 through E5 in table TAB1 by four rows.

Cross-references

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets" in the *Command and Programming Reference*.

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

```
tab1.insertcol(d) 2
```

inserts two new columns beginning at the "d" or fourth column.

You may do conditional column insertions. This will insert four columns after each column from b to d if cell b7 is greater than each cell f7 to h7:

```
tab1.insertcol(b:d if [b7] > [f7:h7]) 4
```

Cross-references

For other row and columns operations, see `Table::deleterow` (p. 1079), `Table::deletecol` (p. 1078), and `Table::insertrow` (p. 1084).

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets," on page 57 of the *Command and Programming Reference*.

See also Chapter 17. "Table and Text Objects," on page 937 of *User's Guide I* for a discussion and examples of table formatting in EViews.

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

```
tab1.insertrow(2) 5
```

inserts five new rows beginning at the second row.

You may do conditional row insertions. This will insert four rows after each row from b to d if cell b7 is greater than each cell f7 to h7:

```
tab1.insertrow(b:d if [b7] > [f7:h7]) 4
```

Cross-references

For other row and columns operations, see `Table::deleterow` (p. 1079), `Table::deletecol` (p. 1078), and `Table::insertcol` (p. 1083).

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets," on page 57 of the *Command and Programming Reference*.

See also Chapter 17. "Table and Text Objects," on page 937 of *User's Guide I* for a discussion and examples of table formatting in EViews.

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

```
tab1.insertrow(2) 5
```

inserts five new rows beginning at the second row.

Cross-references

For other row and columns operations, see `Table::deleterow` (p. 1079), `Table::deletecol` (p. 1078), and `Table::insertcol` (p. 1083).

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets," on page 57 of the *Command and Programming Reference*.

See also Chapter 17. "Table and Text Objects," on page 937 of *User's Guide I* for a discussion and examples of table formatting in EViews.

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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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:

```
tab1.label(u) Millions of bushels
```

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels.

See also `Table::displayname` (p. 1080).

olepush — Table Procs

Push updates to OLE linked objects in open applications.

Syntax

table_name.**olepush**

Cross-references

See Chapter 19. "Object Linking and Embedding (OLE)," on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

save	Table Procs

Save table to disk as an Excel 2007 XLSX, CSV, tab-delimited ASCII text, RTF, HTML, Enhanced Metafile, LaTeX, PDF, or Markdown 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.

The base syntax for writing Excel files is:

table_name.**save**(*options*) *[path\]file_name [table_description]*

where the following *table_description* elements may be employed:

- "range = *arg*", where arg is top left cell of the destination 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.

Options

t = *file_type* (*default* = "csv")	Specifies the file type, where *file_type* may be one of: "excelxml" (Excel 2007 (xml)),"csv" (CSV - comma-separated), "rtf" (Rich-text format), "txt" (tab-delimited text), "html" (HTML - Hypertext Markup Language), "emf" (Enhanced Metafile), "pdf" (PDF - Portable Document Format), "tex" (LaTeX), or "md" (Markdown). Files will be saved with the ".xlsx", ".csv", ".rtf", ".txt", ".htm", ".emf", ".pdf", ".tex", or ".md" extensions, respectively.
s = *arg*	Scale size, where *arg* is from 5 to 200, representing the percentage of the original table size (only valid for HTML or RTF files).
r = *cell_range*	Range of table cells to be saved. See `Table::setfillcolor` (p. 1091) for the *cell_range* syntax. If a range is not provided, the entire table will be saved.
n = *string*	Replace all cells that contain NA values with the specified string. "NA" is the default.
f / -f	[Use full precision values/ Do not use full precision] when saving values to the table (only applicable to numeric cells). By default, the values will be saved as they appear in the currently formatted table.
prompt	Force the dialog to appear from within a program.

PDF Options

landscape	Save in landscape mode (the default is to save in portrait mode).
size = *arg* (*default* = "letter")	Page size: "letter", "legal", "a4", and "custom".
width = *number* (*default* = 8.5)	Page width in inches if "size = custom".
height = *number* (*default* = 11)	Page height in inches if "size = custom".
leftmargin = *number* (*default* = 0.5)	Left margin width in inches.
rightmargin = *number* (*default* = 0.5)	Right margin width in inches.

topmargin = *number (default* = 1*)*	Top margin width in inches.
bottommargin = *number (default* = 1*)*	Bottom margin width in inches.

LaTeX Options

texspec / -texspec	[Include / Do not include] the full LaTeX documentation specification in the LaTeX output. The default behavior is taken from the global default settings.

Excel Options

mode = *arg*	Specify whether to create a new file, overwrite an existing file, or update an existing file. arg may be "create" (create new file only; error on attempt to overwrite) or "update" (update an existing file, only overwriting the area specified by the range = *table_description*). If the "mode = " option is not used, EViews will create a new file, unless the file already exists in which case it will overwrite it. Note that the "mode = update" option is only available for Excel in 1) Excel versions through 2003, if Excel is installed, and 2) Excel 2007 (xml). Note: Excel does not need to be installed for Excel 2007 writing.

Excel 2007 Options

cellfmt = *arg*	Specify whether to use EViews, pre-existing, or remove cell formatting (colors, font, number formatting when possible, column widths and row heights) for the written range. *arg* may be "eviews" (replace current formatting in the file with the same cell formatting in EViews), "preserve" (leave current cell formatting already in the Excel file), or "clear" (remove current formatting and do not replace).
strlen = *arg* (default = 256)	Specify the maximum the number of characters written for cells containing text. Strings in cells which are longer the max, will be truncated.

Examples

The command:

```
tab1.save mytable
```

saves TAB1 to a CSV file named "mytable.CSV" in the default directory.

```
tab1.save(t=csv, n="NaN") mytable
```

saves TAB1 to a CSV (comma separated value) file named “mytable.csv” and writes all NA values as “NaN”.

```
tab1.save(r=B2:C10, t=html, s=50) 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.

```
tab1.save(f, n=".", r=B) mytable
```

saves the contents of the second column of the table in full precision to a CSV file named “mytable.CSV”, and writes all NA values as “.”.

```
tab1.save(t=excelxml, cellfmt=eviews, mode=update) mytable
   range=Country!b5
```

adds TAB1 to the preexisting “mytable.XLSX” Excel file to the “Country” sheet at cell B5, where the cell colors and fonts in TAB1 will also be copied.

Cross-references

For additional discussion of table commands see Chapter 3. “Working with Tables and Spreadsheets,” on page 57 of the *Command and Programming Reference*.

See Chapter 17. “Table and Text Objects,” beginning on page 937 of *User’s Guide I* for a discussion of tables.

setattr	Table Procs

Set the object attribute.

Syntax

table_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the “revised” attribute in the object A to the string “never”, and extracts the attribute into the string object S.

Cross-references

See “Adding Custom Attributes in the Label View” on page 123 and “Adding Your Own Label Attributes” on page 70 of *User’s Guide I*.

setfillcolor	Table Procs

Set the fill (background) color of the specified table cells.

Syntax

table_name.**setfillcolor**(*cell_range, options*) *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 number (e.g., “R1C2”).You can optionally add an ‘if’ condition to the identifier. Without the ‘if’ condition, the formatting will be applied to all the identified cells. If an ‘if’ condition is supplied, the formatting will be applied to all the identified where the ‘if’ condition is true. The ‘if’ condition must be followed by a boolean expression.
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.
col	Column *letter* (*e.g.*, “B”). Apply to all cells in the column.
first_cell[:]*last_cell*, *first_cell*[,]*last_cell*	Top left cell of the selection range (specified in “*cell*” format), followed by bottom right cell of the selection range (specified in “*cell*” format), separated by 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.
first_cell_row[,] *first_cell_col*[,] *last_cell_row*[,] *last_cell_col*	Top left cell of the selection range (specified in “*row*[,] *col*” format), followed by bottom right cell of the selection range (specified in “*row*[,] *col*” format), separated by a “,” (*e.g.*, “2,A,10,C” or “2,1,10,3”). Apply to all cells in the rectangular region defined by the first cell and last cell.

The *color_arg* specifies the color to be applied to the text in the cells. The color may be specified using predefined color names, by specifying the individual red-green-blue (RGB) components using the special “@RGB” function, or by specifying the individual red-green-blue (RGB) components in hexadecimal using the special “@HEX” function. The second and third methods are obviously more difficult, but allow you to use custom colors.

The predefined colors are given by the keywords (with their RGB and HEX equivalents):

blue	@rgb(0, 0, 255)	@hex(0000ff)
red	@rgb(255, 0, 0)	@hex(ff0000)
ltred	@rgb(255, 168, 168)	@hex(ffa8a8)
green	@rgb(0, 128, 0)	@hex(008000)
black	@rgb(0, 0, 0)	@hex(000000)
white	@rgb(255, 255, 255)	@hex(ffffff)
purple	@rgb(128, 0, 128)	@hex(800080)
orange	@rgb(255, 128, 0)	@hex(ff8000)
yellow	@rgb(255, 255, 0)	@hex(ffff00)
gray	@rgb(128, 128, 128)	@hex(808080)
ltgray	@rgb(192, 192, 192)	@hex(c0c0c0)

Options

mode=*arg*	Color mode: "Alt" (alternate and color every other row in the *cell_range*); "All" (color every row in the *cell_range*).

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) @HEX(808080)
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:

```
tab1.setfillcolor(C) @RGB(255, 255, 0)
tab1.setfillcolor(2) yellow
```

or for every other row, the background of the cells in a rectangular region:

```
tab1.setfillcolor(R2C3:R3C6, mode=alt) ltgray
tab1.setfillcolor(2,C,3,F, mode=alt) @rgb(192, 192, 192)
tab1.setfillcolor(2,3,3,6, mode=alt) @hex(c0c0c0)
```

You may conditionally set the fill color for a range of cells. For example, in the below command cells a3 to a18 will be yellow if the sum from a second range of cells (b4 to b19) added to a third range of cells (c4 to c19) is greater than 0.5. Specifically, cell a3 will be set to yellow if cell b4 plus cell c4 is greater than 0.5:

```
tab1.setfillcolor(a3:a18 if [b4:b19]+[c4:c19] > .5) yellow
```

Cross-references

See `Table::settextcolor` (p. 1105) and `Table::setfont` (p. 1093) for details on changing text color and font, and `Table::setlines` (p. 1101) for drawing lines between and through cells.

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets," on page 57 of the *Command and Programming Reference*.

See also Chapter 17. "Table and Text Objects," on page 937 of *User's Guide I* for a discussion and examples of table formatting in EViews.

setfont	Table Procs

Set the font for text in the specified table cells.

Syntax

table_name.**setfont**(*cell_range*) *font_args*

The *font_args* may include one or more of the following:

*type([face], [pt], [+/- **b**], [+/- **i**], [+/- **u**], [+/- **s**])*	Set characteristics of the font for the graph element *type*. The font name *(face)*, size *(pt)*, and characteristics are all optional. *face* should be a valid font name, enclosed in double quotes. *pt* 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.

and *type* is one of "all", "axes", "legend", "text", "obs", where "axes" refers to the axes labels, "legend" refers to the graph legend, "text" refers to the text objects, "obs" refers to the observation scale, and "all" refers to all of the elements.

Examples

```
tab1.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).

```
tab1.setfont(3,B,10,D) 8pt
```

changes all of text in the region to 8 point.

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

removes the italic, and adds boldface to the B4 cell (row 4, column 2).

The commands:

```
tab1.setfont(b) -s +u 14pt
tab1.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

See `Table::settextcolor` (p. 1105) and `Table::setfillcolor` (p. 1091) for details on changing text color and font, and `Table::setlines` (p. 1101) for drawing lines between and through cells.

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets," on page 57 of the *Command and Programming Reference*.

See also Chapter 17. "Table and Text Objects," on page 937 of *User's Guide I* for a discussion and examples of table formatting in EViews.

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:

@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., "A2"), or by using "R" followed by the row number followed by "C" and the column number (e.g., "R1C2").You can optionally add an 'if' condition to the identifier. Without the 'if' condition, the formatting will be applied to all the identified cells. If an 'if' condition is supplied, the formatting will be applied to all the identified where the 'if' condition is true. The 'if' condition must be followed by a boolean expression.
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.
col	Column *letter* (*e.g.*, "B"). Apply to all cells in the column.
first_cell[:]*last_cell*, *first_cell*[,]*last_cell*	Top left cell of the selection range (specified in "*cell*" format), followed by bottom right cell of the selection range (specified in "*cell*" format), separated by 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.
first_cell_row[,] *first_cell_col*[,] *last_cell_row*[,] *last_cell_col*	Top left cell of the selection range (specified in "*row*[,] *col*" format), followed by bottom right cell of the selection range (specified in "*row*[,] *col*" format), separated by a "," (*e.g.*, "2,A,10,C" or "2,1,10,3"). Apply to all cells in the rectangular region defined by the first cell and last cell.

To format numeric values, you should use one of the following format specifications:

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

To specify a format that groups digits into thousands using a comma separator, place a "t" after the format character. For example, to obtain a fixed number of decimal places with commas used to separate thousands, use "ft*[.precision]*".

To use the period character to separate thousands and commas to denote decimal places, use ".." (two periods) when specifying the precision. For example, 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)"), 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 106 of the *Command and Programming Reference*). 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 period 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"
```

You may conditionally set the format for a range of cells. For example, in the command below the cells from b20 to d25 will use format 'e.5' if the matching cells from the second range (b20 to d25) are greater than the cells from the third range (f10 to h15). Specifically, the format for cell b20 will be set to 'e.5' if cell b20 is greater than f10:

```
tab1.setformat(b20:d25 if [b20:d25] > [f10:h15]) e.5
```

Cross-references

See `Table::settextcolor` (p. 1105) and `Table::setfillcolor` (p. 1091) for details on changing text color, and `Table::setlines` (p. 1101) for drawing lines between and through cells. To set other cell properties, see `Table::setheight` (p. 1098), `Table::setindent` (p. 1099), `Table::setjust` (p. 1100), and `Table::setwidth` (p. 1106).

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets," on page 57 of the *Command and Programming Reference*.

See also Chapter 17. "Table and Text Objects," on page 937 of *User's Guide I* for a discussion and examples of table formatting in EViews.

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.

You many conditionally set the row height. This command will set the individual row height of rows 10 through 15 to 2 if the cells in column b (cells b20 to b25) are greater than the cells in column f (cells f10 to f15). Specifically, the row height of row 10 will be set to 2 if cell b20 is greater than f10:

```
tab1.setheight(10:15 if [b20:b25] > [f10:f15]) 2
```

Cross-references

See `Table::setwidth` (p. 1106), `Table::setindent` (p. 1099) and `Table::setjust` (p. 1100) for details on setting table widths, indentation and justification.

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets," on page 57 of the *Command and Programming Reference*.

See also Chapter 17. "Table and Text Objects," on page 937 of *User's Guide I* for a discussion and examples of table formatting in EViews.

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 values are taken from the settings at the time the table is created.

The *cell_range* defines the cells to be modified. See `Table::setformat` (p. 1094) 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

See Table::setwidth (p. 1106), Table::setheight (p. 1098) and Table::setjust (p. 1100) for details on setting table widths, height, and justification.

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets," on page 57 of the *Command and Programming Reference*.

See also Chapter 17. "Table and Text Objects," on page 937 of *User's Guide I* for a discussion and examples of table formatting in EViews.

setjust	Table Procs

Set the justification for cells in the table.

Syntax

table_name.**setjust**(*cell_range*) *format_arg*

where *format_arg* may consist of the following:

top / middle / bottom	Vertical justification setting.
auto / left / center / right	Horizontal justification setting. Strings are left-justified and numbers are right-justified under "auto".

The default settings are taken from the original view when created by freezing a view, or as "middle bottom" for newly created tables.

The *cell_range* defines the cells to be modified. See Table::setformat (p. 1094) for syntax.

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

See Table::setwidth (p. 1106), Table::setheight (p. 1098), and Table::setindent (p. 1099) for details on setting table widths, height and indentation.

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets," on page 57 of the *Command and Programming Reference*.

See also Chapter 17. "Table and Text Objects," on page 937 of *User's Guide I* for a discussion and examples of table formatting in EViews.

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. 1091) for the syntax for *cell_range*.

The *line_args* may contain one or more of the following:

+t / -t	Top border [on/off].
+b / -b	Bottom border [on/off].
+l / -l	Left border [on/off].
+r / -r	Right border [on/off].
+i / -i	Inner borders [on/off].
+o / -o	Outer borders [on/off].
+v / -v	Vertical inner borders [on/off].
+h / -h	Horizontal inner borders [on/off].
+a / -a	All borders [on/off].
+d / -d	Double middle lines [on/off].

Examples

```
tab1.setlines(b2:d6) +o
```

draws borders around the outside of the rectangle defined by B2 and D6. Note that this command is equivalent to:

```
tab1.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.

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

puts a border around all four sides of the B2 cell.

```
tab1.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.

```
tab1.setlines(@all) -a
```

removes all borders from the table.

Cross-references

See `Table::settextcolor` (p. 1105), `Table::setfillcolor` (p. 1091), and `Table::setfont` (p. 1093) for details on changing text color and font.

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets," on page 57 of the *Command and Programming Reference*.

See also "Table Objects" on page 937 of *User's Guide I* for a discussion and examples of table formatting in EViews.

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:

first_cell[:]*last_cell*, *first_cell*[,]*last_cell*	Left (right) cell of the selection range (specified in "*cell*" format), followed by right (left) cell of the selection range (specified in "*cell*" 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.
cell_row[,] *first_cell_col*[,] *cell_row*[,] *last_cell_col*	Left (right) cell of the selection range (specified in "*row*[,] *col*" format), followed by right (left) cell of the selection range (specified in "*row*[,] *col*" format, with a fixed *row*), 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

See `Table::setwidth` (p. 1106), `Table::setheight` (p. 1098) and `Table::setjust` (p. 1100) for details on setting table widths, height, and justification.

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets," on page 57 of the *Command and Programming Reference*.

See also "Table Objects" on page 937 of *User's Guide I* for a discussion and examples of table formatting in EViews.

setprefix	Table Procs

Set the cell prefix string for the specified table cells.

Syntax

table_name.**setprefix**(*cell_range*) *prefix*

where *prefix* is the prefix you wish to assign to the cells. To remove a prefix from a cell, leave *prefix* empty.

The *cell_range* defines the cells to be modified. See `Table::setformat` (p. 1094) for the syntax for *cell_range* specifications.

Examples

```
tab1.setprefix(A1) $
```

prepends the dollar sign ($) to the cell A1.

```
tab1.setprefix(A1)
```

removes the prefix from cell A1.

Cross-references

See `Table::setwidth` (p. 1106), `Table::setindent` (p. 1099) and `Table::setjust` (p. 1100) for details on setting table widths, indentation and justification.

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets," on page 57 of the *Command and Programming Reference*.

See also Chapter 17. "Table and Text Objects," on page 937 of *User's Guide I* for a discussion and examples of table formatting in EViews.

setsuffix	Table Procs

Set the cell suffix string for the specified table cells.

Syntax

table_name.**setsuffix**(*cell_range*) *suffix*

where *suffix* is the suffix you wish to assign to the cells. To remove a suffix from a cell, leave *suffix* empty.

The *cell_range* defines the cells to be modified. See `Table::setformat` (p. 1094) for the syntax for *cell_range* specifications.

Examples

```
tab1.setsuffix(A1) $
```

appends the dollar sign ($) to the cell A1.

```
tab1.setsuffix(A1)
```

removes the suffix from cell A1.

Cross-references

See `Table::setwidth` (p. 1106), `Table::setindent` (p. 1099) and `Table::setjust` (p. 1100) for details on setting table widths, indentation and justification.

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets," on page 57 of the *Command and Programming Reference*.

See also Chapter 17. "Table and Text Objects," on page 937 of *User's Guide I* for a discussion and examples of table formatting in EViews.

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. 1091) 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) @HEX(ff8000)
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) @HEX(00ff00)
```

You may also conditionally set the text color for a range of cells. For example, cell d3 will be set to red if cell b4 plus c4 is greater than cell d4 minus e4, and similarly for cells d4 to d18.

```
tab1.settextcolor(d3:d18 if [b4:b19]+[c4:c19] > [d4:d19]-[e4:e19])
    red
```

This also works for fixed columns and rows. To set the text color for the cells a13 to b18 to red if the matching cells from the fixed column B (cells b4 to b9) are greater than 0.5:

```
tab1.settextcolor(a13:b18 if [b4:b9] > .5) red
```

This sets the text color for the cells e13 to f18 to orange if the matching cells from fixed row 4 (cells b4 to c4) are greater than 0.5. Specifically, the text color of cells e13 through e18 will

be orange if cell b4 is greater than 0.5, and the text color of cells f13 through f18 will be orange if cell c4 is greater than 0.5:

```
tab1.settextcolor(e13:f18 if [b4:c4] > .5) orange
```

Cross-references

See `Table::setfont` (p. 1093) and `Table::setfillcolor` (p. 1091) for details on changing the text font and cell background color.

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets," on page 57 of the *Command and Programming Reference*.

See also Chapter 17. "Table and Text Objects," on page 937 of *User's Guide I* for a discussion and examples of table formatting in EViews.

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.

You many conditionally set the column width. This will set the individual column widths of columns c through e to 5 if the cells in column b are greater than 0.5. Specifically, the width of column c will be set to 5 if cell b20 is greater than 0.5:

```
tab1.setwidth(c:e if [b20:d20] > .5) 5
```

Cross-references

See `Table::setheight` (p. 1098), `Table::setindent` (p. 1099) and `Table::setjust` (p. 1100) for details on setting table height, indentation and justification.

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets," on page 57 of the *Command and Programming Reference*.

See also Chapter 17. "Table and Text Objects," on page 937 of *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.

sort	Table Procs

Sort the selected rows of a table object.

Syntax

table_name.**sort**(*cell_range*) *column_arg*

where *column_arg* is a list of columns by which to sort the *cell_range*. You may specify up to three columns for sorting. The specified columns must be within the *cell_range*. If you list two or more columns, sort uses the values of the second column to resolve ties in the first column, and values of the third column to resolve ties in the first and second. By default, EViews will sort in ascending order. To sort in descending order, precede the column name with a minus sign ("-").

The *cell_range* defines the cells to be modified. See `Table::setformat` (p. 1094) for the syntax for *cell_range* specifications.

Examples

```
tab1.sort(a1:c20) b
```

sorts the cells from the 1st column 1st row to the 3rd column 20th row by column b in ascending order.

```
tab1.sort(b10:z250) -f h q
```

sorts the cells from the 2nd column 10th row to the 26th column 250th row by column f in descending order, then in ascending order by column h and once again in ascending order by column q.

Cross-references

See `Table::setformat` (p. 1094) for the syntax for *cell_range* specifications.

table	Table Declaration | Table Views

Declare a table object.

The `table` command declares and optionally sizes a table object. When used as a table view, `table` displays the contents of the table.

Syntax

table*(rows, cols) table_name*

table_name.**table***(options)*

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.

The `table` view displays the contents of the table. It is a synonym for `sheet` (p. 1107).

Examples

```
table onelement
```

declares a one element table

```
table(10,5) outtable
```

creates a table OUTTABLE with 10 rows and 5 columns.

Cross-references

See also `freeze` (p. 457) of the *Command and Programming Reference* and `Table::sheet` (p. 1107).

For additional discussion of table commands see Chapter 3. "Working with Tables and Spreadsheets," on page 57 of the *Command and Programming Reference*.

See Chapter 17. "Table and Text Objects," on page 937 of *User's Guide I* for a general discussion and examples of table formatting in EViews.

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 `Table::displayname` (p. 1080) and `Table::label` (p. 1085).

transpose	Table Procs

Transposes a set of cells in the table.

Syntax

```
table_name.transpose(cell_range)
```

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 number (e.g., "R1C2").
row[,] col	Row number, followed by column letter or number, separated by "," (e.g., "2,C", or "2,3"). Apply to cell.
row	Row number (e.g., "2"). Apply to all cells in the row.
col	Column letter (e.g., "B"). Apply to all cells in the column.
first_cell[:]last_cell, *first_cell[,]last_cell*	Top left cell of the selection range (specified in "cell" format), followed by bottom right cell of the selection range (specified in "cell" format), separated by 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.
first_cell_row[,] first_cell_col[,] last_cell_row[,] last_cell_col	Top left cell of the selection range (specified in "row[,] col" format), followed by bottom right cell of the selection range (specified in "row[,] col" format), separated by a "," (e.g., "2,A,10,C" or "2,1,10,3"). Apply to all cells in the rectangular region defined by the first cell and last cell.

Examples

To rotate the cells in column 1 to row 1 in table TAB1:

```
tab1.transpose(A)
```

To rotate all of the cells in TAB1:

```
tab1.transpose(@all)
```

To rotate the cells in range B11:C20 to range B11:K12:

```
tab1.transpose(b11:c20)
```

Text

Text object.

Object for holding arbitrary text information.

Text Declaration

text ..... declare text object (p. 1120).

To declare a text object, use the keyword `text`, followed by the object name:

```
text mytext
```

Text Views

display ..... display table, graph, or spool in object window (p. 1116).
label..... label information for the text object (p. 1117).
text ..... view contents of text object (p. 1120).

Text Procs

append ..... appends text to the end of a text object (p. 1113).
clear..... clear a text object (p. 1114).
clearhist ..... clear the contents of the history attribute (p. 1114).
clearremarks ..... clear the contents of the remarks attribute (p. 1115).
copy..... creates a copy of the text (p. 1115).
displayname..... changes the display name for the text object (p. 1116).
olepush ..... push updates to OLE linked objects in open applications (p. 1118).
save ..... save text object to disk as an ASCII text, RTF, HTML, PDF, TEX, or Markdown file (p. 1118).
setattr..... set the value of an object attribute (p. 1119).
svector ..... make svector out of the contents of the text object (p. 1119).

Text Data Members

Scalar Values

@linecount ..... scalar containing the number of lines in a Text object.

String values

@attr("*arg*") ..... string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@description..... string containing the Text object's description (if available).
@detailedtype ..... string with the object type: "TEXT".
@displayname..... string containing the Text object's display name. If the object has no display name set, the name is returned.
@line(i) ..... returns a string containing the Text on i-th line of the Text object.
@name ..... string containing the Text object's name.

@remarks.....string containing the Text object's remarks (if available).
@svector.....returns an Svector where each element is a line of the Text object.
@svectornb.....same as `@svector,` with blank lines removed.
@type.....string with the object type: "TEXT".
@updatetime .....string representation of the time and date at which the Text object was last updated.

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.

append	Text Procs

Appends text or a text file to the end of a text object.

There are different forms of the command, with the syntax depending on whether you are appending a line of text or the contents of a text file to the end of the text object.

Syntax

text_name.**append***"text to append"*
text_name.**append(file)** *[path\]file_name*

Specify the literal text or file name after the `append` keyword.

Examples

```
tt1.append "Add this to the end"
```

appends the text "Add this to the end" at the end of the text object TT1.

To include quotes in the string, use the quote escape sequence, or double quotes:

```
tt1.append """This is a quoted string"""
```

appends "This is a quoted string".

You may also use curly braces with a string object:

```
string s = """This is a quoted string"""
tt1.append {s}
```

appends “This is a quoted string”.

```
tt1.append(file) c:\myfile\file.txt
```

appends the contents of the text file “File.TXT” to the text object.

Cross-references

See also `Text::clear` (p. 1114).

clear	Text Procs

Clear a text object.

Syntax

text_name.**clear**

Examples

The following command clears all text from the text object TT1:

```
tt1.clear
```

Cross-references

See also `Text::append` (p. 1113).

clearhist	Text Procs

Clear the contents of the history attribute for text objects.

Removes the text’s history attribute, as shown in the label view of the text.

Syntax

text_name.**clearhist**

Examples

```
t1.clearhist
t1.label
```

The first line removes the history from the text T1, and the second line displays the label view of T1, including the now blank history field.

Cross-references

See “Labeling Objects” on page 123 of the *User’s Guide I* for a discussion of labels and display names.

See also `Text::label` (p. 1117).

clearremarks	Text Procs

Clear the contents of the remarks attribute.

Removes the text's remarks attribute, as shown in the label view of the text.

Syntax

text_name.**clearremarks**

Examples

```
t1.clearremarks
t1.label
```

The first line removes the remarks from the text T1, and the second line displays the label view of T1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Text::label` (p. 1117).

copy	Text Procs

Creates a copy of the text.

Creates either a named or unnamed copy of the text.

Syntax

text_name.**copy**

text_name.**copy** *dest_name*

Examples

```
t1.copy
```

creates an unnamed copy of the text T1.

```
t1.copy t2
```

creates T2, a copy of the text T1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

display	Text Views

Display table, graph, or spool output in the text object window.

Display the contents of a table, graph, or spool in the window of the text object.

Syntax

text_name.**display** *object_name*

Examples

```
text1.display tab1
```

Display the contents of the table TAB1 in the window of the object TEXT1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

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 123 of *User's Guide I* for a discussion of labels and display names.

See also `Text::label` (p. 1117).

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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User’s Guide I* for a discussion of labels.

See also `Text::displayname` (p. 1116).

olepush	Text Procs

Push updates to OLE linked objects in open applications.

Syntax

text_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

save	Text Procs

Save text object to disk as an ASCII text, RTF, HTML, PDF, TEX, or MD file.

Syntax

text_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.

The MD (Markdown) setting uses very basic syntax and should be usable in most editors.

Options

t = *file_type* *(default* = "txt"*)*	Specifies the file type, where *file_type* may be one of: "rtf" (Rich-text format), "txt" (ASCII text), or "html" (HTML - Hypertext Markup Language), "pdf" (Portable Document Format, PDF), "tex" (LaTeX), or "md" (Markdown). Files will be saved with the ".rtf", ".txt", "html", "pdf", "tex", or "md" extensions, respectively.
dropempty	Removes empty lines from output (empty lines are included by default).

Examples

The command:

```
text1.save mytext
```

saves TEXT1 to an ASCII text file named “MYTEXT.TXT” in the default directory.

```
text1.save mytext.bat
```

saves TEXT1 to an ASCII text file using the explicitly provided name “MYTEXT.BAT”.

```
text1.save(t=rtf, dropempty) mytext
```

saves TEXT1 (excluding any empty lines) to the RTF file “MYTEXT.RTF”.

Cross-references

See Chapter 17. “Table and Text Objects,” beginning on page 937 of *User’s Guide I* for a discussion of tables.

setattr	Text Procs

Set the object attribute.

Syntax

text_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the “revised” attribute in the object A to the string “never”, and extracts the attribute into the string object S.

Cross-references

See “Adding Custom Attributes in the Label View” on page 123 and “Adding Your Own Label Attributes” on page 70 of *User’s Guide I*.

svector	Text Procs

Make an svector out of the contents of the text object.

Syntax

text_name.**svector** *name*

Makes an svector called name, where each row of the svector is equal to a line of the text object.

Examples

```
text01.svector svec
```

makes an svector named SVEC.

Cross-references

See "String Vectors" on page 102 of the *Command and Programming Reference* for a discussion of strings and string vector. See also "Svector" on page 966.

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 "Text Objects" on page 952 of *User's Guide I* for a discussion of text objects in EViews.

Userobj

Userobj (user-defined object).

User Object Declaration

userobj ..... declare an empty, unregistered user object (p. 1131).

A simple, non-registered, user object is created by simply using the `userobj` command followed by the name of the user object:

```
userobj myuserobject
```

User Object Views

Although a registered user object may have user-defined views available, all user objects have the following built-in views.

display ..... display table, graph, or spool output in the user object window (p. 1126).
label ..... display or change the label view of a user object (p. 1128).
members ..... display a list of the members of a user object (p. 1129).

User Object Procs

Although a registered user object may have user-defined procs available, all user objects have the following built-in procs.

add ..... add a data or object member to the user object (p. 1123).
clear ..... remove all members from the user object (p. 1124).
clearhist ..... clear the contents of the history attribute (p. 1125).
clearremarks ..... clear the contents of the remarks attribute (p. 1125).
copy ..... creates a copy of the user object (p. 1126).
displayname ..... attach a display name to the user object (p. 1127).
drop ..... drop a data or object member from the user object (p. 1127).
extract ..... display or copy a data member from the user object (p. 1128).
label ..... display or change the label view of a user object (p. 1128).
olepush ..... push updates to OLE linked objects in open applications (p. 1130).
setattr ..... set the value of an object attribute (p. 1130).

User Object Data Members

Although user objects can have user-defined data members, the following built-in data members also exist for all user objects.

String values

@attr("*arg*") ..... string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@description ..... string containing the user object's description (if available).

@detailedtype .....string with the object type: “USEROBJ”.
@displayname .....string containing the user object’s display name. If the user object has no display name set, the name is returned.
@members .....space delimited list of all the user-defined members currently stored inside the user object.
@name.....string containing the user object’s name.
@remarks.....string containing the user object’s remarks (if available).
@type.....string with the object type: “USEROBJ”.
@updatetime .....string representation of the time and date at which the user object was last updated.

Scalar values

@hasmember(“*name*”) ... returns a 1 or a 0 depending on whether the user object has a data member called *name*.

User Object Entries

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

add	User Object Procs

Add a data or object member to the user object.

Adds a new data or object member to the user object. You may either create a new string or scalar member directly, or copy an existing object from the current workfile page. Note that only view objects (tables, graphs, text objects or spools) or matrix objects can be copied to a user object as members.

Syntax

userobject_name.**add**(*options*) *member arg*
userobject_name.**add**(*options*) *[member] objname*

If creating a new member inside the user object, you should specify the name of the member with member, and then specify its value with arg. If arg is a number, the new member will be created as a scalar, if arg is a string, the new member will be a string.

If copying the member from the current workfile page, you should use *objname* to specify the name of the object in the workfile you wish to copy. If you would like to give the member a different name inside the user object, you can specify that name with *member*.

Options

r	Replace an existing member. If this option is not used, and a member with the same name already exists, EViews will error.
d	When copying an object from the workfile page as the data member, delete the object from the workfile after copying.

Examples

```
myobj.add mymember 3
```

Creates a new member inside the user object MYOBJ called MYMEMBER, and sets its value equal to 3.

```
myobj.add(r) mymember "hello"
```

Replaces the member MYMEMBER with a string value of "hello".

```
myobj.add matv
```

Creates a new member called MATV by copying the existing workfile object MATV into the user object.

```
myobj.add(d) mymat matm
```

Creates a new member called MYMAT by copying the workfile object MATM into the user object. MATM is deleted from the workfile.

Cross-references

See Chapter 9. "User Objects," on page 233 of *Command and Programming Reference* for discussion of user objects.

See also `Userobj::clear` (p. 1124), `Userobj::drop` (p. 1127), and `Userobj::members` (p. 1129).

clear	User Object Procs

Removes all members from the user object.

Syntax

userobject_name.**clear**

Examples

```
myuserobj.clear
```

Deletes all members from the user object MYUSEROBJ.

Cross-references

See Chapter 9. "User Objects," on page 233 of *Command and Programming Reference* for discussion of user objects.

See also `Userobj::add` (p. 1123) and `Userobj::drop` (p. 1127).

clearhist	User Object Procs

Clear the contents of the history attribute.

Removes the user object's history attribute, as shown in the label view of the user object.

Syntax

userobj_name.**clearhist**

Examples

```
u1.clearhist
u1.label
```

The first line removes the history from the user object U1, and the second line displays the label view of U1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Userobj::label` (p. 1128).

clearremarks	User Object Procs

Clear the contents of the remarks attribute.

Removes the user object's remarks attribute, as shown in the label view of the user object.

Syntax

userobj_name.**clearremarks**

Examples

```
u1.clearremarks
u1.label
```

The first line removes the remarks from the user object U1, and the second line displays the label view of U1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Userobj::label` (p. 1128).

copy	User Object Procs

Creates a copy of the userobj.

Creates either a named or unnamed copy of the user object.

Syntax

userobj_name.**copy**

userobj_name.**copy** *dest_name*

Examples

```
u1.copy
```

creates an unnamed copy of the user object U1.

```
u1.copy u2
```

creates U2, a copy of the user object U1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

display	User Object Views

Display table, graph, or spool output in the user object window.

Display the contents of a table, graph, or spool in the window of the user object.

Syntax

userobject_name.**display** *object_name*

Examples

```
uo1.display tab1
```

Display the contents of the table TAB1 in the window of the object UO1.

Cross-references

See "Custom Object Output" on page 231 of *Command and Programming Reference*.

displayname	User Object Procs

Display name for user objects.

Attaches a display name to a user object.

Syntax

userobject_name.**displayname** *display_name*

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

Examples

```
hrs.displayname Hours Worked
hrs.label
```

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

Cross-references

See “Labeling Objects” on page 123 of *User’s Guide I* for a discussion of labels and display names.

drop	User Object Procs

Removes a member from the user object.

Syntax

userobject_name.**drop** *member*

Removes the member *member* from the user object.

Examples

```
myuserobj.drop mymember
```

Deletes the member MYMEMBER from the user object MYUSEROBJ.

Cross-references

See Chapter 9. “User Objects,” on page 233 of *Command and Programming Reference* for discussion of user objects.

See also `Userobj::add` (p. 1123), `Userobj::clear` (p. 1124), and `Userobj::members` (p. 1129).

extract	User Object Procs

Displays or copies a member from the user object.

Syntax

userobject_name.**extract** *member* [*wfname*]

Copies the data member specified by *member* into the current workfile page. If *wfname* is not specified, a new untitled object will be created, allowing you to quickly inspect the contents of the data member. If *wfname* is given, a new object in the workfile will be created with a name equal to *wfname*..

Examples

```
myuserobj.extract mymember
```

copies the data member MYMEMBER as a new untitled object in the workfile.

```
myuserobj.extract mymember xx
```

copies the data member MYMEMBER as a new object in the workfile called XX.

Cross-references

See Chapter 9. "User Objects," on page 233 of *Command and Programming Reference* for discussion of user objects.

See also `Userobj::add` (p. 1123) and `Userobj::members` (p. 1129).

label	User Object Views | User Object Procs

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

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

Syntax

userobject_name.**label**

userobject_name.**label***(options) [text]*

Options

The first version of the command displays the label view of the user 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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* to the remarks field as an additional line.
p	Print the label view.

Examples

The following lines replace the remarks field of user object UO1 with “Data from CPS 1988 March File”:

```
UO1.label(r)
UO1.label(r) Data from CPS 1988 March File
```

To append additional remarks to UO1, and then to print the label view:

```
uo1.label(r) Log of hourly wage
uo1.label(p)
```

To clear and then set the units field, use:

```
uo1.label(u) Millions of bushels
```

Cross-references

See “Labeling Objects” on page 123 of *User’s Guide I* for a discussion of labels.

members	User Object Views

Displays a list of all members currently stored inside the user object.

Syntax

userobject_name.**members***(options)*

Options

p	Print the spreadsheet view.

Examples

```
myuserobj.members(p)
```

displays and prints the members view of user object MYUSEROBJ.

Cross-references

See Chapter 9. "User Objects," on page 233 of *Command and Programming Reference* for discussion of user objects.

See also `Userobj::add` (p. 1123) and `Userobj::drop` (p. 1127).

olepush	User Object Procs

Push updates to OLE linked objects in open applications.

Syntax

userobj_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

setattr	User Object Procs

Set the object attribute.

Syntax

userobject_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the "revised" attribute in the object A to the string "never", and extracts the attribute into the string object S.

Cross-references

See "Adding Custom Attributes in the Label View" on page 123 and "Adding Your Own Label Attributes" on page 70 of *User's Guide I*.

userobj	User Object Declaration

Declare an unregistered, empty user object.

The `userobj` command declares a new empty user object.

Syntax

userobj *userobject_name*

Examples

```
userobj uo1
```

creates a new empty user object called UO1.

Cross-references

See Chapter 9. “User Objects,” on page 233 of *Command and Programming Reference* for discussion of user objects.

Var

Vector autoregression and error correction object.

Var Declaration

var.....declare var estimation object (p. 1206).

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

btvcvar.....estimate a Bayesian time-varying coefficients VAR specification (p. 1142).
bvar.....estimate a Bayesian VAR specification (p. 1144).
ec.....estimate a vector error correction model (p. 1155).
ls.....estimate an unrestricted VAR (p. 1177).
mfvar.....mixed frequency VAR (p. 1189).
switchvar.....switching VAR (including simple and Markov switching) (p. 1199).

Var Views

arlm.....serial correlation LM test (p. 1140).
arroots.....inverse roots of the AR polynomial (p. 1141).
coint.....Johansen cointegration test (p. 1149).
correl.....residual autocorrelations (p. 1153).
display.....display table, graph, or spool in object window (p. 1154).
endog.....table or graph of endogenous variables (p. 1159).
hdecomp.....perform historical decomposition for a standard VAR (p. 1165).
impulse.....impulse response functions (p. 1167).
jbera.....residual normality test (p. 1173).
label.....label information for the var object (p. 1175).
laglen.....lag order selection criteria (p. 1176).
output.....table of estimation results (p. 1191).
qstats.....residual portmanteau tests (p. 1193).
representations.....text describing var specification (p. 1193).
residcor.....residual correlation matrix (p. 1194).
residcov.....residual covariance matrix (p. 1194).
resids.....residual graphs (p. 1195).
results.....table of estimation results (p. 1196).

Var Procs

setattr .....set the value of an object attribute (p. 1197).

svar .....estimate factorization matrix for structural innovations (p. 1198).

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.

@lagcount .....number of lags included in the VAR.

@lagorder.....highest lag order included in the VAR.

@logl.....log likelihood for system.

@ncoefs.....total number of estimated coefficients in the var.

@neqn.....number of equations.

@nrestrict.....number of coefficient restrictions in the system.

@regobs .....number of observations in the var.

@sc .....Schwarz information criterion for the system.

@svarcvgtype .....integer indicating the convergence type of the structural decomposition estimation (structural VAR only): 0 (convergence achieved), 1 (convergence achieved, but first or second order conditions not met), 2 (failure to improve), 3 (maximum iterations reached), 4 (no convergence—structural decomposition not estimated).

@svaroverid..... over-identification LR statistic from structural factorization (structural VAR only).

@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).

@cointadj..... matrix containing cointegrating variable adjustment coefficients α (where applicable).

@cointadjse ..... matrix containing standard errors of cointegrating variable adjustment coefficients (where applicable).

@cointlr..... matrix containing long-run equilibrium coefficients $\Pi = \alpha\beta'$ (where applicable).

@cointlrse..... matrix containing standard errors of long-run equilibrium coefficients (where applicable)

@cointse ..... standard errors of matrix of cointegrating vectors (where applicable).

@cointsr ..... matrix containing short-run adjustment coefficients Γ' (where applicable).

@cointsrse ..... matrix containing standard errors of short-run adjustment coefficients (where applicable).

@cointvec ..... matrix of cointegrating vectors, β (where applicable).

@companion..... companion matrix for the full set of lag coefficients.

@impfact ..... factorization matrix used in last impulse response view.

@lagcoefs..... coefficient matrix containing the full set of horizontally concatenated lag coefficient matrices.

@lagcoef(k) ..... lag coefficient matrix for lag k.

@lagcoefsum..... sum of the lag coefficient matrices.

@lagids..... vector of integers containing the lags used in estimation.

@lrrsp..... accumulated long-run responses from last impulse response view.

@lrrspse..... standard errors of accumulated long-run responses.

@postresidcov..... estimated posterior error covariance for Bayesian models; ordinary residual covariance, otherwise.

@residcov ..... (sym) covariance matrix of the residuals.

@svaramat..... estimated A matrix for structural factorization (structural VAR only).

@svarbmat..... estimated B matrix for structural factorization (structural VAR only).

@svarcovab .....covariance matrix of stacked A and B matrix for structural factorization (structural VAR only).

@svarfmat .....estimated F matrix for long-run impulse responses (structural VAR only).

@svarrcov .....restricted residual covariance matrix from structural factorization (structural VAR only).

@svarsmat.....estimated S matrix for short-run impulse responses (structural VAR only).

@swcompanion(i). ..... switching companion matrix for the full set of lag coefficients in regime i (switching VAR only).

@swimpfact(i) .....switching factorization matrix for regime i used in last impulse response view (switching VAR only).

@swlagcoefs .....switching coefficient matrix containing the full set of horizontally concatenated lag coefficient matrices for regime i (switching VAR only).

@swlagcoefsum(i) ..... sum of the switching lag coefficient matrices for regime i (switching VAR only).

String values

@attr("*arg*").....string containing the value of the *arg* attribute, where the argument is specified as a quoted string.

@command .....full command line form of the estimation command. Note this is a combination of `@method` and `@options`.

@description.....string containing the VAR object's description (if available).

@detailedtype .....returns a string with the object type: "VAR".

@displayname .....returns the VAR's display name. If the VAR has no display name set, the VAR's name is returned.

@method.....command line form of the estimation method ("LS", "BVAR", *etc.*)

@name.....returns the VAR's name.

@options .....command line form of estimation options.

@remarks.....string containing the var object's remarks (if available).

@smpl.....sample used for estimation.

@type.....returns a string with the object type: "VAR".

@updatetime .....returns a string representation of the time and date at which the VAR was last updated.

Var Examples

To declare a var estimate a VEC specification and make a residual series:

```
var finec.ec(e,2) 1 6 cp div r
finec.makeresids
```

To estimate an ordinary var, to create series containing residuals, and to form a model based upon the estimated var:

```
var empvar.ls 1 4 payroll hhold gdp
empvar.makeresids payres hholdres gdpres
empvar.makemodel(inmdl) cp fcp div fdiv r fr
```

To save coefficients in a scalar:

```
scalar coef1=empvar.b(1,2)
```

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:

svar	Text for identifying restrictions for structural VAR.
coint	Text for restrictions on the cointegration relations and/or adjustment coefficients.

VEC Restrictions

Restrictions may be placed on the coefficients B(r, k) of the r-th cointegrating relation and on the adjustment coefficients, where A(k, r) is the coefficient of the r-th cointegrating relation in the k-th VEC equation. Restrictions are entered in a comma separated list of the form:

B(r, k) = *value*

and

A(k, r) = *value*

SVAR Restrictions

SVAR text restriction expressions involve linear equations or use a function-like syntax to specify restrictions on one or more matrix elements for the A, B, S, and F.

For direct restrictions on elements of the matrices, the EViews text syntax requires that the canonical structural matrix names are preceded by "@", as in "@A", or "@F", to avoid ambiguity with workfile objects, *e.g.,* scalars or matrix object elements.

EViews also offers function-like expressions that concisely specify popular sets of SVAR restrictions. In the following list, the token X can be substituted with any of the canonical matrices A, B, S, and F. The canonical names should not be preceded by "@" in this context since there is no potential workfile object ambiguity in the function argument(s).

@*X* = *mat*	Use *mat* as a pattern matrix for matrix X, *e.g.*, "@a = mat1", "@b = @mat2".
@vec(*X*) = n_1, n_2, n_3, ...	Restricts all elements of matrix *X* similar using the specified pattern matrix (provided in list form). Element ordering matches the vectorization of the matrix, *i.e.*, the elements of the first column, followed by the second column, followed by the third column, etc.
@diag(*X*)	Restricts *X* to be a diagonal matrix, *i.e.*, off-diagonal elements are zero. The diagonal elements are unrestricted.
@diag(*X*) = *n*	Restricts *X* to be a diagonal matrix with elements on the diagonal restricted to be *n*.
@lower(*X*)	Restricts *X* to be a lower triangular matrix, *i.e.*, elements above the diagonal are zero.
@unitlower(*X*)	Restricts *X* to be a unit lower triangular matrix, *i.e.*, elements above the diagonal are zero and elements on the diagonal are one.
@upper(*X*)	Restricts *X* to be an upper triangular matrix, *i.e.*, elements below the diagonal are zero.
@unitupper(*X*)	Restricts *X* to be a unit upper triangular matrix, *i.e.*, elements below the diagonal are zero and elements on the diagonal are one.
@row(*X*, *r*) = *n*	Restricts the elements in row *r* of *X* to equal *n*.
@col(*X*, *c*) = *n*	Restricts the elements in column *c* of *X* equal *n*.

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.

```
var v
v.ls 1 3 y1 y2 y3
v.append(svar) @a(1,1) = 2.5
v.append(svar) @b(2,2) = @b(3,3) / 2
v.append(svar) @a(1,1) + @a(2,1) = 1
v.append(svar) @a(1,2) = 3 * @b(3,3)
v.append(svar) @s(1,1) + @s(2,2) - @f(3,3) = 1.5
v.svar
```

For an SVAR, we first estimate the VAR, then append restrictions then perform SVAR estimation.

Using a text expression equivalent to a pattern matrix:

```
v1.append(svar) @vec(s) = na, na, na, 0, na, na, 0, 0, na
v1.svar
```

Using a text expression with specialized function:

```
v1.append(svar) @lower(s)
v1.svar
```

Cross-references

See also `Var::cleartext` (p. 1148).

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=*arg*	Save LM statistics in named matrix object. The matrix has *h* rows and one column.
prompt	Force the dialog to appear from within a program.
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 954 of *User's Guide II* for other VAR diagnostics. See also `Var::qstats` (p. 1193) for related multivariate residual autocorrelation Portmanteau tests.

arroots	Var Views

Inverse roots of the characteristic AR polynomial.

Syntax

var_name.**arroots***(options)*

Options

name = *arg*	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 954 of *User's Guide II* for other VAR diagnostics.

btvcvar	Var Methods

Estimate a Bayesian time-varying coefficients VAR, or BTVCVAR, model.

Syntax

var_name.**btvcvar***(options) lag_pairs endog_list [@ exog_list]*

`btvcvar` estimates a Bayesian time-varying coefficients VAR. The order of the VAR is specified using a lag pair, followed by a list of series or groups for endogenous variables. Exogenous variables can be included using the @-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, use the noconst option.

Options

Prior hyper-parameters

T0 = *int* (*default* = 0)	Set prior sample size T_0. A prior sample is not used if T0 is set to 0. To use a prior sample, T0 must be set to an integer larger than the number of coefficients per equation in a standard VAR version of the model.
tau0 = *num* *(default = 5.0)*	Set prior scaling parameter for the initial state b_0.
tau1 = *num* (*default* = 1.0)	Set prior scaling parameter for the observation covariance S.
nu1 = *num* (*default* = 5.0)	Set prior dof parameter for the observation covariance S.
tau2 = *num* (*default* = 0.01)	Set prior scaling parameter for the process covariance Q.
nu2 = *num* (*default* = 5.0)	Set prior dof parameter for the process covariance Q.

Display options

usemean	Use posterior mean as the point estimate. The posterior median is used if usemean is not included in the options list.
showci	Show credibility intervals (bands).
cilevels = *arg* (*default* = "0.95")	Set credibility levels. For multiple levels, enter a space-delimited list of values surrounded by quotation marks, e.g., "0.3 0.5 0.8".
uselines	Use lines instead of shading for credibility intervals.

MCMC options

burn = *int* (*default* = 5000)	Set burn-in size.
size = *int* (*default* = 5000)	Set posterior sample size.
thin = *int* (*default* = 1)	Set thinning size. A thinning size of r indicates that every r-th draw after the burn-in period is stored.
nsub = *int*	Set the number of subchains.
seed = *int*	Set the random seed. EViews will generate a seed if one is not specified.
rng = *arg* (*default* = "kn" or method set via rndseed)	Set random number generator type. Available types are: 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").

Other options

smoother = *arg* (*default* = "CFA")	Set simulation smoothing method. Available methods are "CFA" (Cholesky factor algorithm), "KFS" (Kalman filter and smoother), and "MMP" (McCausland, Miller, & Pelletier, 2011).
stable	Use the method of Cogley & Sargent for obtaining stable draws.
maxattempts = *int* (*default* = 100)	Set the maximum number of attempts for the sampler to draw stable VAR coefficients for all dates in the data sample.
noconst	Do not include a constant in the exogenous regressors list.

Examples

To declare and estimate a BTVCVAR named MYVAR with endogenous variables DGP and UNEMP, a constant, the first lag, and a prior sample of size 40, run

```
var myvar.btvcvar(t0=40) 1 1 gdp unemp
```

in the command window. Running the command

```
var myvar.btvcvar(t0=40, showci, cilevels="0.3 0.5") 1 1 gdp unemp
```

will also display shaded 30% and 50% credibility bands. For reproducible results, also set the number of subchains (*nsub*), the random seed (*seed*), and the random number generator type (*rng*):

```
var myvar.btvcvar(t0=40, nsub=12, seed=342458900, rng=kn, showci,
   cilevels="0.3 0.5") 1 1 gdp unemp
```

Command capture will always show the *nsub*, *seed*, and *rng* options.

Cross-references

See Chapter 48. “Bayesian Time-varying Coefficients VAR Models,” on page 1083 of *User’s Guide II* for details.

bvar	Var Methods

Estimate a Bayesian VAR specification.

Syntax:

var_name.**bvar***(options) lag_pairs endog_list [@ exog_list]*

`bvar` estimates an Bayesian VAR. 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.
prior = *keyword* (*default* = "lit")	Set the prior type: "lit" (Litterman/Minnesota), "nw" (Normal-Wishart), "sznw" (Sims-Zha Normal-Wishart prior), "sznf" (Sims-Zha Normal-flat prior), "glp" (Giannone, Lenze, and Primiceri).
initcov = *keyword* (*default* = "uni")	Set the (initial) residual variance-covariance: "full" (full Classical VAR), "uni" (univariate AR), "diag" (diagonal of full classical VAR), "arconst" (univariate AR and a constant) By default, EViews uses the "initcov = uni" option so that diagonal elements of the prior residual variance-covariance can be obtained from the estimation of a set of univariate AR models.
nodf	Do not degree-of-freedom correct the initial residual covariance.
initexog	Use exogenous variables in initial covariance estimate.
initdummied	Use dummy variables in initial covariance estimate. (Only applicable if dummy variable options are below).
icsmpl = *arg*	Set the sample used for initial covariance estimate (estimation sample used if omitted).
sumcoef	Use the sum-of-coefficients dummy variable (only applicable if *not* using Sims-Zha or GLP prior).
initobs	Use the initial-observations dummy variable (only applicable if *not* using Sims-Zha or GLP prior).
nsumcoef	Do not use sum-of-coefficients dummy variable (only applicable if using Sims-Zha or GLP prior).
ninitobs	Do not use initial-observations dummy variable (only applicable if using Sims-Zha or GLP prior).
l0 = *num*	Set the residual covariance tightness hyper-parameter.
l1 = *num*	Set the overall tightness hyper-parameter.
l2 = *num*	Set the relative cross-variable weight hyper-parameter.
l3 = *num*	Set the lag decay hyper-parameter.
l4 = *num*	Set the exogenous variables hyper-parameter.
l5 = *num*	Set the other exogenous variables hyper-parameter.
mu1 = *num*	Set the AR(1) coefficient dummies hyper-parameter.

mu5 = *num*	Set the sum of coefficient dummies hyper-parameter.
mu6 = *num*	Set the initial observation dummies hyper-parameter.
c1 = *num*	Set the S scale hyper-parameter.
c2 = *num*	Set the V scale hyper-parameter.
c3 = *num*	Set the degrees-of-freedom hyper-parameter.
optl1	Optimize the L1 hyper-parameter.
optl3	Optimize the L3 hyper-parameter.
optmu5	Optimize the MU5 hyper-parameter.
optmu6	Optimize the MU6 hyper-parameter.
optpsi	Optimize the initial covariance estimates.
draws = *num*	Set the number of MCMC draws (only applicable with "prior = inw").
seed = *num*	Set the seed for the MCMC generator (only applicable with "prior = inw").
burn = *num*	Set the percentage of MCMC draws to use as a burn-in (only applicable with "prior = inw").
m = *integer*	Set maximum number of iterations (only applicable with "prior = glp").
c = *scalar*	Set convergence criterion (only applicable with "prior = glp").
prompt	Force the dialog to appear from within a program.
p	Print basic estimation results.

Examples

```
var mvar.bvar 1 3 m1 gdp
```

declares and estimates an Bayesian VAR with a Litterman prior named MVAR with two endogenous variables (M1 and GDP), a constant and 3 lags (1 through 3).

```
mvar.bvar(noconst) 1 3 ml gdp
```

estimates the same VAR, but with no constant.

```
var mvar.bvar(prior=nw, mu1=0.2, C1=0.2) 1 3 m1 gdp
```

specifies a normal-Wishart with hyper-prior values specified as mu1 = 0.2, C1 = 0.2.

```
var mvar.bvar(prior=inw, c1=0.15,c2=0.15, draws=50000) 1 3 m1 gdp
```

specifies an independent normal-Wishart prior with both C1 and C2 hyper-parameters set equal to 0.15, and the number of Gibb's sampler draws set to 50,000.

```
Var mbvar.bvar(prior=glp, initcov=diag, nsumcoef, ninitobs, optl3,
    m=5000) 1 3 m1 gdp
```

Specifies a Giannone-Lenza-Primeceri prior, where only Lambda 3 will be optimized, the initial covariance matrix is estimated as a diagonal VAR, neither initial dummy observation priors, and the number of optimization iterations set to 5,000.

Cross-references

See Chapter 47. "Bayesian VAR Models," on page 1053 of *User's Guide II* for details.

See also `Var::ls` (p. 1177) and `Var::ec` (p. 1155) for estimation of ordinary VARs and error correction models.

clearhist | Var Procs

Clear the contents of the history attribute for VAR objects.

Removes the VAR's history attribute, as shown in the label view of the VAR.

Syntax

var_name.**clearhist**

Examples

```
v1.clearhist
v1.label
```

The first line removes the history from the VAR V1, and the second line displays the label view of V1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Var::label` (p. 1175).

clearremarks | Var Procs

Clear the contents of the remarks attribute.

Removes the var's remarks attribute, as shown in the label view of the var.

Syntax

var_name.**clearremarks**

Examples

```
v1.clearremarks
v1.label
```

The first line removes the remarks from the var V1, and the second line displays the label view of V1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of the *User's Guide I* for a discussion of labels and display names.

See also `Var::label` (p. 1175).

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 44. "Vector Autoregression (VAR) Models," on page 939 of *User's Guide II* for a discussion of VARs.

See also `Var::append` (p. 1138).

coint	Var Views

Johansen's cointegration test for the series in the var object.

Syntax

var_name.**coint***(options) [lag_spec] [@ x1 x2 x3 ...] [@***exogsr** *sx1 sx2 sx3 ...] [@***exoglr** *lx1 lx2 lx3 ...] [@***exogboth** *bx1 bx2 bx3 ...]*

uses the `coint` keyword followed by *options,* and optionally,

- a *lag_spec* consisting of one or more pairs of lag intervals, where the lag orders are for the differences in the error correction representation of the VEC, not the levels representation of the VAR.
- an "@"-sign or "@exogsr" followed by a list of exogenous variables in the short-run equation only
- "@exoglr" followed by a list of exogenous variables in the long-run relation only
- "@exogboth" followed by a list of exogenous variables in both the long-run relation and the short-run equations

The `coint` command tests for cointegration among the series in the var object using the lag spec, exogenous variables, and if relevant, deterministic spec and VEC restrictions specified in estimation.

- You may provide explicit *lag_spec* to override the one used in estimation.

  Note that if the estimation lags were for a VAR specification in levels, the default *lag_spec* will be the original spec adjusted for the error correction differences. Thus, if the original estimation was for a "1 4" VAR, the default *lag_spec* will be "1 3".
- You may provide a "determ = " option to override an existing VEC deterministic trend specification.
- You may explicitly list any type of exogenous variable to override the entire existing specification for the exogenous variables.

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, Haug, and Michelis (1999). The 0.05 critical values are also 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

Deterministic Trend Option

There are 8 different deterministic trend assumptions that you may specify using the "determ = *arg*" option.

These cases correspond to whether the intercept ("c") and the trend ("t") are either

- not included ("n")
- in the long-run cointegrating relation only ("l")
- in the short-run equation only ("s")
- in both the long and short-run equations ("b")

The values of *arg* are text shortcuts formed by joining a text shortcut for the intercept specification with a text shortcut for the trend specification.

The individual intercept and trend specifications are formed by joining the "c" and the "t" with the appropriate letter describing inclusion in the long and short-run equations.

For example,

- "cb" indicates that the constant is in both the long and short-run equation
- "tl" indicates that the trend is in the long-run cointegrating equation only

so that

- "cbtl" indicates that the constant is in both the long and short-run and the trend is in the long-run only

Using this convention (along with a special "none" option), we may easily describe options arguments for all 8 deterministic cases:

cntn, none	Case 1: No deterministic terms. Corresponding VAR model has no deterministic terms.
cltn	Case 2: Restricted constant. Constant only in the cointegrating relations. Corresponding VAR has a constant.
cbtn *(default)*	Case 3 (JHJ): Unrestricted constant Constant included both in the short-run equation and (artificially) in the cointegrating relations via orthogonalization. Corresponding VAR has a constant and trend.
cstn	Case 3: Unrestricted constant Constant only in the short-run equation. Corresponding VAR has a trend.

cbtl	Case 4 (JHJ): Unrestricted constant and restricted trend Constant included both in the short-run equation and (artificially) in the cointegrating relations via orthogonalization, and trend included only in the cointegrating relations. Corresponding VAR has a constant and trend.
cstl	Case 4: Unrestricted constant and restricted trend Constant only in the short-run equation, and trend only in the cointegrating relation. Corresponding VAR has a trend.
cbtb	Case 5 (JHJ): Unrestricted constant and trend Constant and trend both included in the short-run equation and (artificially) in the cointegrating relations via orthogonalization. Corresponding VAR has a constant, linear, and quadratic trend.
csts	Case 5: Unrestricted constant and trend Constant and trend both included in the short-run equation. Corresponding VAR has a linear and quadratic trend.

or you may use the "determsummary" option to compute tests under all deterministic assumptions.

Other Options

determsummary	Summarize all deterministic trend cases.
restrict	Impose restrictions as specified by the `Var::append` (p. 1138) proc, or the "restspec = " option.
restspec = "*spec*"	Define the restricted VEC specification where *spec* is a space a space delimited list of VEC coefficient restrictions.
m = *integer*, maxit = *integer*	Maximum number of iterations for restricted estimation (only valid if you choose the restrict option).
c = *scalar*, cvg = *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".
prompt	Force the dialog to appear from within a program.
p	Print results.

Examples

```
var1.coint(determ=cbtl) 1 12 @
```

carries out the Johansen test for the series in the var object named VAR1 using lags 1 to 12. The "@"-sign without a list of exogenous variables ensures that the test does not include any exogenous variables in VAR1.

Cross-references

See "Johansen Cointegration Test" on page 1461 of *User's Guide II* for details on the Johansen test.

See also `Var::ec` (p. 1155).

copy	Var Procs

Creates a copy of the var.

Creates either a named or unnamed copy of the var.

Syntax

var_name.**copy**

var_name.**copy** *dest_name*

Examples

```
v1.copy
```

creates an unnamed copy of the var V1.

```
v1.copy v2
```

creates V2, a copy of the var V1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

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 *(default)*	Display correlograms (graphs).
byser	Display autocorrelations in tabular form, by series.
bylag	Display autocorrelations in tabular form, by lag.
name = *arg*	Saves matrix of results containing the data corresponding to the specified view.
prompt	Force the dialog to appear from within a program.
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 489 and "Partial Autocorrelations (PAC)" on page 490 of *User's Guide I* for a discussion of autocorrelation and partial correlation functions, respectively.

display	Var Views

Display table, graph, or spool output in the VAR object window.

Display the contents of a table, graph, or spool in the window of the VAR object.

Syntax

var_name.**display** *object_name*

Examples

```
var1.display tab1
```

Display the contents of the table TAB1 in the window of the object VAR1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

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 123 of *User's Guide I* for a discussion of labels and display names.

See also Var::label (p. 1175).

ec	Var Methods

Estimate a vector error correction model (VEC).

Syntax

var_name.**ec***(options) lag_pairs endog_list [@ x1 x2 x3 ...] [***@exogsr** *sx1 sx2 sx3 ...] [***@exoglr** *lx1 lx2 lx3 ...] [***@exogboth** *bx1 bx2 bx3 ...]*

Specify the order of the VEC by entering *lag_pairs* consisting of one or more pairs of lag intervals, and then list the series or groups to be used as endogenous variables.

Note that the lag orders are for the differences in the error correction representation of the VEC, *not the levels* representation of the VAR. If you are comparing results obtained elsewhere, you should be certain that the specifications for the lag orders are comparable.

In addition, you may optionally provide:

- an "@"-sign or "@exogsr" followed by a list of exogenous variables in the short-run equation only
- "@exoglr" followed by a list of exogenous variables in the long-run relation only
- "@exogboth" followed by a list of exogenous variables in both the long-run relation and the short-run equations

Do not include an intercept or trend in the VEC specification, these deterministic trend terms should be specified using the "determ = " option, as described below.

Options

Deterministic Trend Option

There are 8 different deterministic trend assumptions that you may specify using the "determ = *arg*" option.

These cases correspond to whether the intercept ("c") and the trend ("t") are either

- not included ("n")
- in the long-run cointegrating relation only ("l")
- in the short-run equation only ("s")

- in both the long and short-run equations ("b")

The values of *arg* are text shortcuts formed by joining a text shortcut for the intercept specification with a text shortcut for the trend specification.

The individual intercept and trend specifications are formed by joining the "c" and the "t" with the appropriate letter describing inclusion in the long and short-run equations.

For example,

- "cb" indicates that the constant is in both the long and short-run equation
- "tl" indicates that the trend is in the long-run cointegrating equation only

so that

- "cbtl" indicates that the constant is in both the long and short-run and the trend is in the long-run only

Using this convention (along with a special "none" option), we may easily describe options arguments for all 8 deterministic cases:

cntn, none	Case 1: No deterministic terms. Corresponding VAR model has no deterministic terms.
cltn	Case 2: Restricted constant. Constant only in the cointegrating relations. Corresponding VAR has a constant.
cbtn *(default)*	Case 3 (JHJ): Unrestricted constant Constant included both in the short-run equation and (artificially) in the cointegrating relations via orthogonalization. Corresponding VAR has a constant and trend.
cstn	Case 3: Unrestricted constant Constant only in the short-run equation. Corresponding VAR has a trend.
cbtl	Case 4 (JHJ): Unrestricted constant and restricted trend Constant included both in the short-run equation and (artificially) in the cointegrating relations via orthogonalization, and trend included only in the cointegrating relations. Corresponding VAR has a constant and trend.
cstl	Case 4: Unrestricted constant and restricted trend Constant only in the short-run equation, and trend only in the cointegrating relation. Corresponding VAR has a trend.

cbtb	Case 5 (JHJ): Unrestricted constant and trend Constant and trend both included in the short-run equation and (artificially) in the cointegrating relations via orthogonalization. Corresponding VAR has a constant, linear, and quadratic trend.
csts	Case 5: Unrestricted constant and trend Constant and trend both included in the short-run equation. Corresponding VAR has a linear and quadratic trend.

Other Options

rank = *integer* *(default = 1)*	Number of cointegrating relationships.
restrict	Impose restrictions as specified by the `Var::append` (p. 1138) proc, or the "restspec = " option.
restspec = "*spec*"	Define the restricted VEC specification where *spec* is a space a space delimited list of VEC coefficient restrictions.
m = *integer*, maxit = *integer*	Maximum number of iterations for restricted estimation (only valid if you choose the restrict option).
c = *scalar*, cvg = *scalar*	Convergence criterion for restricted estimation. (only valid if you choose the restrict option).
prompt	Force the dialog to appear from within a program.
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(determ=cstl, rank=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 deterministic trend option "determ = cstl" for a model with a constant in the short-run equation, and a trend in the long-run cointegrating relation.

```
var macro1.ec(determ=cstl, rank=2) 1 2 4 4 tb1 tb3 tb6 @exogsr
    exog1 @exoglr exog2 @exogdual exog3
```

The line above declares a VAR object MACRO01 with the same basic specification as TERM, but with an additional short-run exogenous variable EXOG1, a long-run exogenous variable EXOG2, and a dual exogenous variable EXOG3.

Cross-references

See Chapter 45. "Vector Error Correction Models (VECMs)" of *User's Guide II* for a discussion of VECs.

See `Var::ls` (p. 1177) and `Var::bvar` (p. 1144) for estimation of ordinary VARs and Bayesian VAR models. See also `Var::coint` (p. 1149) and `Var::append` (p. 1138).

drawcoefs

Var Procs

Draw from the posterior coefficient distribution.

Draws samples from the posterior coefficient distribution of a Bayesian VAR and saves the individual draws into a matrix object. Each row of the matrix will contain the vec'd coefficients from a single draw.

Syntax

var_name.**drawcoefs***(options) matrix_name*

Options

draws = *integer* *(default* = 1000*)*	Number of draws.
burn = *arg* *(default* = 0.1*)*	Proportion of initial draws to discard (only applicable if Bayesian VAR was estimated with "prior = inw")
seed = *integer*	Random number seed.

Examples

```
var1.drawcoefs(draws=10000) vardraws
```

will create a matrix object, VARDRAWS, with 10,000 rows, containing draws from the posterior coefficient distribution.

Cross-references

See Chapter 47. "Bayesian VAR Models," on page 1053 of *User's Guide II* for details on drawing from the posterior distributions.

See also `Var::drawrescov` (p. 1159).

drawrescov	Var Procs

Draw from the posterior coefficient distribution.

Draws samples from the posterior residual covariance distribution of a Bayesian VAR and saves the individual draws into a matrix object. Each row of the matrix will contain the vec'd form of the covariance matrix from a single draw.

Syntax

var_name.**drawrescov***(options) matrix_name*

Options

draws = *integer* (*default* = 1000)	Number of draws.
burn = *arg* (*default* = 0.1)	Proportion of initial draws to discard (only applicable if Bayesian VAR was estimated with "prior = inw")
seed = *integer*	Random number seed.

Examples

```
var1.drawrescov(draws=10000) vardraws
```

will create a matrix object, VARDRAWS, with 10,000 rows, containing draws from the posterior residual covariance distribution.

Cross-references

See Chapter 47. "Bayesian VAR Models," on page 1053 of *User's Guide II* for details on drawing from the posterior distributions.

See also `Var::drawcoefs` (p. 1158).

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. 1181) and `Var::var` (p. 1206).

fit — Var Procs

Computes static forecasts of the VAR or VEC equation.

`fit` computes the static forecast of variables and 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

var_name.**fit***(options) f_pattern [se_pattern]*

You should enter a naming suffix for the forecast series and, optionally, a naming suffix for the series containing the standard errors. Standard errors are currently only available for non-Bayesian VARs, and are computed via simulation.

Not currently available for switching VARs

Options

General Options

g	Graph the forecasts in individual graphs - one per dependent variable.
m	Graph the forecasts in a combined graph.
e	Produce the forecast evaluation table.

f = *arg* *(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).
prompt	Force the dialog to appear from within a program.
p	Print view.

Non-Bayesian Options

streps = *integer*	Number of simulation repetitions. Only applicable if a se_pattern is provided.
f = *number*	Fraction of failed repetitions before stopping. Only applicable if a se_pattern is provided.

BVAR Options

classical	Perform classical forecasting – forecast based upon the posterior means of the coefficients as if they were calculated from a classical VAR. If omitted Bayesian sampling is used.

If "classical" is not specified, the following Bayesian forecasting options are available:

mean	Store the mean of the draws from the sampler. If omitted the median is stored.
draws = *integer* *(default* = 100000*)*	Number of draws.
burn = *arg* *(default* = 0.1*)*	Proportion of initial draws to discard.
seed = *integer*	Random number seed.
dropunstable	Drop any draws that produce unstable coefficients.
dgraph	Produce distribution graphs.
fangraph	Produce fan graphs.
page = *arg*	Store the individual draws in a new page.

BTVCVAR Options

usemean	Use posterior mean as the point estimate. The posterior median is used if usemean is not included in the options list.
showci	Show credibility intervals (bands).
cilevels = *arg* (*default* = "0.95")	Set credibility levels. For multiple levels, enter a space-delimited list of values surrounded by quotation marks, e.g., "0.3 0.5 0.8".
uselines	Use lines instead of shading for credibility intervals.
seed = *int*	Set the random seed. EViews will generate a seed if one is not specified.
rng = *arg* (*default* = "kn" or method set via rndseed)	Set random number generator type. Available types are: 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").

Examples

The following lines:

```
smpl 1970q1 1990q4
var var1.ls 1 3 con inc
smpl 1991q1 1995q4
var1.fit(m) _f _se
```

estimate a VAR over the period 1970Q1–1990Q4, and then computes static forecasts for the period 1991Q1–1995Q4, and plots the forecasts as line graphs.

Cross-references

See "Forecasting" on page 964 of *User's Guide II* for a discussion of forecasting from VARs.

See also `Var::forecast` (p. 1162).

forecast	Var Procs

Computes (*n*-period ahead) dynamic forecasts of the VAR or VEC equation.

`forecast` computes the forecast for all variables and 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

var_name.**forecast***(options) f_pattern [se_pattern]*

You should enter a naming suffix for the forecast series and, optionally, a naming suffix for the series containing the standard errors. Forecast standard errors are currently only available for non-Bayesian VARs, and are computed via simulation.

Not currently available for switching VARs

Options

General Options

g	Graph the forecasts in individual graphs - one per dependent variable.
m	Graph the forecasts in a combined graph.
e	Produce the forecast evaluation table.
f = *arg (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).
prompt	Force the dialog to appear from within a program.
p	Print view.

Non-Bayesian Options

streps = *integer*	Number of simulation repetitions. Only applicable if a *se_pattern* is provided.
f = *number*	Fraction of failed repetitions before stopping. Only applicable if a *se_pattern* is provided.

BVAR Options

classical	Perform classical forecasting – forecast based upon the posterior means of the coefficients as if they were calculated from a classical VAR. If omitted Bayesian sampling is used.

If "classical" is not specified, the following Bayesian forecasting options are available:

mean	Store the mean of the draws from the sampler. If omitted the median is stored.
draws = *integer* *(default* = 100000*)*	Number of draws.
burn = *arg* *(default* = 0.1*)*	Proportion of initial draws to discard.
seed = *integer*	Random number seed.
dropunstable	Drop any draws that produce unstable coefficients.
dgraph	Produce distribution graphs.
fangraph	Produce fan graphs.
page = *arg*	Store the individual draws in a new page.

BTVCVAR Options

usemean	Use posterior mean as the point estimate. The posterior median is used if usemean is not included in the options list.
showci	Show credibility intervals (bands).
cilevels = *arg* (*default* = "0.95")	Set credibility levels. For multiple levels, enter a space-delimited list of values surrounded by quotation marks, e.g., "0.3 0.5 0.8".
uselines	Use lines instead of shading for credibility intervals.
seed = *int*	Set the random seed. EViews will generate a seed if one is not specified.
rng = *arg* (*default* = "kn" or method set via rndseed)	Set random number generator type. Available types are: 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").

Examples

The following lines:

```
smpl 1970q1 1990q4
var var1.ls 1 3 con inc
smpl 1991q1 1995q4
var1.forecast(m) _f _se
```

estimate a VAR over the period 1970Q1–1990Q4, and then computes dynamic forecasts for the period 1991Q1–1995Q4, and plots the forecasts as line graphs.

```
smpl 1970q1 1990q4
var var2.bvar(prior=inw) 1 3 con inc
smpl 1991q1 1995q4
var1.forecast(m, draws=50000, burn=.05, dgraph, page=draws) _f
```

estimates a Bayesian VAR with an independent normal-Wishart prior over the same period, and then forecasts that VAR taking 50,000 draws of a Gibbs sampler, discarding the first 2,500 draws, producing a distribution graph of the forecasts and storing the draws into a new panel page called DRAWS.

Cross-references

See "Forecasting" on page 964 of *User's Guide II* for a discussion of forecasting from VARs and VECS.

See also `Var::fit` (p. 1160).

hdecomp	Var Views

Performs graph of historical decomposition for a standard VAR.

Syntax

var_name.**hdecomp**(*n, options*) *[ser1 ser2 ser3 ...] [@ component_series [@ ordering_series]]*

List the series names in the Var you would like to decompose in the order you wish to display the graphs. If you do not specify series, all of the series in the Var will be employed. You may optionally specify the component series by listing the series after an "@" and, if you are using Cholesky weighting for the decomposition, you may change the ordering by listing the order of the series after a second "@".

By default, EViews computes the decomposition of only the stochastic component into all components using the ordering in the Var. You may instead elect to include the baseline in the decomposition.

Options

m *(default)*	Display multiple graphs, with the impact of each component on a dependent variable shown in a separate graphs.
g	Display combined graphs, with decomposition of each variable with respect to all included components shown in one graph.
imp = *arg* *(default* = "chol"*)*	Type of factorization for the decomposition: unit impulses, ignoring correlations among the residuals ("imp = unit"), non-orthogonal, ignoring correlations among the residuals ("imp = nonort"), Cholesky with d.f. correction ("imp = chol"), Cholesky without d.f. correction ("imp = mlechol"), Generalized ("imp = gen"), structural ("imp = struct"), or user specified ("imp = user"). The structural factorization is based on the estimated structural VAR. To use this option, you must first estimate the structural decomposition; see `Var::svar` (p. 1198). For user-specified weights, you must specify the name of the vector/matrix containing the impulses using the "fname = " option.
baseline	Include the baseline in the decomposition.
start = *date*	Start date (within estimation sample) for the decomposition. By default, EViews uses the first date of the estimation sample.
end = *date*	End date (within estimation sample) for the decomposition. By default, EViews uses the last date of the estimation sample.
basename = *arg*	Optional name fragment used in constructing the names for the series in which to save the decomposition results. If the basename is "base," the total stochastic or stochastic plus baseline will be saved in the series BASE1, BASE2, etc... (where the numbers in the name correspond to the ordering of the variables in the VAR specification). The components will be saved in the series BASE1_1, BASE1_2, …, BASE2_1, BASE2_2, …., where the numbers after the "_" refer to the ordering of the series in the VAR. Series will be saved only for those variables and components requested above.
n = *arg*	Optional name of group object to contain the series created using the "basename = " option.
prompt	Force the dialog to appear from within a program.
p	Print the results.

Examples

```
var var1.ls 1 4 m1 gdp cpi
var1.hdecomp(10, baseline, m) gdp @ m1 gdp cpi
```

The first line declares and estimates a VAR with three variables. The second line displays multiple graphs of the historical decomposition of GDP to shocks to the three series in the VAR using the ordering as specified in VAR1. The baseline will be included in the graph.

Cross-references

See Chapter 44. "Vector Autoregression (VAR) Models," on page 939 of *User's Guide II* for a discussion of historical decomposition in VARs.

impulse	Var Views

Display impulse response functions of var object with an estimated VAR or VEC.

Syntax

var_name.**impulse**(*n*, *options*) *[response_series] [@imp impulse_series] [@order ordering_series]*

In the syntax above, *n* is the number of periods over which to compute impulse response functions. When a value for *n* is not provided, it defaults to 10.

You may optionally filter and sort impulse-response results (for displaying results and outputting results into the workfile) using *response_series* and *impulse_series*.

- *response_series* is a space-delimited list of series names used to filter and sort results based on the response series. It is set to the VAR object's *endog_list* by default.
- *impulse_series* is a space-delimited list of series names used to filter and sort results based on the impulse (shock) series, and is preceded by the keyword "*@imp*". It is set to the VAR object's *endog_list* by default.

If you are using impulses based on the Cholesky factor, you may specify the Cholesky ordering by listing the order of the series after "*@order*". The default behavior is equivalent to setting *ordering_series* to the VAR object's *endog_list*. Note that every variable that appears in the VAR object's *endog_list* must be included in *ordering_series*.

Legacy Syntax

The following syntax has been deprecated:

var_name.**impulse**(*n, options*) *[response_series] [@ impulse_series [@ ordering_series]]*

Mixing the current and legacy syntax is not allowed.

Options

General Options

g	Display combined graphs, with impulse responses of one variable to all shocks shown in one graph.
m *(default)*	Display multiple graphs, with impulse response to each shock shown in separate graphs.
t	Tabulate the impulse responses.
a	Accumulate the impulse responses.
imp = *arg* *(default* = "chol"*)*	Type of factorization for the decomposition: unit impulses, ignoring correlations among the residuals ("imp = unit"), non-orthogonal, ignoring correlations among the residuals ("imp = nonort"), Cholesky with d.f. correction ("imp = chol"), Cholesky without d.f. correction ("imp = mlechol"), Generalized ("imp = gen"), structural ("imp = struct"), or user specified ("imp = user"). The structural factorization is based on the estimated structural VAR. To use this option, you must first estimate the structural decomposition; see `Var::svar` (p. 1198). For user-specified impulses, you must specify the name of the vector/matrix containing the impulses using the "*fname* = " option. The option "*imp* = *mlechol*" is provided for backward compatibility with EViews 3.x and earlier.
fname = *name*	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.
irtype = *arg* *(default* = "ordinary"*)*	Type of impulse response calculation engine: "ordinary" (classical VAR), "lp" (local projection: sequential), and "lpjoint" (local projection: joint).

se = *arg*	Standard error calculations: "se = a" (analytic), "se = mc" (Monte Carlo), "se = boot" (bootstrap). If selecting Monte Carlo or bootstrap, you must specify the number of replications with the "rep = " option. Bootstrap type is set with the "bs = " option. Note the following: (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. (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.
rep = *integer*	Number of Monte Carlo and bootstrap replications to be used in computing standard errors. Must be used with the "se = mc" and "se = boot" options.
bs = *arg* *(default = "hp")*	Bootstrap method: "sp" (standard percentile), "hp" (Hall's percentile), "hs" (Hall's studentized), "ku" (Killian's unbiased). Note the following: Hall's studentized and Kilian's unbiased bootstraps require a double bootstrap. They can be computed quickly using a fast double bootstrap approximation by supplying an additional "fdb" option.
dbsrep = *integer* (*default = 499*)	Number of double bootstrap replications. Must be used with the "bs = hs" and "bs = ku" options. Note the following: This option is not necessary if supplying the "fdb" option.
fdb	Approximate double bootstrap routines with fast double bootstrap routines.
cilevels = *arg* *(default = "0.95")*	Confidence interval coverage as a number between 0 and 1. Note the following: (1) Option is only available when "se = boot". (2) Multiple confidence levels can be supplied as a space delimited list.
uselines	Use lines instead of shading for confidence intervals.

save = *name*	Save impulse-response estimates in a named matrix. Grouping: Results are grouped by impulse (shock), unless the byrsp keyword is used in which case results are grouped by response. Filtering and sorting: Results are filtered and sorted according to *response_series* and *impulse_series*.
savese = *name*	Save impulse-response standard errors (or standard deviations) in a named matrix. No output will be generated if the SE/CI method is set to 'None'. Grouping: Results are grouped by impulse (shock), unless the byrsp keyword is used in which case results are grouped by response. Filtering and sorting: Results are filtered and sorted according to *response_series* and *impulse_series*.
saveci = *name*	Save impulse-response confidence intervals (or credibility intervals) in a named matrix. No output will be generated if the SE/CI method is set to 'None'. Grouping: Results are grouped by impulse (shock), unless the byrsp keyword is used in which case results are grouped by response. Filtering and sorting: Results are filtered and sorted according to *response_series* and *impulse_series*.
saverci = *name*	Save impulse-response estimates and confidence intervals (or credibility intervals) in a named matrix. Estimates are interleaved. No output will be generated if the SE/CI method is set to 'None'. Grouping: Results are grouped by impulse (shock), unless the byrsp keyword is used in which case results are grouped by response. Filtering and sorting: Results are filtered and sorted according to *response_series* and *impulse_series*.
byrsp	Group results by response instead of by impulse (shock).
prompt	Force the dialog to appear from within a program.
p	Print the results.

BVAR Options

bvartype = *arg* *(default* = "bayes"*)*	Impulse method: Bayesian sampling ("bayes"), classical impulse response analysis using the posterior residual covariance matrix ("classpost"), classical impulse response analysis using the empirical residual covariance matrix ("classemp").

If you are using Bayesian sampling, the following Bayesian options are available:

draws = *integer* *(default* = 100000*)*	Number of draws.
burn = *arg* *(default* = 0.1*)*	Proportion of initial draws to discard.
seed = *integer*	Random number seed.
dropunstable	Drop any draws that produce unstable coefficients.
dgraph	Produce distribution graphs.
page = *arg*	Store the individual draws in a new page.

BTVCVAR Options

dates = *arg*	Set the impulse date(s). For multiple dates, enter a space-delimited list of values surrounded by quotation marks, e.g., "1980q1 2000q1 2020q1".
horizons = *arg*	Set the elapsed time horizon(s). For multiple horizons, enter a space-delimited list of positive integers surrounded by quotation marks, e.g., "5 9 13".
usemean	Use posterior mean as the point estimate. The posterior median is used if usemean is not included in the options list.
showci	Show credibility intervals (bands).

Local Projection Options

lpband = *arg* *(default* = "marginal"*)*	Standard error band type: "marginal", "scheffe", and "conditional". Note that marginal error bands do not account for cross-impulse effects, whereas Scheffé and conditional do.
asym = *arg*	Asymmetric/nonlinear effects specified as a categorical variable series name or expression.

Legacy Save (Output to Workfile) Options

The following save options have been deprecated and are only supported for impulse response methods introduced before EViews 13.

Mixing current and legacy save options is not allowed.

matbys = *name*	(1) Save impulse-response estimates in a named matrix. (2) Save impulse-response standard errors in a named matrix, unless the SE/CI method is set to 'None'. Grouping: Results are grouped by impulse (shock). Filtering and sorting: Results are neither filtered nor sorted. User's specifications for *response_series* and *impulse_series* are ignored and replaced with the VAR object's *endog_list*.
matbyr = *name*	(1) Save impulse-response estimates in a named matrix. (2) Save impulse-response standard errors in a named matrix, unless the SE/CI method is set to 'None'. Grouping: Results are grouped by response. Filtering and sorting: Results are neither filtered nor sorted. User's specifications for *response_series* and *impulse_series* are ignored and replaced with the VAR object's *endog_list*.
smat = *name*	(1) Save impulse-response estimates in a named matrix. (2) Save impulse-response standard errors in a named matrix, unless the SE/CI method is set to 'None'. Grouping: Results are grouped by impulse (shock). Filtering and sorting: Results are filtered and sorted according to *response_series* and *impulse_series*.
rmat = *name*	(1) Save impulse-response estimates in a named matrix. (2) Save impulse-response standard errors in a named matrix, unless the SE/CI method is set to 'None'. Grouping: Results are grouped by response. Filtering and sorting: Results are filtered and sorted according to *response_series* and *impulse_series*.
cimat = *name*	(1) Save impulse-response estimates in a named matrix. (2) Save impulse-response confidence intervals in a named matrix, unless the SE/CI method is set to 'None'. Grouping: Results are grouped by impulse (shock). Filtering and sorting: Results are filtered and sorted according to *response_series* and *impulse_series*.
rcimat = *name*	(1) Save impulse-response estimates in a named matrix. (2) Save impulse response estimates and confidence intervals in a named matrix, unless the SE/CI method is set to 'None'. Estimates are interleaved. Grouping: Results are grouped by impulse (shock). Filtering and sorting: Results are filtered and sorted according to *response_series* and *impulse_series*.

Examples

```
var var1.ls 1 4 m1 gdp cpi
var1.impulse(10,m) gdp
```

The first line declares and estimates a VAR with three variables. The second line displays multiple graphs of the impulse responses of GDP.

```
var1.impulse(10,m) gdp @imp m1 @order cpi gdp m1
```

displays the impulse response of GDP to a one standard deviation shock in M1 using a different ordering.

```
var.impulse(irtype=lp)
```

Produces impulse response output for a VAR object called "var" using the sequential local projection impulse response engine.

```
var.impulse(irtype=lpjoint, asym=@month=6, lpband=scheffe)
```

Produces impulse response output for a VAR object called "var" using the joint local projection impulse response engine. The standard error band type is set to Scheffé and the asymmetric effect is set to a dummy variable for the month of June. In addition to standard error bands, asymmetric effects are also displayed as two additional curves; one associated with the dummy variable (@month = 6) being equal to 0 (false) and the other associated with the dummy variable being equal to 1 (true).

Cross-references

See Chapter 46. "Impulse Response Analysis," on page 1023 of *User's Guide II* for a discussion of impulse responses in VARs.

See also `Var::vdecomp` (p. 1207).

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. 1198).
name = *arg*	Save the test statistics in a named matrix object. See below for a description of the statistics contained in the stored matrix.
prompt	Force the dialog to appear from within a program.
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 44. "Vector Autoregression (VAR) Models," on page 939 of *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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User's Guide I* for a discussion of labels.

See also `Var::displayname` (p. 1154).

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

vname = *arg*	Save selected lag orders in named vector. See below for a description of the stored vector.
mname = *arg*	Save lag order criteria in named matrix. See below for a description of the stored matrix.
prompt	Force the dialog to appear from within a program.
p	Print table of lag order criteria.

The "*vname* = " option stores a vector with 5 rows containing the selected lags from the following criteria: sequential modified LR test (row 1), final prediction error (row 2), Akaike information criterion (row 3), Schwarz information criterion (row 4), Hannan-Quinn information criterion (row 5).

The "mname = " option stores a $q \times 6$ matrix, where $q = m + 1$ if there are no exogenous variables in the VAR; otherwise $q = m + 2$. The first $(q - 1)$ rows contain the information displayed in the table view, following the same order. 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 lcpi
show var1.laglen(12,vname=v1)
```

The first line declares and estimates a VAR. The second line computes the lag length criteria up to a maximum of 12 lags and stores the selected lag orders in a vector named V1.

Cross-references

See Chapter 44. "Vector Autoregression (VAR) Models," on page 939 of *User's Guide II* for a discussion of the various criteria and other VAR diagnostics.

See also `Var::testlags` (p. 1203).

ls	Var Methods

Estimate VAR specification.

Syntax

var_name.**ls***(options) lag_pairs endog_list [@ exog_list] [@*restrict *restrict_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".

The *restrict_list* is a comma-separated list of text restrictions in the form described below and in greater detail in "VARs With Linear Constraints" on page 945 of *User's Guide II*.

Restriction text expressions use the following "@" keywords to refer to individual coefficient matrix elements:

@l#(r, c)	Element (r, c) of the lag # coefficient matrix.
@e#(r)	Element r of the exogenous variable # coefficient vector.
@e(X, r)	Element r of the exogenous variable X coefficient vector

Note that the canonical names ("L#", "E#", "E(X)") that refer to lag matrices and exogenous variable vectors are preceded by "@" to avoid ambiguity.

For example, we may have:

```
@L1(1,1) = 0
@L2(2,2) = @L1(3,3) / 2
@L2(1,1) + @L4(2,1) = 1
@E(C, 1) = 0
@E(X, 2) = @E(C, 2)
@E1(1) + @E1(2) = 1
```

In addition, you may use text expressions to refer to parts of lag coefficient matrices and to impose specialized restrictions,

@vec(W) = n_1 , n_2 , n_3 , ...	Restricts all elements of matrix W similar to a pattern matrix. Element ordering matches the vectorization of the matrix, *i.e.*, the elements of the first column, followed by the second column, followed by the third column, *etc.*
@diag(W)	Restricts W to be a diagonal matrix, *i.e.*, off-diagonal elements are zero. The diagonal elements are unrestricted.
@diag(W) = n	Restricts W to be a diagonal matrix with elements on the diagonal restricted to be n.
@lower(W)	Restricts W to be a lower triangular matrix, *i.e.*, elements above the diagonal are zero.
@unitlower(W)	Restricts W to be a unit lower triangular matrix, *i.e.*, elements above the diagonal are zero and elements on the diagonal are one.
@upper(W)	Restricts W to be an upper triangular matrix, *i.e.*, elements below the diagonal are zero.
@unitupper(W)	Restricts W to be a unit upper triangular matrix, *i.e.*, elements below the diagonal are zero and elements on the diagonal are one.
@row(W, r) = n	Restricts the elements in row r of W to be n.
@col(W, c) = n	Restricts the elements in column c of W to be n.

where W is a reference to a canonical matrix name (*e.g.*, "L1", "L3").

When a "@vec" restriction is included in the restriction list, its own list of values must be enclosed in double quotes, as in

```
@vec(W) = "1, 2, 3, 4"
```

Options

General options

noconst	Do not include a constant in exogenous regressors list for VARs.
prompt	Force the dialog to appear from within a program.
p	Print basic estimation results.

Restricted VAR options

noiter	Perform GLS with a single iteration.
m = *integer*	Set maximum number of iterations. If iteration and "m = 0" then estimation will be by OLS.
c = *scalar*	Set convergence criterion.
showopts / -showopts	[Do / do not] display the estimation options in the estimation output.

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.

```
mvar.ls 1 2  dlog(invest) dlog(income) dlog(cons)  @restrict
    @vec(l1) = "na, 0, na, 0, na, na, na, na, na", @vec(l2) = "na,
    na, na, 0, 0, na, na, na, na"
```

estimates a VAR with pattern restrictions on elements of the first and second lag matrices.

Cross-references

See Chapter 44. "Vector Autoregression (VAR) Models," on page 939 of *User's Guide II* for details.

See also `Var::ec` (p. 1155) and `Var::bvar` (p. 1144) for estimation of error correction models and Bayesian VAR estimation.

makecoefs	Var Procs

Creates groups of the coefficient series used to make the BTVCVAR estimation output view.

Creates a series group containing estimates and a series group containing credibility intervals for the coefficients of a BTVCVAR model.

Syntax

var_name.**makecoefs** *c_est_name c_cis_name*

Group names can be passed in as arguments. The first argument is the name for the series group containing estimates. The second argument is the name for the series group containing credibility intervals.

Example

```
var var1.btvcvar 1 1 gdp inflation
var1.makecoefs ests cis
```

The first line declares and estimates a BTVCVAR for GDP and inflation. The second line produces two coefficient series groups, `ests` and `cis`. The `est` group contains coefficient estimates and the `cis` group contains credibility intervals for the coefficients.

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 gcoint
```

The first line estimates a VEC with 2 cointegrating relations. The second line creates a group named GCOINT 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 44. "Vector Autoregression (VAR) Models," on page 939 of *User's Guide II* for details.

See also `Var::coint` (p. 1149).

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. 1159) and `Model::makegroup` (p. 627).

makeess	Var Procs

Outputs effective sample sizes (ESSs) of the draws of the parameters in a BTVCVAR model.

Generates three matrix objects (one per parameter block) containing ESSs. The ESS for the draws of a parameter is equal to the relative numerical efficiency (RNE) times the posterior sample size. Values near the posterior sample size are desirable. ESSs much smaller than the posterior sample size indicate slow mixing. An NA is returned if sample autocorrelation does not taper off sufficiently quickly (i.e., if sample autocorrelation does not fall below 0.05 in the first 100 lags).

The ESS is the number of IID draws that is needed to achieve the same level of numerical precision as the obtained set of MCMC draws. It measures the extent of efficiency loss for using MCMC instead of vanilla Monte Carlo for sample generation.

The ESS, inefficiency factor, and relative numerical efficiency (RNE) convey the same information and only differ in interpretation.

Syntax

var_name.**makeess** *c_ess_name s_ess_name q_ess_name*

Names for the output matrices can be passed in as arguments. ESS matrix names are entered in the order: coefficients (C), observation covariance (S), then process covariance (Q). Untitled matrices are returned if no names are given.

Example

```
var var1.btvcvar 1 1 gdp inflation
var1.makeess
```

The first line declares and estimates a BTVCVAR with two variables. Estimation entails posterior simulation, as the posterior distribution is not known analytically. The second line computes and outputs inefficiency factors of the draws obtained.

Cross-references

See also `Var::makeif` (p. 1182) and `Var::makerne` (p. 1186).

makeif	Var Procs

Outputs inefficiency factors of the draws of the parameters in a BTVCVAR model.

Generates three matrix objects (one per parameter block) containing inefficiency factors. The inefficiency factor of the draws of θ is defined as

$$1 + 2\sum_{l=1}^{L} \rho_\theta(l)\left(1 - \frac{l}{L}\right)$$

where $\rho_\theta(l)$ is the sample autocorrelation in the draws of θ at lag l, and L is the lag at which the sample autocorrelation tapers off. Values near one are desirable. Inefficiency factors much greater than one indicate slow mixing. An NA is returned if sample autocorrelation does not taper off sufficiently quickly (i.e., if sample autocorrelation does not fall below 0.05 in the first 100 lags).

The inefficiency factor can be interpreted as the ratio of the numerical variance of the sample mean computed using MCMC draws and the numerical variance of the sample mean computed using hypothetical IID draws. It measures the extent of efficiency loss for using MCMC instead of vanilla Monte Carlo for sample generation.

The inefficiency factor, relative numerical efficiency (RNE), and effective sample size (ESS) convey the same information and only differ in interpretation.

Syntax

var_name.**makeif** *c_if_name s_if_name q_if_name*

Names for the output matrices can be passed in as arguments. Inefficiency factor matrix names are entered in the order: coefficients (C), observation covariance (S), then process covariance (Q). Untitled matrices are returned if no names are given.

Example

```
var var1.btvcvar 1 1 gdp inflation
```

```
var1.makeif
```

The first line declares and estimates a BTVCVAR with two variables. Estimation requires posterior simulation as the posterior distribution is not known analytically. The second line computes and outputs inefficiency factors of the draws obtained.

Cross-references

See also `Var::makeess` (p. 1181) and `Var::makerne` (p. 1186).

makemodel — Var Procs

Make a model from a var object.

Syntax

var_name.**makemodel***(name)*

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)
```

estimates a VAR and makes a model named VARMOD from the estimated var object. Use the command "show varmod" or "varmod.spec" to open the VARMOD window.

Cross-references

See Chapter 52. "Models," on page 1177 of *User's Guide II* for a discussion of specifying and solving models in EViews.

See also `Var::append` (p. 1138), `Model::merge` (p. 628) and `Model::solve` (p. 640).

makergmprobs — Var Procs

Save the regime probabilities for switching VAR in series in the workfile.

Syntax

var_name.**makergmprobs***(options) series_names*

where *var_name* is the name of an switching VAR object. The series to be saved should be listed following the command name and options, with one name per regime for one up to the number of estimated regimes.

Options

type = *arg* *(default* = "pred"*)*	Type of regime probability to compute: one-step ahead predicted ("pred"), filtered ("filt"), smoothed ("smooth").
n = *arg*	(optional) Name of group to contain the saved regime probabilities.
prompt	Force the dialog to appear from within a program.

Examples

```
var var1.switchls(type=markov) y1 y2 y3
var1.makergmprobs r1 r2
```

saves the one-step ahead regime probabilities for the Markov switching regression estimated in VAR in series R1 and R2 in the workfile

```
var1.makergmprobs(type=filt) f1
```

saves the filtered probabilities for regime 1 in F1.

```
var1.makergmprobs(type=smooth, n=smoothed) s1 s2
```

saved the smoothed probabilities for both regimes in the series S1 and S2, and creates the group SMOOTHED containing S1 and S2.

Cross-references

See Chapter 50. "Switching VAR" of the *User's Guide II* for discussion.

See also `Var::rgmprobs` (p. 1196).

makeresids — Var Procs

Create residual series.

Creates and saves residuals in the workfile from an estimated VAR.

Syntax

var_name.**makeresids***(options) [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

struct	Compute structural residuals.
method = *arg*	Structural residual method (if "struct" option is provided): unit impulses, ignoring correlations among the residuals ("imp = unit"), non-orthogonal, ignoring correlations among the residuals ("imp = nonort"), Cholesky with d.f. correction ("imp = chol"), Cholesky without d.f. correction ("imp = mlechol"), Generalized ("imp = gen"), structural ("imp = struct"), or user specified ("imp = user"). The structural factorization is based on the estimated structural VAR. To use this option, you must first estimate the structural decomposition; see `Var::svar` (p. 1198). For user-specified weights, you must specify the name of the vector/matrix containing the impulses using the "fname = " option.
n = *arg*	Create group object to hold the residual series.

Examples

```
var macro_var.ls 1 4 y m1 r
macro_var.makeresids 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 "Views and Procs of a VAR" on page 952 of *User's Guide II* for a discussion of views and procedures of a VAR.

makerne	Var Procs

Outputs relative numerical efficiencies (RNEs) of the draws of the parameters in a BTVC-VAR model.

Generates three matrix objects (one per parameter block) containing RNEs. The RNE of the draws of a parameter is the reciprocal of their inefficiency factor. Values near one are desirable. RNEs much smaller than one indicate slow mixing. An NA is returned if sample autocorrelation does not taper off sufficiently quickly (i.e., if sample autocorrelation does not fall below 0.05 in the first 100 lags).

The RNE can be interpreted as the ratio of the numerical variance of the sample mean computed using hypothetical IID draws and the numerical variance of the sample mean computed using MCMC draws. It measures the extent of efficiency loss for using MCMC instead of vanilla Monte Carlo for sample generation.

The RNE, inefficiency factor, and effective sample size (ESS) convey the same information and only differ in interpretation.

Syntax

var_name.**makerne** *c_rne_name s_rne_name q_rne_name*

Names for the output matrices can be passed in as arguments. RNE matrix names are entered in the order: coefficients (C), observation covariance (S), then process covariance (Q). Untitled matrices are returned if no names are given.

Example

```
var var1.btvcvar 1 1 gdp inflation
var1.makerne
```

The first line declares and estimates a BTVCVAR with two variables. Estimation requires posterior simulation as the posterior distribution is not known analytically. The second line computes and outputs inefficiency factors of the draws obtained.

Cross-references

See also `Var::makeess` (p. 1181) and `Var::makeif` (p. 1182).

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

bylag	Specify system by lags (default is to order by variables).
n = *name*	Specify name for the object.

Examples

```
var1.makesystem(n=sys1)
```

creates a system named SYS1 from the var object VAR1

Cross-references

See Chapter 43. "System Estimation," on page 897 of *User's Guide II* for a discussion of system objects in EViews.

maketransprobs	Var Procs

Save the regime transition probabilities and expected durations of switching VAR into the workfile.

Syntax

var_name.**maketransprobs***(options) [base_name]*

var_name.**maketransprobs**(**out = mat**, *options) [matrix_name]*

where *VAR_name* is the name of a VAR estimated using switching regression.

- In the first form of the command, *base_name* will be used to generate series names for the series that will hold the transition probabilities or durations. The series names for regime transition probabilities will be of the form *base_name*##, where ## are the indices representing elements of the transition matrix (i, j). The series names for expected durations will be of the form *base_name*# where # corresponds to the regime index. Thus, in a two-regime model, the base name "TEMP" corresponds to the transition probability series TEMP11, TEMP12, TEMP21, TEMP22, and the expected duration series TEMP1, TEMP2.

If *base_name* is not provided, EViews will use the default of "TPROB"

- When the option "output = mat" is provided, the *matrix_name* is the name of the output matrix that will hold the transition probabilities or durations.

  If *matrix_name* are not provided, EViews will default to "TPROB" or the next available name of the form "TPROB##".

  EViews will evaluate the transition probabilities or durations at the date specified by the "obs = " option. If no observation is specified, EViews will use the first date of the estimation sample to evaluate the transition probabilities. Note that if the transition probabilities are time-invariant, setting the observation will have no effect on the contents of the saved results.

Options

type = *arg* *(default* = "trans"*)*	Transition probability results to save: transition probabilities ("trans"), expected durations ("expect").
out = *arg* *(default* = "series"*)*	Output format: series ("series") or matrix ("mat"). If saved as a matrix, only a single transition matrix will be saved using the date specified by "obs = ".
obs = *arg*	Date/observation used to evaluate the transition probabilities if saving results as a matrix ("out = mat"). If no observation is specified, EViews will use the first date of the estimation sample to evaluate the transition probabilities. Note that if the transition probabilities are time-invariant, setting the observation will have no effect on the content of the saved results.
n = *arg*	(optional) Name of group to contain the saved transition probabilities.
prompt	Force the dialog to appear from within a program.

Examples

```
var var1.switchls(type=markov) y1 y2 y3
var1.maketransprobs(n=transgrp) trans
```

saves the transition probabilities in the workfile in the series TRANS11, TRANS12, TRANS21, TRANS22 and creates the group TRANSGRP containing the series.

The command

```
var1.maketransprobs(type=expect) AA
```

saves the expected durations in the series AA1 and AA2.

```
var1.maketransprobs(out=mat) BB
```

saves the transition probabilities in the matrix BB.

Cross-references

See Chapter 50. “Switching VAR” of the *User’s Guide II* for discussion.

See also `Var::transprobs` (p. 1204)

mfvar	Var Methods

Estimate a mixed frequency VAR specification.

Syntax:

var_name.**mfvar***(options) var_lag endog_list [@ exog_list]* **@hf** *high_freq_list*

You must specify the order of the VAR by specifying the maximum lag, and then provide a list of series or groups to be used as endogenous variables.

Exogenous variables such as trends and seasonal dummies in the VAR may then be included by adding “@” 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”.

For the higher-frequency endogenous variables, you should enter the “@hf” keyword, followed by a list of the variables. The syntax for high frequency variables is *pagename\seriesname* where *pagename* is the name of the page containing the series, and *seriesname* is the name of the series. Note also that series expressions are allowed, *e.g.* “mypage\log(x)”

Options

General options

noconst	Do not include a constant in exogenous regressors list.
hfobs = *integer*	Number of low frequency regressors for each high frequency variable (default is the full number of observations based on the two frequencies, *e.g.*, 3 for monthly data in a quarterly specification).
bvar	Use Bayesian estimation (default is U-MIDAS).
last	Use last-period frequency conversion (default is to use the first period).
lambda = *arg*	Set the lambda overall tightness hyper-parameter (only applicable with Bayesian estimation where “bvar” is specified).

uphl = *arg*	Set the upsilon high-low frequency scale hyper-parameter (only applicable with Bayesian estimation where "bvar" is specified).
uplh = *arg*	Set the upsilon low-high frequency scale hyper-parameter (only applicable with Bayesian estimation where "bvar" is specified).
rhoh = *arg*	Set the rho high frequency AR(1) hyper-parameter (only applicable with Bayesian estimation where "bvar" is specified).
rhol = *arg*	Set the rho low frequency AR(1) hyper-parameter (only applicable with Bayesian estimation where "bvar" is specified).
initcov = *(default* = "umidas"*)*	Initial covariance estimate: U-MIDAS ("umidas"), identity matrix ("identity") (only applicable with Bayesian estimation where "bvar" is specified).
kappa = *arg*	Set the kappa exogenous tightness hyper-parameter (only applicable with Bayesian estimation where "bvar" is specified).
draws = *num* *(default* = 100000*)*	Set the number of MCMC draws (only applicable with Bayesian estimation where "bvar" is specified).
burn = *num* *(default* = 0.1*)*	Set the proportion of MCMC draws to use as a burn-in (only applicable with Bayesian estimation where "bvar" is specified).
seed = *num*	Set the seed for the MCMC generator (only applicable with Bayesian estimation where "bvar" is specified).
prompt	Force the dialog to appear from within a program.
p	Print basic estimation results.

Examples

Cross-references

See Chapter 49. "Mixed Frequency VAR," on page 1109 of *User's Guide II* for details.

See also `Var::ls` (p. 1177) and `Var::ec` (p. 1155) for estimation of ordinary VARs and error correction models.

olepush	Var Procs

Push updates to OLE linked objects in open applications.

Syntax

var_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

output	Var Views

Display estimation output.

`output` changes the default object view to display the estimation output (equivalent to using `Var::results` (p. 1196)).

Syntax

var_name.**output**

For BTVCVARs, the following syntax can be used to specify the coefficient series to display in the estimation output view:

var_name.**output** *[@**var** var_list] [@**lag** lag_list] [@**exog** exog_list]*

where *var_list* is a list of (endogenous) variables, *lag_list* is a list of lags, and *exog_list* is a list of exogenous variables. Each list is specified using a key-value pair, where the key (@var, @lag, or @exog) indicates the list type, and the value (*var_list*, *lag_list*, or *exog_list*) is a space-delimited list of items to select.

Leave a list unspecified (i.e., omit the associated key-value pair) to select every item in the list in the order in which they appear at the time of estimation. To specify an empty list, include the key but leave the value blank.

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.

```
var var2.btvcvar 1 2 4 4 gdp infl @ c unemp
var2.output @var gdp @lag 1 4
```

shows the coefficient series corresponding to GDP(-1), GDP(-4), C, and UNEMP for the BTVCVAR VAR2. Note that C and UNEMP are included (selected) because *exog_list* is unspecified. To hide coefficients on exogenous variables, use the following instead:

```
var2.output @var gdp @lag 1 4 @exog
```

Cross-references

See `Var::results` (p. 1196).

postdraws — Var Procs

Puts posterior draws of a BTVCVAR model in a new page.

Creates a copy of a posterior sample and pastes it into a new page. A series represents draws for a particular parameter. EViews will prompt you to run the posterior sampler if draws are not available.

Syntax

var_name.**postdraws** *page_name*

Examples

```
myvar.postdraws mydraws
```

creates a new page called mydraws containing a copy of the posterior sample for the BTVCVAR var object MYVAR.

postresidcov — Var Procs

Posterior residual covariance matrix.

Syntax

var_name.**postresidcov**(*options*)

Options

p	Print the posterior residual covariance matrix.

Examples

```
var1.postresidcov
```

displays the posterior residual covariance matrix of VAR1.

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

name = *arg*	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.
prompt	Force the dialog to appear from within a program.
p	Print the Portmanteau test results.

Examples

```
var var1.ls 1 6 lgdp lm1 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 954 of *User's Guide II* for a discussion of the Portmanteau tests and other VAR diagnostics.

See `Var::arlm` (p. 1140) 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.

Cross-references

See also `Var::output` (p. 1191).

residcor — Var Views

Residual correlation matrix.

Displays the correlations of the residuals from each equation in the var object.

Note that for Bayesian VARs, this command will display the empirical residual correlation—the correlation of the residuals calculated from the posterior means of the coefficients.

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. 1194) and `Var::makeresids` (p. 1184).

residcov — Var Views

Residual covariance matrix.

Displays the covariances of the residuals from each equation in the var object.

Note that for Bayesian VARs, this command will display the empirical residual covariance—the covariance of the residuals calculated from the posterior means of the coefficients.

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. 1194) and `Var::makeresids` (p. 1184).

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

struct	Compute structural residuals.
method = *arg*	Structural residual method (if "struct" option is provided): unit impulses, ignoring correlations among the residuals ("imp = unit"), non-orthogonal, ignoring correlations among the residuals ("imp = nonort"), Cholesky with d.f. correction ("imp = chol"), Cholesky without d.f. correction ("imp = mlechol"), Generalized ("imp = gen"), structural ("imp = struct"), or user specified ("imp = user"). The structural factorization is based on the estimated structural VAR. To use this option, you must first estimate the structural decomposition; see `Var::svar` (p. 1198). For user-specified weights, you must specify the name of the vector/matrix containing the impulses using the "fname = " option.
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. 1184).

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.

rgmprobs	Var Views

Display regime probabilities for a switching VAR.

Syntax

var_name.**rgmprobs***(options) [indices]*

where *var_name* is the name of a switching VAR object. The elements to display are given by the optional *indices* corresponding to the regimes (*e.g.*, "1 2 3" or "2 3"). If *indices* is not provided, results for all of the regimes will be displayed.

Options

type = *arg* *(default* = "pred"*)*	Type of regime probability to compute: one-step ahead predicted ("pred"), filtered ("filt"), smoothed ("smooth").
view = *arg* *(default* = "graph"*)*	Display format: multiple graphs ("graph"), single graph "graph1", sheet ("sheet"), summary ("summary").
prompt	Force the dialog to appear from within a program.
p	Print results.

Examples

```
var var1.switchls(type=markov) y1 y2 y3
var1.rgmprobs
```

displays graphs containing the one-step ahead regime probabilities for the Markov switching VAR estimated in VAR1.

```
var1.rgmprobs(type=filt) 2
```

displays the filtered probabilities for regime 2.

```
var1.rgmprobs(type=smooth, view=graph1)
```

displays the smoothed probabilities for both regimes in a single graph.

Cross-references

See Chapter 50. "Switching VAR" of the *User's Guide II* for discussion.

See also `Var::makergmprobs` (p. 1183).

setattr

Var Procs

Set the object attribute.

Syntax

var_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the "revised" attribute in the object A to the string "never", and extracts the attribute into the string object S.

Cross-references

See "Adding Custom Attributes in the Label View" on page 123 and "Adding Your Own Label Attributes" on page 70 of *User's Guide I*.

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 SVAR Restrictions in EViews" on page 971 of *User's Guide II* for details on specifying restrictions.

Options

You may specify any of the following options:

a = *mat*	Name of the pattern matrix for factorization matrix A.
b = *mat*	Name of the pattern matrix for factorization matrix B.
s = *mat*	Name of the pattern matrix for short-run impulse response matrix S.
f = *mat*	Name of the pattern matrix for long-run impulse response matrix F.
f0 = *arg* (default = 0.1)	Specify the starting values for estimation free parameters: a scalar value, or 's' for user-specified values in the C coefficient object, or 'u' for values randomly drawn from the uniform distribution on [0,1], or 'n' for values randomly drawn from the standard normal distribution. The default is a scalar value of 0.1.
maxiter = *num*, m = *num*	Maximum number of optimization iterations. The default is taken from the global option settings.
conv = *num*, c = *num*	The convergence criterion (lower bound on optimization step size). The default is taken from the global option settings.
trace = *num*, t = *num* (default = 0)	Summarize the ongoing optimization every *num* iterations. Summary information is displayed in an unnamed text object. The default is a trace period of 0, which disables tracing.

fsign	Do not perform sign normalization. See "Sign Restrictions" on page 977 for a description of sign normalization.
nostop	Suppress "Near Singular Matrix" and other error messages during estimation.
preset = *num*, p = *num*	Apply a restriction preset, as described in the SVAR Options Identifying Restrictions dialog. *num* may be 1 through 6, corresponding to the first six preset options.
prompt	Force the dialog to appear from within a program.

Examples

```
var var1.ls 1 4 m1 gdp cpi
matrix(3,3) pata
pata.fill 1, na, na, 0, 1, na, 0, 0, 1
matrix(3,3) patb
pata.fill na, 0, 0, 0, na, 0, 0, 0, na
var1.svar(a=pata,b=patb)
```

The first line declares and estimates a VAR with three variables. We then create the factorization pattern matrices and perform the estimation.

```
var var1.ls 1 8 dy u @
var1.append(svar) @f(2,1)=0
freeze(out1) var1.svar
```

The first line declares and estimates a VAR with two variables without a constant. The next two lines specify a long-run restriction and store the estimation output in a table object named OUT1.

Cross-references

See "Structural (Identified) VARs" on page 968 of *User's Guide II* for a discussion of structural VARs.

switchvar	Var Methods

Estimate a switching VAR specification.

Syntax:

var_name.**switchvar***(options) lag_pairs endog_list [@ exog_list] [* **@prv** *list_of_probability_regressors]*

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

msm	Switching mean specification (default is switching intercept).
exognv	Exogenous variables are non-regime specific (default is for exogenous variables to vary).
endogvary	Lagged endogenous variables are regime specific (default is for the endogenous variables to be non-varying).
noconst	Do not include a constant in exogenous regressors list.
fprobmat = *arg*	Name of fixed transition probability matrix allows for fixing specific elements of the time-invariant transition matrix. Leave NAs in elements of the matrix to estimate. The (i, j) element of the matrix corresponds to $P(s_t = j \mid s_{t-1} = i)$.
initprob = *arg* *(default* = "ergodic"*)*	Method for determining initial Markov regime probabilities: ergodic solution ("ergodic"), estimated parameter ("est"), equal probabilities ("uniform"), user-specified probabilities ("user"). If "initprob = user" is specified, you will need to specify the "userinit = " option.
userinit = *arg*	Name of vector containing user-specified initial Markov probabilities. The vector should have rows equal to the number of states; we expand this to the size of the initial lag state vector where necessary for AR specifications. For use in specifications containing both the "type = markov" and "initprob = user" options.
startnum = *arg* *(default* = 0 or 25*)*	Number of random starting values tried. The default is 0 for user-supplied coefficients (option "s") and 25 in all other cases.
startiter = *arg* *(default* = 10*)*	Number of iterations taken after each random start before comparing objective to determine final starting value.
searchnum = *arg* *(default* = 0*)*	Number of post-estimation perturbed starting values tried.

searchsds = *arg* (*default* = 1)	Number of standard deviations to use in perturbed starts (if "searchnum = ") is specified.
seed = *positive_integer* from 0 to 2,147,483,647	Seed the random number generator. If not specified, EViews will seed random number generator with a single integer draw from the default global random number generator.
rnd = *arg* (*default* = "kn" or method previously set using `rndseed` (p. 577) in the *Command and Programming Reference*).	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").

In addition to the specification options, there are options for estimation and covariance calculation.

Additional Options

optmethod = *arg*	Optimization method: "bfgs" (BFGS); "newton" (Newton-Raphson), "opg" or "bhhh" (OPG or BHHH), "legacy" (EViews legacy). BFGS is the default method.
optstep = *arg*	Step method: "marquardt" (Marquardt); "dogleg" (Dogleg); "linesearch" (Line search). Marquardt is the default method.
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.
cov = *arg*	Covariance method: "ordinary" (default method based on inverse of the estimated information matrix), "huber" or "white" (Huber-White sandwich method).
covinfo = *arg*	Information matrix method: "opg" (OPG); "hessian" (observed Hessian). (Applicable when non-legacy "optmethod = ".)
nodf	Do not degree-of-freedom correct the coefficient covariance estimate.

coef = *arg*	Specify the name of the coefficient vector (if specified by list); the default behavior is to use the "C" coefficient vector.
s	Use the current coefficient values in "C" as starting values (see also `param` (p. 564) of the *Command and Programming Reference*).
s = *number*	Specify a number between zero and one to determine starting values as a fraction of EViews default values (out of range values are set to "s = 1").
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
prompt	Force the dialog to appear from within a program.
p	Print results.

Examples

```
var01.switchvar(type=markov) 1 3 m1 gdp
```

declares and estimates a MSI(2)-VAR(3) with two endogenous variables (M1 and GDP) and a switching constant.

```
var01.switchvar(type=markov, msm) 1 3 m1 gdp
```

estimates the MSM(2)-VAR(3) variant of the same specification..

```
var01.switchvar(type=markov, msm, endogvary, seed=1551063419) 1 3
    m1 gdp
```

estimates a MSMA(2)-VAR(3) specification.

Cross-references

See Chapter 50. "Switching VAR," on page 1121 of *User's Guide II* for details.

See also `Var::ls` (p. 1177) and `Var::ec` (p. 1155) for estimation of ordinary VARs and error correction models.

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 lm1 lcpi
freeze(tab1) var1.testexog(name=exog)
```

The first line declares and estimates a VAR. The second line stores the exclusion test results in a named table TAB1, and stores the Wald statistics in a matrix named EXOG.

Cross-references

See "Diagnostic Views" on page 954 of *User's Guide II* for a discussion of other VAR diagnostics.

See also `Var::testlags` (p. 1203).

testlags	Var Views

Perform lag exclusion (Wald) tests on a VAR.

Syntax

var_name.**testlags***(options)*

Options

name = *arg*	Save the Wald test statistics in named matrix object. See below for a description of the statistics contained in the stored matrix.
p	Print the result of the test.

The "*name=*" option stores 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 lcpi
freeze(tab1) var1.testlags(name=lags)
```

The first line declares and estimates a VAR. The second line stores the lag exclusion test results in a table named TAB1, and stores the Wald statistics in a matrix named LAGS.

Cross-references

See "Diagnostic Views" on page 954 of *User's Guide II* for a discussion other VAR diagnostics.

See also `Var::laglen` (p. 1176) and `Var::testexog` (p. 1202).

transprobs	Var Views

Display regime transition probabilities and expected durations for a switching VAR.

Syntax

var_name.**transprobs***(options)*

where *equation_name* is the name of an equation estimated using switching regression.

Options

type = *arg* *(default* = "summary"*)*	Transition probability results to display: summary ("default"), transition probabilities ("trans"), expected durations ("expect"). The default summary displays the transition matrix and expected regime durations for constant transition probability models, and descriptive statistics for the transition and expected durations for varying probability models.
view = *arg* *(default* = "graph"*)*	Display method: graph ("graph"), spreadsheet ("sheet"), table ("table"). Applicable when displaying the transition probabilities or expected durations ("type = trans" or "type = expect"). The spreadsheet form represents shows the transition probabilities or regime expected durations in columns and observations in rows. The table form displays the transition probabilities or expected durations in a table (in a single matrix for a time-constant model, and individual matrices for a time-varying model).
prompt	Force the dialog to appear from within a program.
p	Print results.

Examples

```
var var1.switchls(type=markov) y1 y2 y3
var1.transprobs
```

displays the default summary of the transition probabilities estimated in VAR1.

The command

```
var1.transprobs(type=trans)
```

displays the transition probabilities in a graph, while

```
var1.transprobs(type=trans, view=sheet)
```

displays the transition probabilities in a spreadsheet, with each row column representing one of the probabilities and each row representing an observation.

```
var1.transprobs(type=trans, view=table)
```

displays the transition probabilities in a table.

```
var1.transprobs(type=expect, view=sheet)
```

displays the expected durations in spreadsheet form.

Cross-references

See Chapter 50. "Switching VAR" of *User's Guide II* for discussion.

See also `Var::maketransprobs` (p. 1187).

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*]

var var_name.**bvar***(trend, n) lag_pairs endog_list [@ exog_list*]

var var_name.**btvcvar***(options) lag_pairs endog_list [@ exog_list]*

var var_name.**mfvar***(options) var_lag endog_list [@ exog_list]* **@hf** *high_freq_list*

var var_name.**switchreg***(options) lag_pairs endog_list [@ exog_list] [* **@prv** *list_of_probability_regressors]*

Declare the var as a name, or a name followed by an estimation method and specification.

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 44. "Vector Autoregression (VAR) Models," on page 939 of *User's Guide II* for a discussion of vector autoregressions and EC models.

See `Var::ls` (p. 1177) for specification of a standard VAR.

See `Var::ec` (p. 1155) for the error correction specification of a VAR.

See `Var::bvar` (p. 1144) for the specification of a Bayesian VAR.

See `Var::btvcvar` (p. 1142) for the specification of a Bayesian Time-varying Coefficient VAR.

See `Var::mfvar` (p. 1189) for the specification of a mixed-frequency VAR.

See `Var::switchvar` (p. 1199) for the specification of a switching VAR.

vdecomp	Var Views

Variance decomposition in VARs.

Syntax

var_name.**vdecomp***(n, options) [response_series] [@comp component_series] [@order ordering_series]*

In the syntax above, *n* is the number of periods over which to compute variance decompositions. When a value for *n* is not provided, it defaults to 10.

You may optionally filter and sort variance decomposition results (for displaying results and outputting results into the workfile) using *response_series* and *component_series*.

- *response_series* is a space-delimited list of series names used to filter and sort results based on the response (decomposition) series. It is set to the VAR object's *endog_list* by default.
- *component_series* is a space-delimited list of series names used to filter and sort results based on the component series, and is preceded by the keyword "*@comp*". It is set to the VAR object's *endog_list* by default.

If using Cholesky factorization, you may specify the Cholesky ordering by listing the order of the series after "*@order*". The default behavior is equivalent to setting *ordering_series* to the VAR object's *endog_list*. Note that every variable that appears in the VAR object's *endog_list* must be included in *ordering_series*.

Legacy Syntax

The following syntax has been deprecated:

var_name.**vdecomp***(n, options) [response_series] [@ @ ordering_series]*

var_name.**decomp***(n, options) [response_series] [@ @ ordering_series]*

While you may use the older forms of the command, mixing the current and legacy syntax is not allowed.

Options

General Options

g	Display combined graphs, with the decompositions for each variable shown in a graph.
m	Display multiple graphs, with each response-component pair shown in a separate graph.
t *(default)*	Show numerical results in table.
imp = *arg* *(default* = "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. 1198). The option "imp = mlechol" is provided for backward compatibility with EViews 3.x and earlier.
se = mcarlo	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).
rep = *integer*	Number of Monte Carlo replications to be used in computing the standard errors. Must be used with the "se = mcarlo" option.
cilevels = *arg* *(default* = "0.95")	Confidence interval coverage: space limited list of numbers between 0 and 1.
uselines	Use lines instead of shading for confidence intervals.
save = *name*	Save variance decomposition estimates in a named matrix. Grouping: Results are grouped by component, unless the byrsp keyword is used in which case results are grouped by response (decomposition). Filtering and sorting: Results are filtered and sorted according to *response_series* and *component_series*.
savese = *name*	Save variance decomposition standard errors in a named matrix. No output will be generated if the SE/CI method is set to 'None'. Grouping: Results are grouped by component, unless the byrsp keyword is used in which case results are grouped by response (decomposition). Filtering and sorting: Results are filtered and sorted according to *response_series* and *component_series*.

saveci = *name*	Save variance decomposition confidence intervals in a named matrix. No output will be generated if the SE/CI method is set to 'None'. Grouping: Results are grouped by component, unless the byrsp keyword is used in which case results are grouped by response (decomposition). Filtering and sorting: Results are filtered and sorted according to *response_series* and *component_series*.
saverci = *name*	Save variance decomposition estimates and confidence intervals in a named matrix. Estimates are interleaved. No output will be generated if the SE/CI method is set to 'None'. Grouping: Results are grouped by component, unless the byrsp keyword is used in which case results are grouped by response (decomposition). Filtering and sorting: Results are filtered and sorted according to *response_series* and *component_series*.
savefse = *name*	Save forecast standard errors in a named matrix. Filtering and sorting: Results are filtered and sorted according to *response_series*.
byrsp	Group results by response (decomposition) instead of by component.
prompt	Force the dialog to appear from within a program.
p	Print results.

Legacy Save (Output to Workfile) Options

The following save options have been deprecated and are only supported for variance decomposition methods introduced before EViews 13.

Mixing current and legacy save options is not allowed.

matbys = *name*	(1) Save variance decomposition estimates in a named matrix. (2) Save forecast standard errors in a named matrix. (3) Save variance decomposition standard errors in a named matrix, unless the SE/CI method is set to 'None'. Grouping: Results are grouped by component. Filtering and sorting: Results are neither filtered nor sorted. User's specifications for *response_series* and *component_series* are ignored and replaced with the VAR object's *endog_list*.
matbyr = *name*	(1) Save variance decomposition estimates in a named matrix. (2) Save forecast standard errors in a named matrix. (3) Save variance decomposition standard errors in a named matrix, unless the SE/CI method is set to 'None'. Grouping: Results are grouped by response (decomposition). Filtering and sorting: Results are neither filtered nor sorted. User's specifications for *response_series* and *component_series* are ignored and replaced with the VAR object's *endog_list*.

Examples

```
var var1.ls 1 4 m1 gdp cpi
var1.vdecomp(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.vdecomp(10,t) gdp @order cpi gdp m1
```

performs the same variance decomposition as above using a different Cholesky ordering.

Cross-references

See "Variance Decomposition" on page 961 of *User's Guide II* for additional details.

See also `Var::impulse` (p. 1167).

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 = *arg*	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 $T \times R^2$ χ^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 227 of *User's Guide II* for a discussion of White's test. For the multivariate version of this test, see "White Heteroskedasticity Test" on page 958 of *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.

Valmap

Valmap (value map). Assigns descriptive labels to values in numeric or alpha series.

Valmap Declaration

valmap.....declare valmap object (p. 1220).

To declare a valmap use the keyword `valmap`, followed by a name

```
valmap mymap
```

Valmap Views

label .....label information for the valmap object (p. 1217).
sheet.....view table of map definitions (p. 1218).
stats.....summary of map definitions (p. 1219).
usage .....list of series and alphas which use the map (p. 1219).

Valmap Procs

append.....append a definition to a valmap (p. 1214).
clearhist.....clear the contents of the history attribute (p. 1214).
clearremarks .....clear the contents of the remarks attribute (p. 1215).
copy.....creates a copy of the valmap (p. 1215).
displayname.....set display name (p. 1216).
olepush.....push updates to OLE linked objects in open applications (p. 1218).
setattr .....set the value of an object attribute (p. 1218).

Valmap Data Members

String values

@attr("*arg*").....string containing the value of the *arg* attribute, where the argument is specified as a quoted string.
@description.....string containing the Valmap object's description (if available).
@detailedtype .....string with the object type: "VALMAP".
@displayname .....string containing the Valmap object's display name. If the matrix has no display name set, the name is returned.
@name.....string containing the Valmap object's name.
@remarks.....string containing the Valmap object's remarks (if available).
@type.....string with the object type: "VALMAP".
@updatetime .....string representation of the time and date at which the Valmap was last updated.

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. The valmap values and labels may be enclosed in quotation marks, if necessary.

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 one or more value/label mappings to a valmap's specification.

Syntax

valmap_name.**append** *value1 label1 [value2 label2...]*

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 valvalmap: 0s are mapped to “no” and 1s are mapped to “yes”. The valmap values and labels may be enclosed in quotation marks, if necessary.

Cross-references

For details, see “Value Maps” on page 235 of *User’s Guide I*.

clearhist	Valmap Procs

Clear the contents of the history attribute.

Removes the valmap’s history attribute, as shown in the label view of the valmap.

Syntax

valmap_name.**clearhist**

Examples

```
v1.clearhist
v1.label
```

The first line removes the history from the valmap V1, and the second line displays the label view of V1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names.

See also `Valmap::label` (p. 1217).

clearremarks	Valmap Procs

Clear the contents of the remarks attribute.

Removes the valmap's remarks attribute, as shown in the label view of the valmap.

Syntax

valmap_name.**clearremarks**

Examples

```
v1.clearremarks
v1.label
```

The first line removes the remarks from the valmap V1, and the second line displays the label view of V1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names.

See also `Valmap::label` (p. 1217).

copy	Valmap Procs

Creates a copy of the valmap.

Creates either a named or unnamed copy of the valmap.

Syntax

valmap_name.**copy**

valmap_name.**copy** *dest_name*

Examples

```
v1.copy
```

creates an unnamed copy of the valmap V1.

```
v1.copy v2
```

creates V2, a copy of the valmap V1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

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 123 of *User’s Guide I* for a discussion of labels and display names.

See also `Valmap::label` (p. 1217).

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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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”:

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

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

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

To clear and then set the units field, use:

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

Cross-references

See “Labeling Objects” on page 123 of *User’s Guide I* for a discussion of labels.

See also Valmap::displayname (p. 1216).

olepush	Valmap Procs

Push updates to OLE linked objects in open applications.

Syntax

valmap_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

setattr	Valmap Procs

Set the object attribute.

Syntax

valmap_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the "revised" attribute in the object A to the string "never", and extracts the attribute into the string object S.

Cross-references

See "Adding Custom Attributes in the Label View" on page 123 and "Adding Your Own Label Attributes" on page 70 of *User's Guide I*.

sheet	Valmap Views

Definitions view of a valmap object.

Syntax

valmap_name.**sheet***(options)*

Options

p	Print the definitions view.

Examples

```
vm1.sheet
```

displays the definitions view of the valmap object VM1.

Cross-references

See "Value Maps" on page 235 of *User's Guide I* for a discussion of value maps.

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 235 of *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 235 of *User's Guide I*.

See also `Series::map` (p. 816) and `Alpha::map` (p. 16).

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 235 of *User's Guide I*.

See also `Series::map` (p. 816) and `Alpha::map` (p. 16).

Vector

Vector. (One dimensional array of numbers).

Vector Declaration

vector .....declare vector object (p. 1263).

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

cov .....compute variance measures for the data in the vector (p. 1228).
edftest.....empirical distribution function tests (p. 1233).
freq.....one-way tabulation (p. 1239).
hist .....descriptive statistics and histogram (p. 1241).
label .....label information for the vector object (p. 1247).
sheet.....spreadsheet view of the vector (p. 1257).
stats.....descriptive statistics (p. 1260).
statby.....statistics by classification (p. 1258).
testby.....equality test by classification (p. 1261).
teststat .....simple hypothesis tests (p. 1262).

Vector Procs

classify.....recode vector into classes defined by a grid, specified limits, or quantiles (p. 1224).
clearcollabels .....clear the column labels in a vector object (p. 1226).
clearhist.....clear the contents of the history attribute (p. 1226).
clearremarks .....clear the contents of the remarks attribute (p. 1227).
clearrowlabels.....clear the row labels in a vector object (p. 1227).
copy.....creates a copy of the vector (p. 1228).
displayname.....set display name (p. 1231).
distdata.....save a matrix containing distribution plot data computed from the vector (p. 1232).

export ..... export vector as Excel 2007 XLSX, CSV, tab-delimited ASCII text, RTF, HTML, Enhanced Metafile, PDF, TEX, or MD file on disk (p. 1235).
fill ..... fill elements of the vector (p. 1238).
getglobalc ..... copy the contents of the workfile C coefficient vector into the vector object (p. 1240).
import ..... imports data from a foreign file into the vector object (p. 1241).
olepush ..... push updates to OLE linked objects in open applications (p. 1248).
read ..... (deprecated) import data from disk (p. 1249).
resample ..... resample from the rows of the vector (p. 1251).
resize ..... resize the vector object (p. 1252).
setattr ..... set the value of an object attribute (p. 1253).
setcollabels ..... set the column label for the vector object (p. 1253).
setformat ..... set the display format for the vector spreadsheet (p. 1254).
setglobalc ..... copy the contents of the vector object into the workfile C coefficient vector (p. 1255).
setindent ..... set the indentation for the vector spreadsheet (p. 1255).
setjust ..... set the horizontal justification for all cells in the spreadsheet view of the vector object (p. 1256).
setrowlabels ..... set the row labels for the vector object (p. 1256).
setwidth ..... set the column width for the vector spreadsheet (p. 1257).
showlabels ..... displays the custom row and column labels of a vector spreadsheet (p. 1258).
write ..... export data to disk (p. 1264).

Vector Graph Views

Graph creation views are discussed in detail in "Graph Creation Command Summary" on page 1267.

area ..... area graph of the vector (p. 1269).
bar ..... bar graph of data against the row index (p. 1275).
boxplot ..... boxplot graph (p. 1279).
distplot ..... distribution graph (p. 1283).
dot ..... dot plot graph (p. 1290).
line ..... line graph of the data against the row index (p. 1298).
qqplot ..... quantile-quantile graph (p. 1306).
spike ..... spike graph (p. 1323).

Vector Data Members

String values

@attr("*arg*")..... string containing the value of the *arg* attribute, where the argument is specified as a quoted string.

@collabels ..... string containing the column label of the vector.

@description..... string containing the Vector object's description (if available).

@detailedtype ..... string with the object type: "VECTOR".

@displayname ..... string containing the Vector object's display name. If the Vector has no display name set, the name is returned.

@name ..... string containing the Vector object's name.

@remarks ..... string containing the Vector object's remarks (if available).

@rowlabels..... string containing the row labels of the vector.

@type ..... string with the object type: "VECTOR".

@updatetime ..... string representation of the time and date at which the Vector was last updated.

Scalar values

(i) ..... *i*-th element of the vector. Simply append "(i)" to the vector name (without a ".").

@rows ..... number of rows in the svector.

Vector values

@droprow(*arg*) ..... Returns the vector with the rows defined by *arg* removed. *arg* may be an integer, vector of integers, string, or svector of strings. Integers correspond to row numbers so that, for example, *arg* = 2 specifies the second row. Strings correspond to row labels so that *arg* = "2" specifies the first row labeled "2".

@irow(*arg*) ..... Returns the indices for the rows defined by *arg* where *arg* is a string or svector of strings. The strings correspond to row labels so that *arg* = "2" specifies the first row labeled "2".

@row(*arg*) ..... Returns the rows defined by *arg*. *arg* may be an integer, vector of integers, string, or svector of strings. Integers correspond to row numbers so that, for example, *arg* = 2 specifies the second row. Strings correspond to row labels so that *arg* = "2" specifies the first row labeled "2".

@t ..... Returns transpose.

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.

classify	Vector Procs

Recode the vector into classes defined by a grid, specified limits, or quantiles.

Syntax

vector_name.**classify**(*options*) *spec* @ *outname [mapname]*

You should follow the `classify` keyword with any desired options, a specification *spec*, 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**min* or 1.05**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* in 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 vector objects containing limit values. The first limit value defines the upper limit of the first interval, and the last limit value defines the lower limit of the last interval. Note that there must be at least one limit value 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

method = *arg* *(default* = "step"*)*	Method for determining classification values: "step"– create a grid from *start* through *end* using the *stepsize*; "bins" – create bins by dividing the region from *start* to *end* into a specified number of bins; "quants" – create bins using the quantile values; "limits" - create bins using the specified limit points.
rightclosed	Bins formed using right-closed intervals. x is defined to be in the bin from a to b if $a < x \leq b$. The default is to use intervals closed on the left.
rangeerr	Generate error if data value is found outside of defined bins. The default is to classify out-of-range values as NAs.
q = *arg* *(default* = "r"*)*	Quantile calculation method. "b" (Blom), "r" (Rankit-Cleveland), "o" (Ordinary), "t" (Tukey), "v" (van der Waerden), "g" (Gumbel). Only relevant where "method = quants".
encode = *arg* *(default* = "index"*)*	Encoding method for output series: "index" – encode as integers from 0 to k where k 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).
prompt	Force the dialog to appear from within a program.
p	Print the results.

Examples

```
vec1.classify 100 200 @ vec1_ct
```

classifies the values of VEC1 into bins of width 100 starting at 200 and ending at the data maximum times 1.05. The classification results are saved in the vector VEC1_CT.

```
vec1.classify(encode=right) 100 200 1100 @ vec1_ct1
```

classifies VEC1 into bins of size 100, from 200 through 1100. The output vector VEC1_CT1 will contain classification values taken from the right endpoints of the classification intervals. Rows with out-of-range values will be assign an NA.

```
vec1.classify(method=bins, rightclosed, rangeerr) 9 200 1100 @
   vec1_ct2
```

defines 9 equally sized bins, starting at 200 and ending at 1100, and classifies the data into the vector VEC1_CT2. The bins are closed on the right, and out-of-range values will generate an error.

```
vec1.classify(method=quants, q=g, keepna) 4 @ vec1_ct3
```

classifies the values of VEC1 into quartiles (using the Gumbel definition) in the vector VEC1_CT3. Rows with NA values in VEC1 will be encoded as 0 in the output vector.

Cross-references

See "Generate by Classification" on page 497 of *User's Guide I* for additional discussion.

clearcollabels	Vector Procs

Clear the column label in a vector object.

Syntax

vector_name.**clearcollabels**

Examples

```
vec1.clearcollabels
```

clears the custom column label from the vector VEC1.

Cross-references

See `Vector::clearrowlabels` (p. 1227).

clearhist	Vector Procs

Clear the contents of the history attribute for vector objects.

Removes the vector's history attribute, as shown in the label view of the vector.

Syntax

vector_name.**clearhist**

Examples

```
v1.clearhist
```

```
v1.label
```

The first line removes the history from the vector V1, and the second line displays the label view of V1, including the now blank history field.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names.

See also `Vector::label` (p. 1247).

clearremarks	Vector Procs

Clear the contents of the remarks attribute.

Removes the vector's remarks attribute, as shown in the label view of the vector.

Syntax

vector_name.**clearremarks**

Examples

```
v1.clearremarks
v1.label
```

The first line removes the remarks from the vector V1, and the second line displays the label view of V1, including the now blank remarks field.

Cross-references

See "Labeling Objects" on page 123 of *User's Guide I* for a discussion of labels and display names.

See also `Vector::label` (p. 1247).

clearrowlabels	Vector Procs

Clear the row labels in a vector object.

Syntax

vector_name.**clearrowlabels**

Examples

```
vec1.clearrowlabels
```

clears the custom row labels from the vector VEC1.

Cross-references

See `Vector::clearcollabels` (p. 1226).

copy	Vector Procs

Creates a copy of the vector.

Creates either a named or unnamed copy of the vector.

Syntax

vector_name.**copy**

vector_name.**copy** *dest_name*

Examples

```
v1.copy
```

creates an unnamed copy of the vector V1.

```
v1.copy v2
```

creates V2, a copy of the vector V1.

Cross-references

See also copy (p. 411) in the *Command and Programming Reference*.

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.

Pearson Correlation

cov	Product moment covariance.
corr	Product moment correlation.
sscp	Sums-of-squared cross-products.
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.
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.
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.
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

wgt = *name* (optional)	Name of series containing weights.
wgtmethod = *arg* *(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.
df	Compute covariances with a degree-of-freedom correction for the mean (for centered specifications), and any partial conditioning variables.
outfmt = *arg* *(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").
prompt	Force the dialog to appear from within a program.
p	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.

Cross-references

For simple forms of the calculation see @cov (p. 767) in the *Command and Programming Reference*.

display	Vector Views

Display table, graph, or spool output in the vector object window.

Display the contents of a table, graph, or spool in the window of the vector object.

Syntax

vector_name.**display** *object_name*

Examples

```
vector1.display tab1
```

Display the contents of the table TAB1 in the window of the object VECTOR1.

Cross-references

Most often used in constructing an EViews Add-in. See "Custom Object Output" on page 231 in the *Command and Programming Reference*.

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 123 of *User's Guide I* for a discussion of labels and display names.

See also Vector::label (p. 1247) and Graph::legend (p. 402).

distdata	Vector Procs

Save a matrix containing distribution plot data computed from the vector.

Saves the data used to display a histogram, kernel density, theoretical distribution, empirical CDF or survivor plot, or quantile plot to the workfile.

Syntax

vector_name.**distdata**(**dtype** = *dist_type, dist_options*) *matrix_name*

saves the distribution plot data specified by *dist_type*, where *dist_type* must be one of the following keywords:

hist	Histogram *(default)*.
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.
theoryqq	Theoretical quantile-quantile plot.

Options

The theoretical quantile-quantile plot type "theoryqq" takes the options described in qqplot (p. 1306) under "Theoretical Options" on page 1308.

For the remaining types, *dist_options* are any of the distribution type-specific options described in distplot (p. 1283).

Note that the graph display specific options such as "fill," "nofill," and "leg," and "noline" are not relevant for this procedure.

You may use the "prompt" option to force the dialog display

prompt	Force the dialog to appear from within a program.

Examples

```
rvec1.distdata(dtype=hist, anchor=0, scale=dens, rightclosed)
   matrix01
```

creates the data used to draw a histogram from the vector VEC1 with the anchor at 0, density scaling, and right-closed intervals, and stores that data in a matrix called MATRIX01 in the workfile.

```
vec1.distdata(dtype=kernel, k=b, ngrid=50, b=.5) matrix02
```

generates the kernel density data computed with a biweight kernel at 50 grid points, using a bandwidth of 0.5 and linear binning, and stores that data in MATRIX02.

```
vec1.distdata(dtype=theoryqq, q=o, dist=logit, p1=.5) matrix03
```

creates theoretical quantile-quantile data from VEC1 using the ordinary quantile method to calculate quantiles. The theoretical distribution is the logit distribution, with the location parameter set to 0.5. The data is saved into the matrix MATRIX03.

Cross-references

For a description of distribution graphs and quantile-quantile graphs, see "Analytical Graph Types," on page 836 of *User's Guide I*.

See also `distplot` (p. 1283) and `qqplot` (p. 1306).

edftest	Vector Views

Computes goodness-of-fit tests based on the empirical distribution function.

Compute Kolmogorov-Smirnov, Lilliefors, Cramer-von Mises, Anderson-Darling, and Watson empirical distribution function tests.

Syntax

vector_name.**edftest***(options)*

Options

General Options

dist = *arg* *(default* = "nomal"*)*	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).
p1 = *number*	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.
p2 = *number*	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.
p3 = *number*	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.
prompt	Force the dialog to appear from within a program.
p	Print test results.

Estimation Options

The following options apply if iterative estimation of parameters is required:

b	Use Berndt-Hall-Hall-Hausman (BHHH) algorithm. The default is Marquardt.
m = *integer*	Maximum number of iterations.
c = *number*	Set convergence criterion. The criterion is based upon the maximum of the percentage changes in the scaled coefficients.
showopts / -showopts	[Do / do not] display the starting coefficient values and estimation options in the estimation output.
s	Take starting values from the C coefficient vector. By default, EViews uses distribution specific starting values that typically are based on the method of the moments.

Examples

The command

```
vec1.edftest
```

uses the default settings to test whether the data in the vector VEC1 comes from a normal distribution. Both the location and scale parameters are estimated from the data in VEC1.

```
vec1.edftest(type=chisq)
```

tests whether the data in VEC1 come from a χ^2 distribution with degrees-of-freedom estimated from the data.

```
freeze(tab1) vec1.edftest(type=chisq, p1=5)
```

tests whether the data in VEC1 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 465 of *User's Guide I* for a description of the goodness-of-fit tests.

See also qqplot (p. 1306).

export	Vector Procs

Export vector to disk as an Excel 2007 XLSX, CSV, tab-delimited ASCII text, RTF, HTML, Enhanced Metafile, LaTeX, PDF, or Markdown file.

Syntax

vector_name.**export***(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.

The base syntax for writing Excel 2007 files is:

vector_name.**export**(*options*) *[path\]file_name [table_description]*

where the *table_description* may contain:

- "range = *arg*", where arg is top left cell of the destination 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.

Options

t = *file_type* (*default* = "csv")	Specifies the file type, where *file_type* may be one of: "excelxml" (Excel 2007 (xml)),"csv" (CSV - comma-separated), "rtf" (Rich-text format), "txt" (tab-delimited text), "html" (HTML - Hypertext Markup Language), "emf" (Enhanced Metafile), "pdf" (PDF - Portable Document Format), "tex" (LaTeX), or "md" (Markdown). Files will be saved with the ".xlsx", ".csv", ".rtf", ".txt", ".htm", ".emf", ".pdf", ".tex", or ".md" extensions, respectively.
s = *arg*	Scale size, where *arg* is from 5 to 200, representing the percentage of the original table size (only valid for HTML or RTF files).
n = *string*	Replace all cells that contain NA values with the specified string. "NA" is the default.
h / -h	Include(/do not include) column and row headers. The default is to not include the headers
prompt	Force the dialog to appear from within a program.

PDF Options

landscape	Save in landscape mode (the default is to save in portrait mode).
size = *arg* (*default* = "letter")	Page size: "letter", "legal", "a4", and "custom".
width = *number* (*default* = 8.5)	Page width in inches if "size = custom".
height = *number* (*default* = 11)	Page height in inches if "size = custom".
leftmargin = *number* (*default* = 0.5)	Left margin width in inches.
rightmargin = *number* (*default* = 0.5)	Right margin width in inches.
topmargin = *number* (*default* = 1)	Top margin width in inches.
bottommargin = *number* (*default* = 1)	Bottom margin width in inches.

LaTeX Options

texspec / -texspec	[Include / Do not include] the full LaTeX documentation specification in the LaTeX output. The default behavior is taken from the global default settings.

Excel Options

mode = *arg*	Specify whether to create a new file, overwrite an existing file, or update an existing file. arg may be "create" (create new file only; error on attempt to overwrite) or "update" (update an existing file, only overwriting the area specified by the range = *table_description*). If the "mode = " option is not used, EViews will create a new file, unless the file already exists in which case it will overwrite it. Note that the "mode = update" option is only available for Excel in 1) Excel versions through 2003, if Excel is installed, and 2) Excel 2007 (xml). Note: Excel does not need to be installed for Excel 2007 writing.

Excel 2007 Options

cellfmt = *arg*	Specify whether to use EViews, pre-existing, or remove cell formatting (colors, font, number formatting when possible, column widths and row heights) for the written range. *arg* may be "eviews" (replace current formatting in the file with the same cell formatting in EViews), "preserve" (leave current cell formatting already in the Excel file), or "clear" (remove current formatting and do not replace).
strlen = *arg* (default = 256)	Specify the maximum the number of characters written for cells containing text. Strings in cells which are longer the max, will be truncated.

Examples

The command:

```
vector1.export myvector
```

exports data in VECTOR1 to a CSV file named "myvector.CSV" in the default directory.

```
vector1.export(h,t=csv, n="NaN") myvector
```

saves the contents of VECTOR1 along with the column and row headers to a CSV (comma separated value) file named "myvector.CSV" and writes all NA values as "NaN".

```
vector1.export(h,t=html, s=50) myvector
```

writes the data of VECTOR1 along with the column and row headers to a HTML file named "myvector.HTM" at half of the original size.

```
vector1.export(n=".", r=B) myvector
```

exports the data in the second column to a CSV file named "myvector.CSV", and writes all NA values as ".".

```
vector1.export(t=excelxml, cellfmt=clear, mode=update) myvector
    range=Country!b5
```

writes the data in VECTOR1 to the preexisting "myvector.XLSX" Excel file to the "Country" sheet at cell B5, where all cell formatting is cleared.

Cross-references

See `Vector::import` (p. 1241).

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 "l" (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

l	Loop repeatedly over the list of values as many times as it takes to fill the vector.
o = *integer* *(default* = 1*)*	Fill the vector starting 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, l) -1
```

Cross-references

See Chapter 11. "Matrix Language," on page 279 of the *Command and Programming Reference* for a detailed discussion of vector and matrix manipulation in EViews.

freq	Vector Views

Compute frequency tables.

The `freq` command performs a one-way frequency tabulation.

Frequencies are computed for all of the rows in the vector. Rows with NAs are dropped unless included by option. You may use options to control automatic binning (grouping) of values and the order of the entries of the table.

Syntax

vector_name.**freq***(options)*

Options

dropna *(default)* / keepna	[Drop/Keep] NA as a category.
v = *integer* *(default* = 1000*)*	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*	(optional) Make bins if average count per distinct value is less than the specified number.
b = *integer* *(default* = 50*)*	Maximum number of categories to bin into if performing automatic binning.
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.

sort = *arg* *(default* = "lohi"*)*	Sort order for entries in the frequency table: high data value to low ("hilo"), low data value to high ("lohi" – *default*), high frequency to low ("freqhilo"), low frequency to high ("freqlohi").
prompt	Force the dialog to appear from within a program.
p	Print the table.

Examples

```
vec1.freq(nov, noa)
```

tabulates all values (no binning) of VEC1, with entries in ascending value order. The table will display counts, percentages, and cumulative frequencies.

```
vec1.freq(v=200, b=50, keepna, noa)
```

tabulates VEC1 including NAs. The observations will be binned if VEC1 has more than 200 distinct values; EViews will create at most 50 equal value-width bins. The number of bins may be smaller than 50.

```
vec1.freq(sort=freqhilo)
```

tabulates VEC1 with the table rows ordered from values with highest frequency to lowest.

Cross-references

See "One-Way Tabulation" on page 467 of *User's Guide I* for a discussion of frequency tables.

getglobalc — Vector Procs

Copy the contents of the workfile C coefficient vector into the vector object.

Syntax

vector_name.**getglobalc**

This function only applies to vectors, rowvectors and coef objects. The contents of the vector will be replaced with the first N elements of the workfile C coefficient vector, where N is the length of the vector object. This may be useful for storing starting values used in estimation.

Examples

```
vector(5) vec1
vec1.getglobalc
```

Creates a vector object with 5 rows, and then copies the first 5 elements of the C vector into it.

hist	Vector Views

Histogram and descriptive statistics of a vector.

The `hist` command displays descriptive statistics and a histogram for the data in the vector.

Syntax

vector_name.**hist***(options)*

Options

p	Print the histogram.

Examples

```
vec1.hist
```

Displays the histogram and descriptive statistics of VEC1.

Cross-references

See "Histogram and Stats" on page 450 of *User's Guide I* for a discussion of the descriptive statistics reported in the histogram view.

See `distplot` (p. 1283) for a more fully-featured and customizable method of constructing histograms and `Vector::stats` (p. 1260) stats for a view with a more extensive set of basic descriptive statistics.

import	Vector Procs

Imports data from a foreign file into the vector object.

Syntax

vector_name.**import***([type=]) source_description import_specification*

- *source_description* should contain a description of the file from which the data is to be imported. The specification of the description is usually just the path and file name of the file, however you can also specify more precise information. See `wfopen` (p. 640) of the *Command and Programming Reference* for more details on the specification of *source_description*.
- The optional "type=" option may be used to specify a source type. For the most part, you should not need to specify a "type=" option as EViews will automatically deter-

mine the type from the filename. The following table summaries the various source formats and along with the corresponding "type = " keywords:

	Option Keywords
Excel (through 2003)	"excel"
Excel 2007 (xml)	"excelxml"
HTML	"html"
Text / ASCII	"text"

- *import_specification* can be used to provide additional information about the file to be read. The details of import_specification will depend upon the type of file being imported.

Excel Files

The syntax for reading Excel files is:

vector_name.**import(type = excel***[xml]*) *source_description [table_description] [variables_description]*

The following *table_description* elements may be used when reading Excel data:

- "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.

- "byrow", transpose the incoming data. This option allows you to read files where the series are contained in rows (one row per series) rather than columns.

The optional *variables_description* may be formed using the elements:

- "colhead = *int*", number of table rows to be treated as column headers.
- "types = ("*arg1*","*arg2*",…)", user specified data types of the series. If types are provided they will override the types automatically detected by EViews. You may use any of the following format keywords: "a" (character data), "f" (numeric data), "d" (dates), or "w" (EViews automatic detection). This option is rarely required.

- "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 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 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.

Excel Examples

```
vec_obj.import "c:\data files\data.xls"
```

loads the active sheet of DATA.XLSX into the VEC_NAME vector object.

```
vec_obj.import "c:\data files\data.xls" range="GDP data"
```

reads the data contained in the "GDP data" sheet of "Data.XLS" into the VEC_OBJ object.

HTML Files

The syntax for reading HTML pages is:

vector_name.**import(type = html)** *source_description [table_description] [variables_description]*

The following *table_description* elements may be used when reading an HTML file or page:

- "table = *arg*", where *arg* specifies which HTML 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.
- "byrow", transpose the incoming data. This option allows you to import files where the series are contained in rows (one row per series) rather than columns.

The optional *variables_description* may be formed using the elements:

- "colhead = *int*", number of table rows to be treated as 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.

HTML Examples

```
vec1.import "c:\data.html"
```

loads into the VEC1 vector object the data located in the HTML file "Data.HTML" located on the C:\ drive

```
vec1.import(type=html) "http://www.tradingroom.com.au/apps/mkt/
    forex.ac" colhead=3
```

loads into a vector object called VEC1 the data with the given URL located on the website site "http://www.tradingroom.com.au". The column header is set to three rows.

Text and Binary Files

The syntax for reading text or binary files is:

vector_name.**import(type** = *arg***)** *source_description [table_description] [variables_description]*

If a *table_description* is not provided, EViews will attempt to read the file as a free-format text file. The following *table_description* elements may be used when reading a text or binary file:

- "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.
- "byrow", transpose the incoming data. This option allows you to import files where the series are contained in rows (one row per series) rather than columns.
- "lastcol", include implied last column. For lines that end with a delimiter, this option adds an additional column.

  When importing a CSV file, lines which have the delimiter as the last character (for example: 'name,description,date,'), EViews normally determines the line to have 3 columns. With the above option, EViews will determine the line to have 4 columns. Note this is not the same as a line containing 'name,description,date'. In this case, EViews will always determine the line to have 3 columns regardless if the option is set.

A central component of the *table_description* element is the format statement. You may specify the data format using the following table descriptors:

- Fortran Format:

  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 Format:

  rformat = "*[n1]Type[Width][.Precision]*, *[n2]Type[Width][.Precision]*, ...)"

  where optional type is "$" for string or "#" for number, and *n1*, *n2*, *n3*, *n4*, *etc*. are the range of columns containing the data.

- C printf/scanf Format:

  cformat = "*fmt*"

where *fmt* follows standard C language (printf/scanf) format rules.

The optional *variables_description* may be formed using the elements:

- "colhead = *int*", number of table rows to be treated as column headers.
- "types = ("*arg1*","*arg2*",...)", user specified data types of the series. If types are provided they will override the types automatically detected by EViews. You may use any of the following format keywords: "a" (character data), "f" (numeric data), "d" (dates), or "w" (EViews automatic detection). This option is rarely used.
- "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.

Text and Binary File Examples (.txt, .csv, etc.)

```
vec2.import c:\data.csv skip=5
```

reads "Data.CSV" into VEC2, skipping the first 5 rows.

```
vec01.import(type=text) c:\date.txt delim=comma
```

loads the comma delimited data DATE.TXT into the VEC01 vector object.

Cross-references

See `Vector::export` (p. 1235).

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 *text*.
s	Sets the source field to *text*.
u	Sets the units field to *text*.
r	Appends *text* 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 123 of *User's Guide I* for a discussion of labels. See also `Vector::displayname` (p. 1231).

olepush — Vector Procs

Push updates to OLE linked objects in open applications.

Syntax

vector_name.**olepush**

Cross-references

See "Object Linking and Embedding (OLE)" on page 977 of *User's Guide I* for a discussion of using OLE with EViews.

read	Vector Procs

Import data from a foreign disk file into a vector.

(This is a deprecated method of importing into a vector. See `Vector::import` (p. 1241) for the supported method.)

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

prompt	Force the dialog to appear from within a program.

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 *default* 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.
skipcol = *integer*	Number of columns to skip. Must be used with the "*rect*" option.
skiprow = *integer*	Number of rows to skip. Must be used with the "rect" option.
comment = *symbol*	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.
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 152 of *User's Guide I* for a discussion and examples of importing data from external files.

See also `Vector::export` (p. 1235).

resample	Vector Procs

Resample from observations in a vector.

Syntax

vector_name.**resample***(options) [output_spec]*

You should follow the `resample` keyword and options and an *output_spec* containing 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

permute	Draw from rows without replacement. Default is to draw with replacement.
weight = *vector_name*	Name of vector to be used for weighted sampling, containing values proportional to the desired row probabilities (importance sampling). The weight vector must have the same number of rows as the source, with non-missing, non-negative values. The weight values need not add up to 1, as EViews will normalize the weights. If no weights are specified, rows will be drawn with equal probability weights.
block = *integer* (default = *1)*	Block length for each draw. Must be a positive integer. The default block length is 1.
withna *(default)*	[Draw / Do not draw] from all rows in the current sample, including those with NAs.
dropna	Do not draw from rows that contain missing values in the current workfile sample.
fixna	Excludes NAs from draws but copies rows containing missing values to the output series.
prompt	Force the dialog to appear from within a program.

- 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

```
vec1.resample vec1_b
```

creates a new vector VEC1 by drawing with replacement from the rows of VEC1 in the current workfile sample. If VEC1_B already exists in the workfile and is a vector object, it will be overwritten. If VEC1_B exists and is not a vector, EViews will error.

```
vec1.resample(weight=wt, suffix=_2) vec1_c
```

The rows in the source vector will be drawn from with probabilities proportional to the corresponding values in the WT vector. WT must have the same number of rows as VEC1 and must have non-missing, non-negative values.

Cross-references

See "Resample" on page 502 of *User's Guide I* for a discussion of the resampling procedure. For additional discussion of wildcards, see Appendix A. "Wildcards," on page 1227 of *User's Guide II*.

See also `@resample` (p. 1071) and `@permute` (p. 1037) in the *Command and Programming Reference* for sampling from matrices.

resize	Vector Procs

Resize the vector object.

Syntax

vector_name.**resize** *rows*

Examples

```
vec1.resize 20
```

resizes the vector VEC1 to 20 rows, retaining the contents of any existing elements and initializing new elements to 0.

setattr	Vector Procs

Set the object attribute.

Syntax

vector_name.**setattr***(attr) attr_value*

Sets the attribute *attr* to *attr_value*. Note that quoting the arguments may be required. Once added to an object, the attribute may be extracted using the `@attr` data member.

Examples

```
a.setattr(revised) never
String s = a.@attr("revised")
```

sets the “revised” attribute in the object A to the string “never”, and extracts the attribute into the string object S.

Cross-references

See “Adding Custom Attributes in the Label View” on page 123 and “Adding Your Own Label Attributes” on page 70 of *User’s Guide I*.

setcollabels	Vector Procs

Set the column label in a vector object.

Syntax

vector_name.**setcollabels** *label1*

Follow the `setcollabels` command with the column label. Note that the column label should not contain spaces unless it is enclosed in quotes.

Examples

```
vec1.setcollabels MyResults
```

sets the column label to “MyResults”.

Cross-references

See also `Vector::setrowlabels` (p. 1256).

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:

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

To specify a format that groups digits into thousands using a comma separator, place a "t" after the format character. For example, to obtain a fixed number of decimal places with commas used to separate thousands, use "ft*[.precision]*".

To use the period character to separate thousands and commas to denote decimal places, use ".." (two periods) when specifying the precision. For example, 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)"), 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. 1257), `Vector::setindent` (p. 1255) and `Vector::setjust` (p. 1256) for details on setting spreadsheet widths, indentation and justification.

setglobalc

Vector Procs

Copy the contents of the vector object into the workfile C coefficient vector.

Syntax

vector_name.**setglobalc**

This function only applies to vectors, rowvectors and coef objects. The contents of the vector will be copied into the first N elements of the workfile C coefficient vector, where N is the length of the vector object. This may be useful for re-specifying starting values for estimation.

Examples

```
vec1.setglobalc
```

Copies the contents of VEC1 into the workfile C vector.

Cross-references

See also `Coef::coef` (p. 26)

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 indentation setttings are taken from the Global Defaults for spreadsheet views ("Spreadsheet Data Display" on page 1019 of *User's Guide I*) at the time the spreadsheet was created.

Examples

```
v1.setindent 2
```

sets the indentation for the vector spreadsheet view to 2.

Cross-references

See `Vector::setwidth` (p. 1257) and `Vector::setjust` (p. 1256) for details on setting spreadsheet widths and justification.

setjust	Vector Procs

Set the horizontal justification for all cells in the spreadsheet view of the vector object.

Syntax

vector_name.**setjust** *format_arg*

where *format_arg* may be set to left, center, right, or auto (strings are left-justified and numbers are right-justified). Default display settings can be set in General Options; see "Spreadsheet Data Display" on page 1019 of *User's Guide I*.

Examples

```
vec1.setjust left
```

left-justifies the cells in the spreadsheet view of the vector VEC1.

Cross-references

See `Vector::setwidth` (p. 1257) and `Vector::setindent` (p. 1255) for details on setting spreadsheet widths and indentation.

setrowlabels	Vector Procs

Set the row labels in a vector object.

Syntax

vector_name.**setrowlabels** *label1 label2 label3....*

Follow the `setrowlabels` command with a space delimited list of row labels. Note that each row label should not contain spaces unless it is enclosed in quotes. If you provide fewer labels than there are rows, EViews will keep the corresponding default row names ("R11", "R12", etc...).

Examples

```
vec1.setrowlabels USA UK FRANCE
```

sets the row label for the first row in vector VEC1 to USA, the second to UK, and the third to FRANCE.

Cross-references

See Vector::setcollabels (p. 1253).

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. 1255) and Vector::setjust (p. 1256) 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.

showlabels	Vector Procs

Displays the custom row and column labels of a vector spreadsheet.

Syntax

vector_name.**showlabels** *mode*

where *mode* is either 0 or 1 where 0 displays the default row and column labels and 1 displays the custom row and column labels (if present).

Examples

```
v1.showlabels 1
```

displays the custom row and column labels for the V1 spreadsheet. If custom labels have not been set the default labels will be displayed.

```
v1.showlabels 0
```

displays the default row and column labels for the V1 spreadsheet.

Cross-references

See `Vector::setcollabels` (p. 1253) and `Vector::setrowlabels` (p. 1256).

statby	Vector Views

Basic statistics by classification.

The `statby` view displays descriptive statistics for the elements of a vector classified into categories by a vector or columns of a matrix.

Syntax

vector_name.**statby***(options) classifier*

You should follow the vector name with a period, the `statby` keyword, and a name (or a list of names) for the vector or matrix containing row values 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.

Options

Options to control statistics to be displayed

sum	Display sums.
med	Display medians.
max	Display maxima.
min	Display minima.
quant = *arg* *(default* = .5*)*	Display quantile with value given by the argument.
q = *arg* *(default* = "r"*)*	Compute quantiles using the specified definition: "b" (Blom), "r" (Rankit-Cleveland), "o" (Ordinary), "t" (Tukey), "v" (van der Waerden), "g" (Gumbel).
skew	Display skewness.
kurt	Display kurtosis.
na	Display counts of NAs.
nomean	Do not display means.
nostd	Do not display standard deviations.
nocount	Do not display counts.

Options to control layout

l	Display in list mode (for more than one classifier).
nor	Do not display row margin statistics.
noc	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* = 1000*)*	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.
nolimit	Remove prompt warning for continuing when the total number of cells is very large.

Other options

prompt	Force the dialog to appear from within a program.
p	Print the descriptive statistics table.

Examples

```
vec1.statby(max, min) vec2
```

displays the mean, standard deviation, maximum, and minimum of the vector VEC1 by (possibly binned) values of the matching length vector VEC2.

Cross-references

See "By-Group Statistics" on page 703 of the *Command and Programming Reference* for a list of functions to compute by-group statistics. See also "Stats by Classification" on page 453 of *User's Guide I* for discussion.

See also `Series::statby` (p. 853) and `boxplot` (p. 1279).

stats	Vector Views

Descriptive statistics.

Computes and displays a table of means, medians, maximum and minimum values, standard deviations, and other descriptive statistics of the vector.

Syntax

vector_name.**stats***(options)*

Options

p	Print the stats table.

Examples

```
vec1.stats
```

displays the descriptive statistics view of the elements of the vector VEC1.

Cross-references

See "Descriptive Statistics & Tests" on page 450 of *User's Guide I* for a discussion of the descriptive statistics views of series.

See also `boxplot` (p. 1279) and `Vector::hist` (p. 1241).

testby

Vector Views

Test equality of the mean, median, or variance of the data in a vector across categories defined by the row values of a vector or matrix.

Syntax

vector_name.**testby***(options) arg1 [arg2 arg2 …]*

Follow the `testby` keyword by a list of the names of the vector or matrix whose rows contain the classifier values.

By default, `testby` will test for equality of means, but you may specify instead tests of medians or variances as an option, choose whether to use balanced or unbalanced samples, and control binning.

Options

mean *(default)*	Test equality of means.
med	Test equality of medians.
var	Test equality of variances.
dropna *(default)*, keepna	[Drop /Keep] NAs as a classification category.
v = *integer* *(default* = 1000*)*	Bin categories if classification series take 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 on the basis of average cell count.
b = *integer* *(default* = 5*)*	Set maximum number of binned categories.
nolimit	Remove prompt warning for continuing when the total number of cells is very large.
prompt	Force the dialog to appear from within a program.
p	Print the test results.

Examples

```
vec1.testby(a=10) vec2
```

tests equality of means of VEC1 across groups classified by values of the matching length vector VEC2, with binning of the average cell count is less than 10.

```
vec1.testby(med, nov) vec2
```

tests equality of medians of VEC1 across groups classified by VEC2, with no automatic binning on the basis of the number of unique VEC2 values.

Cross-references

See "Equality Tests by Classification" on page 459 of *User's Guide I* for a discussion of equality tests.

See also `Series::testby` (p. 857).

teststat	Vector Views

Test simple hypotheses of whether the mean, median, or variance of the elements of a vector are equal to specific values.

Syntax

vector_name.**teststat***(options)*

Specify the type of test and the value under the null hypothesis as an option. You must specify at least one hypothesis.

For tests of means, you may either estimate the variance or specify the variance as an option.

Options

mean = *number*	Test the null hypothesis that the mean equals the specified number.
med = *number*	Test the null hypothesis that the median equals the specified number.
var = *number*	Test the null hypothesis that the variance equals the specified number. The number must be positive.
std = *number*	Test equality of mean conditional on the specified standard deviation. The standard deviation must be positive.
prompt	Force the dialog to appear from within a program.
p	Print the test results.

Examples

```
vec1.teststat(mean=7)
```

tests the null hypothesis that the mean of VEC1 is equal to 7, using an estimated standard deviation.

```
vec1.teststat(mean=7, std=2)
```

tests the null that the mean is 7, using an estimated standard deviation, and also assuming that the standard deviation is known to be 2.

```
vec1.teststat(var=4)
```

tests the null hypothesis that the variance of VEC1 is equal to 4.

Cross-references

See "Descriptive Statistics & Tests" on page 450 of *User's Guide I* for a discussion of simple hypothesis tests.

See also `Series::teststat` (p. 858).

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 11. "Matrix Language," on page 279 of the *Command and Programming Reference* for a discussion of matrices and vectors in EViews.

See also `Coef::coef` (p. 26) and `Rowvector::rowvector` (p. 727).

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.

This routine should realistically only be used in the oft-hand chance that you wish to write into a Lotus file. Improved Excel, text, and other format writing is available in `Vector::export` (p. 1235).

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

prompt	Force the dialog to appear from within a program.

File type

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

If you omit the "*t* = " option, EViews will determine the type based on the file extension. Unrecognized extensions will be treated as ASCII files. For Lotus and Excel spreadsheet files specified without the "*t* = " option, EViews will automatically append the appropriate extension if it is not otherwise specified.

ASCII text files

na = *string*	Specify text string for NAs. Default is "NA".
d = *arg*	Specify delimiter *(default* is tab*)*: "s" (space), "c" (comma).

Spreadsheet (Lotus, Excel) files

letter_number	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 171 of *User's Guide I* for a discussion. See also `Vector::export` (p. 1235) and `Vector::read` (p. 1249).

Appendix A. 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 instead of focusing on specific objects, it describes the ways in which the graph commands may be used with multiple objects. For details on commands to customize existing graphs, see the graph object reference: "Graph" on page 366.

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:

scatpair ..... scatterplot pairs graph (p. 1318).
seasplot ..... seasonal line graph (p. 1322).
spike ..... spike graph (p. 1323).
xyarea ..... XY area graph (p. 1327).
xybar ..... XY bar graph (p. 1330).
xyerrbar ..... XY error bar graph (p. 1332).
xyline ..... XY line graph (p. 1333).
xypair ..... XY line pairs graph (p. 1337).

Graph Creation Object Summary

The graph creation commands may be used with the following EViews data objects:

area ..... coef (p. 22), group (p. 434), matrix (p. 552), series (p. 755), sym (p. 989), vector (p. 1221).
band ..... group (p. 434), matrix (p. 552), sym (p. 989).
bar ..... coef (p. 22), group (p. 434), matrix (p. 552), rowvector (p. 701), series (p. 755), sym (p. 989), vector (p. 1221).
boxplot ..... coef (p. 22), group (p. 434), matrix (p. 552), rowvector (p. 701), series (p. 755), sym (p. 989), vector (p. 1221).
distplot ..... coef (p. 22), group (p. 434), matrix (p. 552), rowvector (p. 701), series (p. 755), sym (p. 989), vector (p. 1221).
dot ..... coef (p. 22), group (p. 434), matrix (p. 552), rowvector (p. 701), series (p. 755), sym (p. 989), vector (p. 1221).
errbar ..... group (p. 434), matrix (p. 552), rowvector (p. 701), sym (p. 989).
hilo ..... group (p. 434), matrix (p. 552), sym (p. 989).
line ..... coef (p. 22), group (p. 434), matrix (p. 552), series (p. 755), sym (p. 989), vector (p. 1221).
pie ..... group (p. 434), matrix (p. 552), rowvector (p. 701), sym (p. 989).
qqplot ..... coef (p. 22), group (p. 434), matrix (p. 552), rowvector (p. 701), series (p. 755), sym (p. 989), vector (p. 1221).
scat ..... group (p. 434), matrix (p. 552), rowvector (p. 701), sym (p. 989).
scatmat ..... group (p. 434), matrix (p. 552), rowvector (p. 701), sym (p. 989).
scatpair ..... group (p. 434), matrix (p. 552), rowvector (p. 701), sym (p. 989).
seasplot ..... coef (p. 22), group (p. 434), matrix (p. 552), rowvector (p. 701), series (p. 755), sym (p. 989), vector (p. 1221).
spike ..... coef (p. 22), group (p. 434), matrix (p. 552), rowvector (p. 701), series (p. 755), sym (p. 989), vector (p. 1221).
xyarea ..... group (p. 434), matrix (p. 552), sym (p. 989).
xybar ..... group (p. 434), matrix (p. 552), rowvector (p. 701), sym (p. 989).

xyerrbar.....group (p. 434), matrix (p. 552).
xyline .....group (p. 434), matrix (p. 552), sym (p. 989).
xypair .....group (p. 434), matrix (p. 552), rowvector (p. 701), sym (p. 989).

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 1341).

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 *type* 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.

Template and printing options

o = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword ("default" - current global defaults, "classic", "modern", "reverse", "midnight", "spartan", "monochrome") or a graph in the workfile.
t = *graph_name*	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 = *key*	Contract the data as specified by *key*, where *key* 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(*quantile*)" - where *quantile* is a number between 0 and 1.

Panel options

The following option applies when graphing panel structured data:

panel = *arg* (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 1341) where the graph has one or more **within** factors and a contraction method other than raw data (see the **contract** option above).

favorlegend	Favor the use of legends over axis labels to describe categories.
elemcommon = *int*	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 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See `Graph::graph` (p. 399) 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 *(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).
rotate	Rotate the graph so the observation axis is on the left.

Template and printing options

o = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword ("default" - current global defaults, "classic", "modern", "reverse", "midnight", "spartan", "monochrome") or a graph in the workfile.
t = *graph_name*	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 = *arg* *(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

```
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, l)
```

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 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See `Graph::graph` (p. 399) 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. 1269).)

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 1341).

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 *type* 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.

Template and printing options

o = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword ("default" - current global defaults, "classic", "modern", "reverse", "midnight", "spartan", "monochrome") or a graph in the workfile.
t = *graph_name*	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 = *key*	Contract the data as specified by *key*, where *key* 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(*quantile*)" - where *quantile* is a number between 0 and 1.

Panel options

The following option applies when graphing panel structured data:

panel = *arg* (*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 1341) where the graph has one or more **within** factors and a contraction method other than raw data (see the **contract** option above).

favorlegend	Favor the use of legends over axis labels to describe categories.
elemcommon = *int*	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,
    bintype=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 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See `Graph::graph` (p. 399) for graph declaration and other graph types.

You may assign labels to the bars in (frozen) graph objects using the `Graph::options` (p. 408) 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 1341).

Options

q = *arg*	Set the quantile method, where *arg* can be: "r" - Rankit-Cleveland, "o" - Ordinary, "v" - van der Waerden, "b" - Blom, "t" - Tukey, "g" - Gumbel.
rotate	Rotate the graph so the observation axis is on the left.

Multiple series options (categorical graph settings will override these options)

m	Plot boxplots in multiple graphs.

Panel options

The following option applies when graphing panel structured data:

panel = *arg* (*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" (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-referencesC

See "Boxplot" on page 852 of *User's Guide I* for a discussion of boxplots. See Chapter 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See `Graph::graph` (p. 399) for graph declaration and other graph types, and `Graph::setbpelem` (p. 416) for a discussion of boxplot customization.

bubble	Group View | Matrix View

Displays a XY..YZ bubble plot.

At least three series must be present in the group. The first series will be plotted on the horizontal axis. The remaining series, aside from the last, will be plotted on the vertical axis. The last series will be used to determine the size of the bubbles.

Syntax

group_name.**bubble***(options)*

Options

Multiple Y-Series options

m	Place bubble plots in multiple graphs (for groups containing more than three series).

Examples

```
group g1 x ser1 ser2 ser3 ser4 z
g1.bubble
```

defines a group G1 containing the six series X, SER1, SER2, SER3, SER4, and Z, and then plots a bubble graph of the series in the group. X is on the horizontal axis, SER1, SER2, SER3, and SER4 are on the vertical axis, and the bubble size is determined by Z.

Cross-references

See Chapter 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See `Graph::Graph` (p. 366) for graph declaration and other graph types.

bubbletrip	Group View | Matrix View

Display a bubble triplet plot.

Groups should contain series in multiples of three (triplets). Series not part of a triplet will be ignored. The first series of each triplet will be plotted on the horizontal axis. The second series of the triplet will be plotted on the vertical axis. The last series of the triplet will be used to determine the size of the bubbles.

Syntax

group_name.**bubbletrip***(options)*

Options

Multiple Series Triplet Options

m	Place bubble plots in multiple graphs. (for groups containing more than two triplets or six series).

Examples

```
group g1 x1 ser1 z1 x2 ser2 z2
g1.bubbletrip
```

defines a group G1 containing the two triplets or six series X1, SER1, Z1 and X2, SER2, Z2. It then plots a bubble graph X1 vs SER1, where Z1 is the bubble size, and X2 vs SER2, where Z2 is the bubble size.

Cross-references

See Chapter 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See `Graph::Graph` (p. 366) for graph declaration and other graph types.

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 1308).

The optional *categorical_spec* allows you to specify a categorical graph (see "Categorical Spec," on page 1341)

Options

Multiple series options

s	Plot in a single graph. (Categorical graph settings will override this option.)

Template and printing options

o = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword ("default" - current global defaults, "classic", "modern", "reverse", "midnight", "spartan", "monochrome") or a graph in the workfile.
t = *graph_name*	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 = *arg* (*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 distribution graph you wish to create in the analytical spec. For a description of distribution graphs, see "Analytical Graph Types," on page 836 of *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 = *arg*	*arg* 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, *e.g.*, kernel density or theoretical distribution, plot.)
binw = *arg*	*arg* specifies the bin width, and may be "eviews" *(default)*, "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), "user" (user-specifed).
binval = *arg*	*arg* specifies the numeric value of the bin width, when the option "binw = user" is specified.
anchor = *arg*	*arg* specifies the anchor position.
rightclosed	Right-closed bin intervals.
nshifts = *int* *(default* = 25*)*	Specifies the number of shift evaluations. (Only applies to average shifted histograms.)
fill	Fill the graph. (Does not apply to the **hist** type.)
nofill	Don't fill the graph. (Does not apply to the **hist** type.)
leg = *arg*	Specify the legend display settings, where *arg* 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 bin width 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

k = *arg* *(default* = "e")	Kernel type: "e" (Epanechnikov), "r" (Triangular), "u" (Uniform), "n" (Normal–Gaussian), "b" (Biweight–Quartic), "t" (Triweight), "c" (Cosinus).
b = *number*	Specify a number for the bandwidth.
b	Bracket bandwidth.
ngrid = *integer* *(default* = 100*)*	Number of grid points to evaluate.
x	Exact evaluation.
fill	Fill the area.
nofill	Don't fill the area.
leg = *arg*	Specify the legend display settings, where *arg* 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

dist = *arg*	*arg* 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 *t*-distribution.
p1 = *int*	Set first parameter.
p2 = *int*	Set second parameter.
p3 = *int*	Set third parameter.
fill	Fill the area.
nofill	Don't fill the area.
leg = *arg*	Specify the legend display settings, where *arg* can be: "def" - default, "n" - none, "s" - short, "det"- detailed.
m = *int*	Set the iterations maximum. (Applies to logistic, extreme max, extreme min, chi-squared, Weibull, gamma or *t*-distributions.)
c = *int*	Sets the convergence criterion. (Applies to logistic, extreme max, extreme min, chi-squared, Weibull, gamma or *t*-distributions.)
s	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 *t*-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 gro1 weight height
gro1.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.

```
gro1.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

q = *arg*	Set the quantile method, where *arg* can be: “r” - Rankit-Cleveland, “o” - Ordinary, “v” - van der Waerden, “b” - Blom, “t” - Tukey, “g” - Gumbel.
n or noci	Do not include confidence intervals.
ci = *number* *(default* = 0.95*)*	Set confidence interval levels.
leg = *arg*	Specify the legend display settings, where *arg* 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 gro1 weight height
gro1.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.

```
gro1.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,
    bintype=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 836 of *User's Guide I*.

See Chapter 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See `Graph::graph` (p. 399) for graph declaration and other graph types.

To save the data from a distribution plot, see `Series::distdata` (p. 780), `Group::distdata` (p. 470), `Vector::distdata` (p. 1232), and `Matrix::distdata` (p. 566).

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 1341).

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 *type* 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 = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword ("default" - current global defaults, "classic", "modern", "reverse", "midnight", "spartan", "monochrome") or a graph in the workfile.
t = *graph_name*	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 = *key*	Contract the data as specified by *key*, where *key* 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(*quantile*)" - where *quantile* is a number between 0 and 1.

Panel options

The following option applies when graphing panel structured data:

panel = *arg* *(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 1341) where the graph has one or more **within** factors and a contraction method other than raw data (see the “contract” option above).

favorlegend	Favor the use of legends over axis labels to describe categories.
elemcommon = *int*	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 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See `Graph::graph` (p. 399) 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 = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword ("default" - current global defaults, "classic", "modern", "reverse", "midnight", "spartan", "monochrome") or a graph in the workfile.
t = *graph_name*	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 = *arg* (*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

```
errbar xhigh xlow xval
```

displays an error bar graph using the series XLOW, XHIGH, and XVAL.

```
group g1 xhigh xlow 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 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See `Graph::graph` (p. 399) 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

o = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword ("default" - current global defaults, "classic", "modern", "reverse", "midnight", "spartan", "monochrome") or a graph in the workfile.
t = *graph_name*	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 = *arg* (*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

```
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 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See `Graph::graph` (p. 399) 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 1341).

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 *type* 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

o = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword ("default" - current global defaults, "classic", "modern", "reverse", "midnight", "spartan", "monochrome") or a graph in the workfile.
t = *graph_name*	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 = *key*	Contract the data as specified by *key*, where *key* 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(*quantile*)" - where *quantile* is a number between 0 and 1.

Panel options

The following option applies when graphing panel structured data:

panel = *arg* (*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), "meanfan" (plot the fan chart of all of the aforementioned 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), "medfan" (plot the fan chart of all the aforementioned quantiles). (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 1341) where the graph has one or more **within** factors and a contraction method other than raw data (see the **contract** option above).

favorlegend	Favor the use of legends over axis labels to describe categories.
elemcommon = *int*	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 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See `Graph::graph` (p. 399) for graph declaration and other graph types.

mixed	Command | Graph Command | Group View | Matrix View | Rowvector View | Sym View

Plots a graph with various graph types.

Syntax

group_name.**mixed***(options) type_list*

The type_list argument controls the types of graphs included. Include a space delimited list of graph type keywords along with the series attached to that type. Available graph types are “line”, “bar”, “area”, “spike”, “band”, “stackedline”, “stackedbar”, “stackedarea”, and “stackedspike”.

Each keyword should be followed by parenthesis containing a comma separated list of series which will be graphed with that type. Series can be specified by name or by a number corresponding to their position in the group.

Options

llast/-llast	Draw all line types on top of all fill types (llast) or below all fill types (-llast).
ab = *type*	Add axis border along data scale, where *type* 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.*)

Template and printing options

o = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
t = *graph_name*	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 = *key*	Contract the data as specified by *key*, where *key* 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(*quantile*)" - where *quantile* is a number between 0 and 1.

Panel options

The following option applies when graphing panel structured data:

panel = *arg* (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

```
group g1 ser1 ser2 ser3 ser4
g1.mixed line(1,3) bar(2,4)
```

defines a group G1 containing the four series SER1, SER2, SER3, and SER4 then plots a mixed type graph of the series in the group, with SER1 and SER3 being shown in line graph form, and SER2 and SER4 in bar graph form.

```
g1.mixed(o=midnight,-lline) stackedarea(ser1, ser2) line(ser3)
    bar(4)
```

creates a mixed type graph of the group G1, using the settings of the predefined template "midnight," applying the *bold* and *wide* modifiers. Series SER1 and SER2 are stacked into an area graph, whereas series SER3 is shown as a line and SER4 as a bar. The lines of SER3 are drawn behind the fill areas of SER1, SER2, and SER4.

Cross-references

See Chapter 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See `Graph::graph` (p. 399) 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 1341).

Options

Template and printing options

o = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword ("default" - current global defaults, "classic", "modern", "reverse", "midnight", "spartan", "monochrome") or a graph in the workfile.
t = *graph_name*	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 = *key*	Contract the data as specified by *key*, where *key* 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(*quantile*)" - where *quantile* is a number between 0 and 1.

Panel options

The following option applies when graphing panel structured data.

panel = *arg* *(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

```
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,

```
g1.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 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See `Graph::graph` (p. 399) 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 1308).

The optional *categorical_spec* allows you to specify a categorical graph (see "Categorical Spec," on page 1341).

Options

Multiple series pair options (categorical graph settings will override these options)

s	Plot in a single graph (applies only to theoretical QQ and symmetry Q-Q graphs).
mult = *mat_type*	Multiple series or column handling: where *mat_type* may be: "pairs" or "p" - pairs, "mat" or "m" - scatterplot matrix, "lower" or "l" - lower triangular matrix.

Template and printing options

o = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword ("default" - current global defaults, "classic", "modern", "reverse", "midnight", "spartan", "monochrome") or a graph in the workfile.
t = *graph_name*	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 = *arg* (*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 836 of *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:

theory	Theoretical quantile-quantile plot.
empirical	Empirical quantile-quantile plot (requires at least two series or columns of a matrix)
symmetry	Quantile quantile symmetry plot.

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

dist = *arg*	*arg* 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 *t*-distribution.
p1 = *int*	Set first parameter.
p2 = *int*	Set second parameter.
p3 = *int*	Set third parameter.
q = *arg*	Set the quantile method, where *arg* can be: "r" - Rankit-Cleveland, "o" - Ordinary, "v" - van der Waerden, "b" - Blom, "t" - Tukey, "g" - Gumbel.
noline	Don't display a fit line.
m = *int*	Set the iterations maximum. (Applies to logistic, extreme max, extreme min, chi-squared, Weibull, gamma, or *t*-distributions.)

c = *int*	Sets the convergence criterion. (Applies to logistic, extreme max, extreme min, chi-squared, Weibull, gamma, or *t*-distributions.)
s	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 *t*-distributions.)
leg = *arg*	Specify the legend display settings, where *arg* can be: "def" - default, "n" - none, "s" - short, "det"- detailed.

Empirical Options

q = *arg*	Set the quantile method, where *arg* can be: "r" - Rankit-Cleveland, "o" - Ordinary, "v" - van der Waerden, "b" - Blom, "t" - Tukey, "g" - Gumbel.
noline	Don't display a regression line.
leg = *arg*	Specify the legend display settings, where *arg* 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 836 of *User's Guide I*.

See Chapter 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See `Graph::graph` (p. 399) 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. 417)).

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 *auxilary_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 1343).

The optional *categorical_spec* allows you to specify a categorical graph (see "Categorical Spec," on page 1341).

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). May not be used with the "s" option.
ab = *type*	Add axis border along data scales, where *type* 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	Place scatterplots in multiple graphs.
mult = *mat_type*	Multiple series or column handling: where *mat_type* 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. 1315) command; using the “pairs” option is the same as using `scatpair` (p. 1318).)
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 = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
t = *graph_name*	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:

contract = *key*	Contract the data as specified by *key*, where *key* 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(*quantile*)" - where *quantile* is a number between 0 and 1.

Panel options

The following option applies when graphing panel structured data.

panel = *arg* (*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 1341) where the graph has one or more **within** factors and a contraction method other than raw data (see the **contract** option above).

favorlegend	Favor the use of legends over axis labels to describe categories.
elemcommon = *int*	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.scat 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.scat within(sex) kernfit linefit
```

displays the same graph along with linear and kernel regression fits for each category.

```
cgrp.scat(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.scat 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 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See `Graph::graph` (p. 399) for graph declaration and other graph types.

For a description of the available fit lines, see "Auxiliary Graph Types," on page 855 of *User's Guide I*.

See `xyline` (p. 1333) for a description of 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. 417)). The `scatmat` graph type is equivalent to using `scat` (p. 1310) 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.**scatmat***(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 *auxilary_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 1343).

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 = *type*	Add axis border along data scales, where *type* 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

l	Plot lower triangular scatterplot matrix.

Template and printing options

o = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword ("default" - current global defaults, "classic", "modern", "reverse", "midnight", "spartan", "monochrome") or a graph in the workfile.
t = *graph_name*	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.

Panel options

The following option applies when graphing panel structured data.

panel = *arg* *(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

```
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(l)
```

shows the portion of the matrix below the diagonal.

```
g1.scatmat(l, 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, l) linefit
```

displays a lower-triangular scatterplot matrix with regression fit for each cross-section, each in an individual frame.

Cross-references

See Chapter 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates. See `Graph::graph` (p. 399) for graph declaration and other graph types.

For a description of the available fit lines, see "Auxiliary Graph Types," on page 855 of *User's Guide I*.

See `xyline` (p. 1333) 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. 417)). The `scatpair` graph type is equivalent to using `scat` (p. 1310) 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 *auxilary_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 1343).

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 = *type*	Add axis border along data scales, where *type* 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 = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword ("default" - current global defaults, "classic", "modern", "reverse", "midnight", "spartan", "monochrome") or a graph in the workfile.
t = *graph_name*	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:

contract = *key*	Contract the data as specified by *key*, where *key* 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(*quantile*)" - where *quantile* is a number between 0 and 1.

Panel options

The following option applies when graphing panel structured data.

panel = *arg* (*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 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates. See `Graph::graph` (p. 399) for graph declaration and other graph types.

For a description of the available fit lines, see "Auxiliary Graph Types," on page 855 of *User's Guide I*.

See xyline (p. 1333) for a description of XY graphs.

seasplot	Command || Graph Command | Group View | Series 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

o = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword ("default" - current global defaults, "classic", "modern", "reverse", "midnight", "spartan", "monochrome") or a graph in the workfile.
t = *graph_name*	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 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 834 of *User's Guide I* for a brief discussion of seasonal line graphs.

See Chapter 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates. See `Graph::graph` (p. 399) for graph declaration and other graph types.

See also `Series::seas` (p. 828), `Series::x11` (p. 885) and `Series::x12` (p. 885).

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 1341).

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).
rotate	Rotate the graph so the observation axis is on the left.
ab = *type*	Add axis border along data scale, where *type* 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.

Template and printing options

o = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword ("default" - current global defaults, "classic", "modern", "reverse", "midnight", "spartan", "monochrome") or a graph in the workfile.
t = *graph_name*	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:

contract = *key*	Contract the data as specified by *key*, where *key* 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(*quantile*)" - where *quantile* is a number between 0 and 1.

Panel options

The following option applies when graphing panel structured data:

panel = *arg* (*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).

favorlegend	Favor the use of legends over axis labels to describe categories.
elemcommon = *int*	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.

```
g1.spike(o=midnight, b, w)
```

creates a spike graph of the group G1, using the settings of the predefined template "midnight", applying the *bold* and *wide* modifiers.

Cross-references

See Chapter 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See `Graph::graph` (p. 399) for graph declaration and other graph types.

xyarea	Command || Graph Command | Group View | Matrix 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 = *type*	Add axis border along data scales, where *type* 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 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 = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword ("default" - current global defaults, "classic", "modern", "reverse", "midnight", "spartan", "monochrome") or a graph in the workfile.
t = *graph_name*	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 = *arg* (*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

```
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. 1310) and `xyline` (p. 1333) are specialized forms of XY graphs.

See Chapter 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See Graph::graph (p. 399) 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 = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
t = *graph_name*	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 = *arg* *(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 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

```
xybar 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. 1310), `xyarea` (p. 1327), `xyline` (p. 1333), and `xypair` (p. 1337) are specialized forms of XY graphs.

See Chapter 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See Graph::graph (p. 399) for graph declaration and other graph types.

xyerrbar	Command || Graph Command | Group View | Matrix View

Display an XY error bar graph view (if possible).

The data must be in the form of a multiple of four series or columns. The first series is the x-axis points. The second series is the high error bar and the third series is the low error bar. The fourth series or column is the data of interest plotted as a symbol.

Syntax

xyerrbar*(options) o1 o2 o3 o4 [o5 ...]*

object_name.**xyerrbar***(options)*

where o1, o2, o3, o4,... are series or group objects.

Options

Template and printing options

o = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
t = *graph_name*	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 = *arg* (*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.)

Basic examples

```
xyerrbar open xhigh xlow xval
```

displays an error bar graph using the series XHIGH, XLOW, and XVAL against the OPEN series.

```
group g1 open xhigh xlow xval
g1.xyerrbar
```

creates an error bar graph view of the four series in G1.

```
g1.xyerrbar(o=magazine, w)
```

displays an xyerrbar graph using the settings of the predefined template "magazine," applying the wide modifier.

Panel examples

```
g1.xyerrbar(panel=individual)
```

displays an xyerrbar graph for each cross-section in a separate frame.

```
g1.xyerrbar(panel=mean)
```

displays an xyerrbar graph formed by computing the means for the series across cross-sections.

Cross-references

See Chapter 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See `Graph::graph` (p. 399) for graph declaration and other graph types.

xyline	Command || Graph Command | Group View | Matrix 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. 417)).

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 *auxilary_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 1343).

The optional *categorical_spec* allows you to specify a categorical graph (see "Categorical Spec," on page 1341).

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). May not be used with the "s" option.
ab = *type*	Add axis border along data scales, where *type* 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 = *mat_type*	Multiple series or column handling: where *mat_type* 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. 1337) 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 = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword (“default” - current global defaults, “classic”, “modern”, “reverse”, “midnight”, “spartan”, “monochrome”) or a graph in the workfile.
t = *graph_name*	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:

contract = *key*	Contract the data as specified by *key*, where *key* 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(*quantile*)" - where *quantile* is a number between 0 and 1.

Panel options

The following option applies when graphing panel structured data.

panel = *arg* *(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. 1310) is a specialized form of an XY graph.

See Chapter 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See `Graph::graph` (p. 399) for graph declaration and other graph types.

xypair	Command || Graph Command | Group View | Matrix View | Rowvector 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. 417)). The `xypair` graph type is equivalent to using `xyline` (p. 1333) 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 *auxilary_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 1343).

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 = *type*	Add axis border along data scales, where *type* 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	Plot XY lines in multiple graphs.

Template and printing options

o = *template*	Use appearance options from the specified template. *template* may be a predefined template keyword ("default" - current global defaults, "classic", "modern", "reverse", "midnight", "spartan", "monochrome") or a graph in the workfile.
t = *graph_name*	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 pair and line settings.

Graph data options

The following option is available in categorical graph settings:

contract = *key*	Contract the data as specified by *key*, where *key* 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(*quantile*)" - where *quantile* is a number between 0 and 1.

Panel options

The following option applies when graphing panel structured data.

panel = *arg* (*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.)

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. 1310) and `xyline` (p. 1333) are specialized forms of XY graphs.

See Chapter 14. "Graphing Data," on page 733 of *User's Guide I* for a detailed discussion of graphs in EViews, and "Templates" on page 925 of *User's Guide I* for a discussion of graph templates.

See `Graph::graph` (p. 399) 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. 1310), `scatmat` (p. 1315), `scatpair` (p. 1318), `xyarea` (p. 1327), `xybar` (p. 1330), `xyline` (p. 1333), `xypair` (p. 1337)) 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

incna	include NA category
inctot	include total category
iscale, cscale	individua/common scale for this factor. The default is individual for the "@series" factor, and common for all others.

iscalex, cscalex	individual/common X axis scale for this factor. The default is individual for the "@series" factor, and common for all others.
iscaley, cscalex	individual/common Y axis scale for this factor. The default is individual for the "@series" factor, and common for all others.
bintype = *type*	bin type, where *type* can be: "auto" (default), "quant" - quantile binning, "value" - value binning, "none" - forces no binning.
bincount = *int*	*int* is the number of quantile bins or maximum number of value bins.
dispname	use display name in labels
label = *key*	*key* can be: "auto" (default), "value" - factor value only, "both" - factor name and value.
ncase = *key*	sets the capitalization for factor names in labels, where *key* can be: "upper", "lower", "title". The default is to preserve case.
vcase = *key*	sets the capitalization for factor values in labels, where *key* 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,
    bintype=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 855 of *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 (local) fit line.
orthreg	Add an orthogonal regression line.
cellipse	Add a confidence ellipse.
user	Add a user-specified line.

Each fit line type has its own set of options, to be entered in *type_options*:

To save the data from selected auxiliary graph types in the workfile, see `Group::distdata` (p. 470), and `Matrix::distdata` (p. 566).

Linefit Options

yl	Take the natural log of first series or column, y.
yi	Take the inverse of y.
yp = *number*	Take y to the power of the specified number.
yb = *number*	Take the Box-Cox transformation of y with the specified parameter.
xl	Take the natural log of x.
xi	Take the inverse of x.
xp = *number*	Take x to the power of the specified number.
xb = *number*	Take the Box-Cox transformation of x with the specified parameter.
xd = *integer*	Fit a polynomial of x up to the specified power.
m = *integer*	Set number of robustness iterations.
leg = *arg*	Specify the legend display settings, where *arg* 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

k = *arg* *(default* = "e"*)*	Kernel type: "*e*" (Epanechnikov), "*r*" (Triangular), "*u*" (Uniform), "*n*" (Normal–Gaussian), "*b*" (Biweight–Quartic), "*t*" (Triweight), "*c*" (Cosinus).
b = *number*	Specify a number for the bandwidth.
b	Bracket bandwidth.
ngrid = *integer* *(default* = 100*)*	Number of grid points to evaluate.
x	Exact evaluation of the polynomial fit.
d = *integer* *(default* = 1*)*	Degree of polynomial to fit. Set "*d* = 0" for Nadaraya-Watson regression.
leg = *arg*	Specify the legend display settings, where *arg* 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

d = *integer* *(default* = 1*)*	Degree of polynomial to fit.
b = *fraction* *(default* = 0.3*)*	Bandwidth as a fraction of the total sample. The larger the fraction, the smoother the fit.
b	Bracket bandwidth span.
s	Symmetric neighbors. Default is nearest neighbors.

u	No local weighting. Default is local weighting using tricube weights.
m = *integer*	Set number of robustness iterations.
x	Exact (full) sampling. Default is Cleveland subsampling.
neval = *integer* *(default* = 100*)*	Approximate number of data points at which to compute the fit (if performing Cleveland subsampling).
leg = *arg*	Specify the legend display settings, where *arg* 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

leg = *arg*	Specify the legend display settings, where *arg* can be: "def" - default, "n" - none, "s" - short, "det"- detailed.

Orthreg Examples

```
group gg weight height length volume
gro1.scatmat(l) orthreg
```

displays the orthogonal regression fit for each pair in the lower-triangle scatterplot matrix.

Cellipse Options

size = *arg*	Specify the confidence levels.
c	Use χ^2 distribution to compute the confidence ellipses. The default is to use the F-distribution.
leg = *arg*	Specify the legend display settings, where *arg* 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.

User Options

A user specified line can be specified either using a pair of data points (where you specify the X and Y values for each point, or using a simple line specification with a Y-intercept, slope and, optionally, transformation value. Entering data point values overrides and simple line options.

x1 = *arg*	Set the X (horizontal) value for the first data point.
y1 = *arg*	Set the Y (vertical) value for the first data point.
x2 = *arg*	Set the X (horizontal) value for the second data point.
y2 = *arg*	Set the Y (vertical) value for the second data point.
icept = *arg*	Set simple line Y-intercept of value. Default is 0
slope = *arg*	Set simple line slope. Default is 0.
xl	Use a logarithmic transformation on the X series.
yl	Use a logarithmic transformation on the Y series.
xi	Use an inverse transformation on the X series.
yi	Use an inverse transformation on the Y series.
xp = *arg*	Use a power transformation, with power equal to *arg* on the X series.
yp = *arg*	Use a power transformation, with power equal to *arg* on the Y series.
xb = *arg*	Use a Box-Cox transformation of order *arg* on the X series.
yb = *arg*	Use a Box-Cox transformation of order *arg* on the Y series.
xd = *arg*	Use a polynomial transformation of order *arg* on the X series.

User Examples

```
group gro1 age income
```

```
gro1.scat user(x1=3, y1=4, x2=10, y2=15)
```

Draws a user specified straight line joining the two points (4,3) and (15,10).

```
gro1.scat user(icept=5, slope=0.5, xp=2)
```

Draws a user specified line with an intercept of 5, a slope of 0.5 and a power transformation on the X series.

Appendix B. Object Command Summary

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

autospec ..... System (p. 1029).
axis ..... Graph (p. 366).
band..... Graph (p. 366), Group (p. 434), Matrix (p. 552), Sym (p. 989).
bar ..... Coef (p. 22), Graph (p. 366), Group (p. 434), Matrix (p. 552), Rowvector (p. 701), Series (p. 755), Sym (p. 989), Vector (p. 1221).
bdstest..... Series (p. 755).
binary ..... Equation (p. 51).
block..... Model (p. 604).
boundscheck..... Model (p. 604).
boundstest..... Equation (p. 51).
boxplot..... Coef (p. 22), Graph (p. 366), Group (p. 434), Matrix (p. 552), Rowvector (p. 701), Series (p. 755), Sym (p. 989), Vector (p. 1221).
bpf ..... Series (p. 755).
breakls ..... Equation (p. 51).
breakspec ..... Equation (p. 51).
breaktest ..... Equation (p. 51).
btvcvar..... Var (p. 1133).
bubble..... Graph (p. 366), Group (p. 434), Matrix (p. 552), Rowvector (p. 701), Sym (p. 989).
bubbletest..... Series (p. 755).
bubbletrip..... Graph (p. 366), Group (p. 434), Matrix (p. 552), Rowvector (p. 701), Sym (p. 989).
buroot ..... Series (p. 755).
bvar ..... Var (p. 1133).
cause..... Group (p. 434).
cdtest ..... Equation (p. 51), Series (p. 755).
cellipse..... Equation (p. 51), Logl (p. 535), Pool (p. 648), Sspace (p. 903), System (p. 1029).
censored..... Equation (p. 51).
checkbounds..... Model (p. 604).
checkderivs ..... Logl (p. 535).
chow..... Equation (p. 51).
cinterval ..... Equation (p. 51).
classify ..... Series (p. 755), Vector (p. 1221).
clear..... Text (p. 1112), Userobj (p. 1122).
clearcollabels ..... Coef (p. 22), Matrix (p. 552), Rowvector (p. 701), Svector (p. 966), Sym (p. 989), Vector (p. 1221).

tor (p. 966), Sym (p. 989), System (p. 1029), Table (p. 1070), Text (p. 1112), Userobj (p. 1122), Valmap (p. 1213), Var (p. 1133), Vector (p. 1221).

copyrange..... Table (p. 1070).

copytable..... Table (p. 1070).

cor ..... Group (p. 434), Matrix (p. 552), Sym (p. 989), Vector (p. 1221).

correl ..... Equation (p. 51), Group (p. 434), Series (p. 755), System (p. 1029), Var (p. 1133).

correlsq ..... Equation (p. 51).

count..... Equation (p. 51).

cov..... Group (p. 434), Matrix (p. 552), Sym (p. 989), Vector (p. 1221).

cross ..... Group (p. 434).

cvardecomp ..... Equation (p. 51).

cvgraph ..... Equation (p. 51).

datalabel..... Graph (p. 366).

datelabel..... Graph (p. 366).

ddloadtmpl ..... Group (p. 434),

ddrowopts ..... Group (p. 434).

ddsavetmpl..... Group (p. 434).

ddtabopts ..... Group (p. 434).

decomp ..... Var (p. 1133).

define..... Pool (p. 648).

delete ..... Graph (p. 366), Pool (p. 648).

deletecells..... Table (p. 1070).

deletecol..... Table (p. 1070).

deleteobs ..... Group (p. 434).

deleterow ..... Table (p. 1070).

depfreq..... Equation (p. 51).

derivs ..... Equation (p. 51), System (p. 1029).

describe..... Pool (p. 648).

did ..... Equation (p. 51).

didcs ..... Equation (p. 51).

didgbdecomp ..... Equation (p. 51).

didmakeeq..... Equation (p. 51).

didtrends ..... Equation (p. 51).

digraph..... Model (p. 604).

display ..... Alpha (p. 6), Coef (p. 22), Equation (p. 51), Factor (p. 271), Geomap (p. 329), Graph (p. 366), Group (p. 434), Link (p. 523), Logl (p. 535), Matrix (p. 552), Model (p. 604), Pool (p. 648),

errbar ..... Graph (p. 366), Group (p. 434), Matrix (p. 552), Rowvector (p. 701), Sym (p. 989).
estcov..... System (p. 1029).
ets..... Series (p. 755).
exclude..... Model (p. 604).
export..... Coef (p. 22), Matrix (p. 552), Rowvector (p. 701), Svector (p. 966), Sym (p. 989), Vector (p. 1221).
extract ..... Spool (p. 932), Userobj (p. 1122).
facbreak ..... Equation (p. 51).
factnames ..... Factor (p. 271).
factor ..... Factor (p. 271).
fetch..... Pool (p. 648).
fill ..... Coef (p. 22), Matrix (p. 552), Rowvector (p. 701), Series (p. 755), Svector (p. 966), Sym (p. 989), Vector (p. 1221).
fiml ..... System (p. 1029).
fit ..... Equation (p. 51), Var (p. 1133).
fitstats ..... Factor (p. 271).
fitted ..... Factor (p. 271).
fixcol..... Table (p. 1070).
fixrow ..... Table (p. 1070).
fixrowcol ..... Table (p. 1070).
fixedtest ..... Equation (p. 51), Pool (p. 648).
flatten..... Spool (p. 932).
fliptype..... Model (p. 604).
forcavg ..... Series (p. 755).
forceval ..... Series (p. 755).
forecast ..... Equation (p. 51), Sspace (p. 903), Var (p. 1133).
formatshapelabel... Geomap (p. 329).
freeze ..... Graph (p. 366), Table (p. 1070).
freq ..... Alpha (p. 6), Group (p. 434), Matrix (p. 552), Rowvector (p. 701), Series (p. 755), Svector (p. 966), Vector (p. 1221).
frml..... Alpha (p. 6), Series (p. 755).
fsel..... Factor (p. 271).
funbias ..... Equation (p. 51).
funbw ..... Equation (p. 51).
funcoef..... Equation (p. 51).
funci ..... Equation (p. 51).
funcov..... Equation (p. 51).
funtest..... Equation (p. 51).

label..... Alpha (p. 6), Coef (p. 22), Equation (p. 51), Factor (p. 271), Geomap (p. 329), Graph (p. 366), Group (p. 434), Link (p. 523), Logl (p. 535), Matrix (p. 552), Model (p. 604), Pool (p. 648), Rowvector (p. 701), Sample (p. 738), Scalar (p. 747), Series (p. 755), Spool (p. 932), Sspace (p. 903), String (p. 959), Svector (p. 966), Sym (p. 989), System (p. 1029), Table (p. 1070), Text (p. 1112), Userobj (p. 1122), Valmap (p. 1213), Var (p. 1133), Vector (p. 1221).
laglen ..... Var (p. 1133).
lambdacoefs ..... Equation (p. 51).
lambdacoefs ..... Equation (p. 51).
lambdacoefs ..... Equation (p. 51).
lambdacoefs ..... Equation (p. 51).
leftmargin..... Spool (p. 932).
legend ..... Geomap (p. 329), Graph (p. 366).
liml ..... Equation (p. 51).
line ..... Graph (p. 366), Coef (p. 22), Group (p. 434), Matrix (p. 552), Series (p. 755), Sym (p. 989), Vector (p. 1221).
link ..... Geomap (p. 329), Link (p. 523).
linkto ..... Link (p. 523).
list..... String (p. 959).
load..... Geomap (p. 329).
loadings..... Factor (p. 271).
logit ..... Equation (p. 51).
logl..... Logl (p. 535).
lrcov..... Group (p. 434).
lrvar ..... Series (p. 755).
ls..... Equation (p. 51), Pool (p. 648), System (p. 1029), Var (p. 1133).
lvageplot ..... Equation (p. 51).
makeattrser..... Geomap (p. 329).
makecoefs ..... Var (p. 1133).
makecoint..... Equation (p. 51), Var (p. 1133).
makederivs..... Equation (p. 51).
makeendog..... Sspace (p. 903), System (p. 1029), Var (p. 1133).
makeess ..... Var (p. 1133).
makeif..... Var (p. 1133).
makefilter ..... Sspace (p. 903).
makefunobj ..... Equation (p. 51).
makegarch..... Equation (p. 51), System (p. 1029).

movetitle ..... Geomap (p. 329).
msa ..... Factor (p. 271).
msg ..... Model (p. 604).
multibreak ..... Equation (p. 51).
name ..... Graph (p. 366), Spool (p. 932).
newsimpact ..... Equation (p. 51).
nyblom ..... Equation (p. 51).
observed ..... Factor (p. 271).
olepush ..... Alpha (p. 6), Coef (p. 22), Equation (p. 51), Factor (p. 271), Geomap (p. 329), Graph (p. 366), Group (p. 434), Link (p. 523), Logl (p. 535), Matrix (p. 552), Model (p. 604), Pool (p. 648), Rowvector (p. 701), Sample (p. 738), Scalar (p. 747), Series (p. 755), Spool (p. 932), Sspace (p. 903), String (p. 959), Svector (p. 966), Sym (p. 989), System (p. 1029), Table (p. 1070), Text (p. 1112), Userobj (p. 1122), Valmap (p. 1213), Var (p. 1133), Vector (p. 1221).
options ..... Geomap (p. 329), Graph (p. 366), Spool (p. 932).
ordered ..... Equation (p. 51).
orthogtest ..... Equation (p. 51).
outliers ..... Equation (p. 51), Series (p. 755).
output ..... Equation (p. 51), Factor (p. 271), Logl (p. 535), Pool (p. 648), Sspace (p. 903), System (p. 1029), Var (p. 1133).
override ..... Model (p. 604).
pace ..... Factor (p. 271).
pancov ..... Series (p. 755).
panpcomp ..... Series (p. 755).
partcor ..... Factor (p. 271).
pcomp ..... Group (p. 434), Matrix (p. 552).
pf ..... Factor (p. 271).
pie ..... Graph (p. 366), Group (p. 434), Matrix (p. 552), Rowvector (p. 701), Sym (p. 989).
pmghausmantest ... Equation (p. 51).
pool ..... Pool (p. 648).
postdraws ..... Var (p. 1133).
postresidcov ..... Var (p. 1133).
predict ..... Equation (p. 51).
print ..... Spool (p. 932).
printview ..... Model (p. 604).
probit ..... Equation (p. 51).

rotate ..... Factor (p. 271).
rotateclear ..... Factor (p. 271).
rotateout..... Factor (p. 271).
rowvector ..... Rowvector (p. 701).
sample ..... Sample (p. 738).
save ..... Geomap (p. 329), Graph (p. 366), Spool (p. 932), Table (p. 1070).
scalar ..... Scalar (p. 747).
scat ..... Graph (p. 366), Group (p. 434), Matrix (p. 552), Rowvector (p. 701), Sym (p. 989).
scatmat..... Graph (p. 366), Group (p. 434), Matrix (p. 552), Rowvector (p. 701), Sym (p. 989).
scatpair ..... Graph (p. 366), Group (p. 434), Matrix (p. 552), Rowvector (p. 701), Sym (p. 989).
scenario..... Model (p. 604).
scenlist ..... Model (p. 604).
scores..... Factor (p. 271).
seas..... Series (p. 755).
seasplot ..... Graph (p. 366), Group (p. 434), Series (p. 755).
seasuroot..... Series (p. 755).
series..... Series (p. 755).
set..... Sample (p. 738).
setattr..... Alpha (p. 6), Coef (p. 22), Equation (p. 51), Factor (p. 271), Geomap (p. 329), Graph (p. 366), Group (p. 434), Link (p. 523), Logl (p. 535), Matrix (p. 552), Model (p. 604), Pool (p. 648), Rowvector (p. 701), Sample (p. 738), Scalar (p. 747), Series (p. 755), Spool (p. 932), Sspace (p. 903), String (p. 959), Svector (p. 966), Sym (p. 989), System (p. 1029), Table (p. 1070), Text (p. 1112), Userobj (p. 1122), Valmap (p. 1213), Var (p. 1133), Vector (p. 1221).
setbounds..... Model (p. 604).
setbpelem ..... Graph (p. 366).
setcell..... Table (p. 1070).
setcollabels ..... Coef (p. 22), Matrix (p. 552), Rowvector (p. 701), Svector (p. 966), Sym (p. 989), Vector (p. 1221).
setcolwidth ..... Table (p. 1070).
setconvert..... Series (p. 755).
setelem..... Graph (p. 366).
setfillcolor..... Geomap (p. 329), Group (p. 434), Series (p. 755), Table (p. 1070).

smooth ..... Series (p. 755).
solve ..... Model (p. 604).
solveopt..... Model (p. 604).
sort ..... Alpha (p. 6), Graph (p. 366), Group (p. 434), Series (p. 755), Table (p. 1070).
spec ..... Logl (p. 535), Model (p. 604), Sample (p. 738), Sspace (p. 903), System (p. 1029).
spike ..... Coef (p. 22), Graph (p. 366), Group (p. 434), Matrix (p. 552), Rowvector (p. 701), Series (p. 755), Sym (p. 989), Vector (p. 1221).
spool ..... Spool (p. 932).
srcoefs..... Equation (p. 51).
sspace ..... Sspace (p. 903).
statby ..... Series (p. 755).
statefinal ..... Sspace (p. 903).
stategraphs ..... Sspace (p. 903).
stateinit ..... Sspace (p. 903).
stats ..... Coef (p. 22), Group (p. 434), Matrix (p. 552), Rowvector (p. 701), Series (p. 755), Sym (p. 989), Valmap (p. 1213), Vector (p. 1221).
stl ..... Series (p. 755).
stochastic ..... Model (p. 604).
stom..... Group (p. 434), Series (p. 755).
stomna ..... Group (p. 434), Series (p. 755).
store..... Pool (p. 648).
strconstant..... Equation (p. 51).
string..... String (p. 959).
strlinear..... Equation (p. 51).
strnonlin..... Equation (p. 51).
structure..... Factor (p. 271), Sspace (p. 903).
strwgts ..... Equation (p. 51).
sur ..... System (p. 1029).
svar..... Var (p. 1133).
svector ..... Svector (p. 966), Text (p. 1112).
switchreg..... Equation (p. 51).
switchvar..... Var (p. 1133).
sym..... Sym (p. 989).
symmtest..... Equation (p. 51).
system..... System (p. 1029).

Index

C

D

E

F

G

H

I

Q

R

S

T

U

V

W

X

Z