# RATS

VERSION 9.0

## *ADDITIONAL TOPICS*

# Additional Topics

This PDF supplement provides additional information on various aspects of using RATS, including details on new features added to the program, more information on the RATS interface, details on the supported data file formats, coverage of some additional topics in econometrics, documentation on several little-used instructions no longer included in the *Reference Manual*, and more.

Estima
1560 Sherman Ave., Suite 510
Evanston, IL 60201

Published in the United States of America

# Table of Contents

# Additional Topics

# 1. New Features

This section lists new features added to RATS since the original version 9.0 release. These features are documented in the relevant sections of the PDF versions of the main RATS manuals, but not in the printed versions.

# 2. The RATS Interface

This section includes information on the menu operations and toolbar icons available in RATS, tells you how to use the *File–Preferences* operation to customize the way RATS behaves, and describes how to use RATS in batch mode.

## 2.1 Menu Operations

The RATS Editor provides eight menus: *File*, *Edit*, *View*, *Data/Graphics*, *Statistics*, *Time Series*, *Window*, and *Help*. Most of these are very similar to those available in many other Windows applications. We describe the menu operations briefly below. The on-line help provides additional details on many of the menu operations.

### The File Menu

The *File* menu provides standard file-handling and related operations.

*New*
: opens a new text window or a new RATS format data file window.

*Open...*
: opens an existing file. Using the file-type field in the dialog box, you can open text files (such as RATS program and procedure files), RATS graph files, or RATS format data files. For text files, the new window will only be flagged as the input window if there are no other open text windows.

*Recent Files*
: provides a list of the most recently used files. The list is preserved from one session to the next so you can easily reopen files you were using in a previous session.

*Close*
: closes the active window. RATS will give you a chance to save any changes before closing the window. If you close the input window, RATS does a *Clear Memory* operation (see next page).

*Merge...*
: inserts the contents of an existing file into the active window.

*Save*
: saves the entire contents of the active window to disk using the current file name (*Save* will prompt you for a file name if the file hasn't been saved previously).

*Save As...*
: saves the contents of the active window to disk using a filename you select. If you have text selected in the window, *Save As* will ask if you want to save the selected block, or the entire window.

*Export...*
: exports the contents of the active window to a file. This operation is supported for most windows other than text windows (where you just use *File–Save As...*). *Export* allows you to choose from a variety of file formats.

*Page Setup...*
: selects a printer and allows you to set printer–specific options.

*Print*
prints the contents of the active text or graph window. To print just a portion of a text window, select the desired lines before doing *Print*.

*Properties*
Shows more detail about the current window (such as a more complete name).

*Directory...*
lets you change the default drive and directory for the current session.

*Recent Directories*
allows you to set the default directory by selecting from a list of recently-used directories.

*Preferences...*
customizes the behavior of the RATS editor. See page page AT-22 for details. On Macintosh versions, this operation is located on the *RATS* menu.

*Batch Mode*
puts RATS into "batch" mode. See page page AT-26.

*Clear Memory*
clears the current data and settings (all data series, other variables, **CALENDAR** and **ALLOCATE** settings, etc.) from memory. Use *Clear Memory* (or the  icon) when you want to enter a new program or re-execute a program from the beginning. This does *not* delete any text, close any windows, or close open data files.

*Exit*
closes all windows and exits the RATS program. RATS will give you an opportunity to save any changes before quitting. On Macintosh versions, this operation is called *Quit* and is located on the *RATS* menu.

## The Edit Menu

The *Edit* menu provides cut and paste and other editing functions. See the on-line help for further details.

*Undo*
: "undoes" the recent changes. For example, if you have *Cut* a block of text, *Undo* will replace the text. *Undo* also works with data editing windows. You can "undo" up to the twenty most recent changes in each window. Note that *Undo* never reverses calculations—it applies only to editing operations.

*Redo*
: "redoes" the results of the most recent *Undo*.

*Cut*
: for a text window, this deletes the selected text and copies it to the Clipboard. This can also be used to delete series from a RATS format file when applied to a Datafile List window (the deleted series can be pasted to another file if desired).

*Copy, Copy Special..., Copy as TeX*
: copies the selected information to the clipboard for later pasting, but does not delete it from the window. You can use *Copy* to copy text, graphs, or the contents of a spreadsheet window, or to copy-and-paste series from one RATS format file to another. For some types of windows, the *Copy Special...* operation will be available allowing you to copy information to the clipboard in a particular format. Currently, this is limited to the ability to copy the contents of Report Windows as a TeX tabular item (the menu operation will show *Copy as TeX* to indicate this).

*Paste*
: copies the contents of the clipboard into the active window. Most commonly used for copying text, you can also copy-and-paste (or cut-and-paste) series from one RATS format file to another using Datafile List windows. Note that you cannot paste graphics into RATS.

*Delete*
: deletes the selected text (does *not* copy it to the clipboard).

*Select All*
: selects every line in the window. To execute every instruction in the input window, you can do *Edit–Select All* (or click on the "Select all" icon) and hit `<Enter>` (or click the "Run" icon).

*Find...*
: searches for a string of text. You can search from the top of the window, or forward or backward from the current cursor position. Turn on the "Case Sensitive" check box for a case-sensitive search.

*Find Next*
repeats the most recent *Find...* operation.

*Replace...*
does a find-and-replace operation, locating a specified string in the active window and replacing it with another string.

*Set Marker*
sets a marker at the current line so you can easily return to it.

*Goto Marker*
Moves the cursor to the position marked using "Set Marker".

*Select To Marker*
Selects (highlights) all lines of text in the active window between the current position and the one marked using "Set Marker".

*To Lower Case*
converts the selected lines to all lower-case.

*Format comments*
Takes the selected line(s) and turns them into a block of comments (lines prefixed with an * that RATS will ignore rather than execute as commands). The lines will be reformatted to be of approximately uniform length. You can change this length with the "Comment Length" box on the *Preferences* dialog box.

*Comment-Out Lines*
This turns the selected lines into comment lines that will be ignored by RATS when executing instructions. This is done by adding the * symbol to the beginning of each line.

*Uncomment Lines*
The opposite of the *Comment-Out Lines* operation, this removes the * symbols from the beginning of the selected lines.

*Indent Lines*
Indents the selected lines. You can change the width of the indenting with the "Indenting" box on the *Preferences* dialog box.

*Unindent Lines*
Removes indenting from the selected lines. The amount of indenting that will be removed is the indenting amount set via the *Preferences* operation.

*Show Last Error*
moves the cursor to the line (instruction) that caused the most recent error.

## The View Menu

The *View* menu allows you to view lists of the variables and series currently stored in memory and quickly generate simple graphs and statistical tables for series displayed in the Series Window or a RATS data file window. The first three operations are only available for certain types of windows.

*Reset List*
When a RATS format file window is active, this allows you to change which series are listed, and whether the series are sorted by name.

*Show as B&W/Show as Color*
When a Graph Window is active, you can use these operations to change the graph from color mode to black and white mode or vice versa.

*Change Layout*
allows you to customize the appearance of the window. Only applies to some types of windows.

*Series Window*
brings up a window showing all the series currently in memory. From this window, you can use the other *View* menu operations or the toolbar icons to graph the series, display basic statistics, export series, edit the series, and more.

*Standard Functions*
opens the Function Lookup dialog box, which you can use to browse through the built-in functions, check the function syntax, and paste a function into the input window.

*Standard Variables*
displays a list all of the reserved variables defined by various RATS instructions. You can sort the list alphabetically or by category.

*User Variables*
displays a list of all the variables defined by the user in the current session.

*All Symbols*
displays a list of all global variable and function names, including reserved variables and functions defined by RATS as well as user-defined variables.

The following are available when the Series Window or a RATS format file window is active. These serve the same function as the corresponding toolbar icons (see page page AT-20). The last two operations apply only for a list of series in memory.

*Time Series Graph*
produces a time series graph of the selected series.

*Histogram*
produces a histogram plot of the selected series (can only be applied to one series at a time).

*Box Plot*
produces box plots for the selected series.

*Autocorrelations*
produces a plot of the autocorrelations and partial autocorrelations for the series.

*Statistics*
produces basic descriptive statistics for the selected series

*Covariance Matrix*
produces a covariance\correlation matrix of the selected series

*Data Table*
displays the selected series as columns in a spreadsheet-style window, similar to doing **PRINT** with the WINDOW option. You cannot edit series values using this data view—double click on a series if you want to edit values in the series.

## The Data/Graphics Menu

*Calendar*
: provides an easy way to enter the desired periodicity and starting date for the current session. Note: If you will be using one of the Data Wizards, there is no need to do the Calendar Wizard.

*Data (RATS format)*
: The Data Wizards simplify the process of setting up your RATS session, including setting the **CALENDAR** and **ALLOCATE** commands, opening a data file, and reading in data. If you will be working with a RATS format file, first do *File–Open RATSData* to open the data file, select the series you want to read in from the file, and then do the *Data (RATS Format) Wizard* to read in the data.

*Data (Other formats)*
: If you will be working with any format other than RATS format, just select the *Data (Other Formats) Wizard*. From the dialog box, select the desired file format and open the file you want to read.

*Create Series (Data Editor)*
: opens a series editing window which you can use to create a new series in memory.

*Trend/Seasonals/Dummies*
: generates trend series, seasonal and 0/1 dummies, or a series of draws from a random normal distribution.

*Transformations*
: provides a convenient way to do a variety of data transformations, from taking logs or square roots to generating growth rates, and more.

*Differencing*
: allows you to difference series using regular, seasonal, and/or fractional differences.

*Filter/Smooth*
: implements several types of time-series filters.

*Moving Window Statistics*
: extracts means, variances, fractiles, and extreme values from a moving window of data.

*X11 (Pro version)*
: implements the X11 seasonal adjustment process. Note: This feature is only supported in the Professional version of RATS.

*Graph Settings*
: This generates a **GRPARM** instruction, allowing you to set the font, style, and size of labels used on graphs.

*Graph*
generates time series graphs.

*Scatter (X-Y) Graph*
generates scatter plots.

*Data Browsers: FRED Browser*
accesses the online FRED economic database maintained by the St. Louis Federal Reserve (http://research.stlouisfed.org/fred2/), and displays a list of the main database categories in a new window. You can double-click on categories and sub-categories to drill down to a series list. You can download series to a RATS format file (by dragging and dropping onto a RATS data file window), or bring the series into memory by dragging and dropping onto the Series Window.

*Data Browsers: Haver DLX*
This opens a Series List Window showing the contents of an already-open Haver DLX database file. From this window, you can view/graph data, bring series into memory, or export series to another file. Note: You must do `OPEN HAVER database`, or have a database specified in the Data Sources section of the *Preferences* dialog, prior to selecting this operation.

*Data Browsers: FAME Database*
This opens a dialog box you can use to open a FAME database stored on your computer or connect to a database stored on a remote server. After opening a database, this displays a Series List Window showing the contents of the FAME database.

## The Statistics Menu

*Univariate Statistics*
generates descriptive statistics for a single series.

*Covariance Matrix*
computes a variance/covariance matrix for a set of series.

*Linear Regressions*
performs a variety of linear regressions, including OLS, instrumental variables, GMM, and AR1.

*Limited/Discrete Dependent Variables*
performs estimations involving limited and discrete dependent variable models, such as probit and tobit.

*Panel Data Regressions*
performs panel data regressions, including fixed and random effects.

*Recursive Least Squares*
does recursive least squares regressions.

*Regression Tests*
does hypothesis testing on the most recently completed regression.

*Nonparametric Regression*
provides an interface to the **NPREG** instruction for non-parametric regressions.

*(Kernel) Density Estimation*
provides an interface to the **DENSITY** instruction for estimating the density function of a series.

*Equation/FRML definition*
defines an equation or formula. For formulas, you can define the list of nonlinear parameters.

## The Time Series Menu

The Time Series menu offers operations for analyzing time series, including estimation a variety of time series models, including ARIMA and ARCH/GARCH models.

*Autocorrelations*
computes regular and partial autocorrelations of a series.

*Cross Correlations*
computes cross correlations and covariances for a pair of series.

*Unit Root Tests*
provides access to a number of unit root testing procedures.

*ARCH/GARCH(Univariate)*
estimates univariate ARCH, GARCH, and related models.

*ARCH/GARCH(Multivariate)*
estimates multivariate ARCH, GARCH, and related models.

*Box-Jenkins (ARIMA) Models*
estimates ARIMA models.

*Exponential Smoothing*
implements exponential smoothing.

*VAR (Setup/Estimate)*
defines and estimates VAR models.

*VAR (Forecast/Analyze)*
generates forecasts, impulse responses, variance decompositions, and historical decompositions for VARs and other multi-equation models.

*Cointegration Test*
interface to several cointegration test procedures provided with RATS, including Enders-Granger (**@EGTEST**), Phillips-Ouiliaris (**@POTEST**), Johansen Likelihood (**@JOHMLE**), and Gregory-Hansen (**@GREGORYHANSEN**).

*Cointegration Estimation*
Provides access to several procedures for estimating cointegration models.

*CATS Cointegration*
If you have version 2 of the CATS cointegration analysis procedure (sold separately), you can use this operation to execute the procedure.

*Single-Equation Forecasts*
generates forecasts for a single equation.

## The Window Menu

The *Window* menu offers several operations for working with RATS windows. Also, a list of all open windows appears at the bottom of the *Window* menu. You can switch to a window by selecting it from this list.

*Tile Horizontal*
: "tiles" the open windows so they are all visible on the screen, using a horizontal orientation (windows will be wider than they are tall).

*Tile Vertical*
: tiles the open windows, using a vertical orientation (windows will generally be taller than they are wide).

*Cascade*
: stacks the open windows so the title bar of each window is visible.

*Close All*
: closes all open windows. RATS will give you a chance to save any changes to text windows.

*Keep Open on Close All*
: flags the top window so that it will remain open if you do *Window–Close All*.

*Close All Graphs*
: closes all graph windows. You will not be asked if you want to save changes, so if you want to save any of your graphs, be sure to do so before selecting this operation.

*Close All Reports*
: closes all spreadsheet windows created using `WINDOW` options.

*Use for Input*
: makes the active text window the input window.

*Use for Output*
: makes the active text window the output window.

*Create Scratch Window*
: creates a new text window you can use for quick calculations or to test instructions without affecting the contents of the main input and output window(s). Output from any instruction executed in a scratch window is always directed only to the scratch window. The status and content of existing input/output windows are not affected.

  However, any changes to values of variables, matrices, or series will affect those in the main workspace.

  You can open more than one scratch window at a time.

*Paste AND Execute*

This is a "switch" controlling how the other Wizards behave. When this is switch is on, clicking OK in a Wizard dialog box automatically executes the instruction(s) generated by the Wizard. If you would prefer that the Wizard only insert the instructions without executing them, select *Paste AND Execute* to turn off this setting.

*Report Windows*

This displays a list of the recently generated "report" windows. Selecting one of these will display the report in a new Report Window. You can use this to re-display the formatted output produced by instructions like **REPORT**, **LINREG**, **TABLE**, **STATISTICS**, and many others. From the report windows, you can copy, paste, and export information to spreadsheet programs and other applications.

## 2.2 Toolbar Icons

The RATS toolbars provide icons you can click on to accomplish a variety of tasks. The toolbar appears at the top of the screen, just below the menu bar. Note that many of the operations are also available via the *View* menu.

### Text Editing Windows

The following icons appear in the toolbar when a text window is active. Some icons are disabled (and appear dimmed) under certain conditions.

| | |
|---|---|
|  (Open) | Shortcut for *File–Open*. Opens a file. |
|  (Save) | Shortcut for *File–Save*. Saves the contents of the active window. |
|  (Print) | Shortcut for *File–Print*. Prints the contents of the active window. |
|  (Cut) | Shortcut for *Edit–Cut*. Removes the selected content from the window and copies it to the clipboard. |
|  (Copy) | Shortcut for *Edit–Copy*. Copies the selected content to the clipboard. |
|  (Paste) | Shortcut for *Edit–Paste*. Pastes content from the clipboard into the current window. |
|  (Find) | Shortcut for *Edit–Find*. Allows you to search for text. |
|  (Functions) | This opens the *Function Wizard*, which displays a list of all of the built-in functions in RATS. See Section 1.2 of the *User's Guide*. |
|  (Input) | This designates the active (front) window as the Input window. It is equivalent to the *Window–Use for Input* operation. See page Int-7 of the *Introduction* for more on the Input and Output icons. |
|  (Output) | This designates the active (front) window as the Output window. It is equivalent to the *Window–Use for Output* operation. |
|  | These set up split Input and Output windows, tiled horizontally or vertically.. |
|  (Select All) | Shortcut for the *Edit–Select All* operation. Selects all the text or items in the active window. |
|  (Run) | Runs the selected instructions, or the instruction on the cursor line, if any (equivalent to hitting <Enter>). Disabled if the active window is not the input window. See page Int-7 of the *Introduction* for more on the Run and Stop icons. |
|  (Stop) | While RATS is processing instructions, the "Run" icon changes to this "Stop" icon. You can click on this to halt the processing of the instructions. |

 (Ready/Local) Switches RATS from Ready mode to Local mode. The icon will change to "Local/Ready" (), indicating that RATS is in Local mode. Click the icon again to change back to Ready mode. You can also hit <Ctrl>+L to switch modes. See page Int-7 of the *Introduction* for details.

 (Clear Mem.) This clears the memory. It is equivalent to *File–Clear Memory*.

## Graph Windows

When a graph window is active, RATS will display the "Save", "Print", and "Copy" icons described above, along with the additional icons for fixing and un-fixing the graph proportions and for switching between color and grayscale mode. See page Int-56 of the *Introduction* for details.

## Series Window

See page Int-38 in the *Introduction* for details on the icons available when viewing the Series Window (a list of the series stored in memory), which you can display by selecting *View–Series Window*.

## RATS Data File Windows

When you display a window listing the series stored in a RATS format file, the same icons available for the Series Window described on page Int-38 are available, except for the "Covariance Matrix" and "View Data" buttons. The following additional buttons are also available (see Int-112 for additional information).

 (Copy) Copies the selected series to the clipboard (can be pasted into another RATS format file).

 (Un/Redo) Undo and Redo operations.

 (Rename) allows you to change the name of the selected series.

The stand-alone RATSDATA utility program uses the same window format. You may find it handy to use RATSDATA for converting data to and from RATS format files, or for quickly creating or editing series or generating graphs (see the Help system in RATSDATA for details).

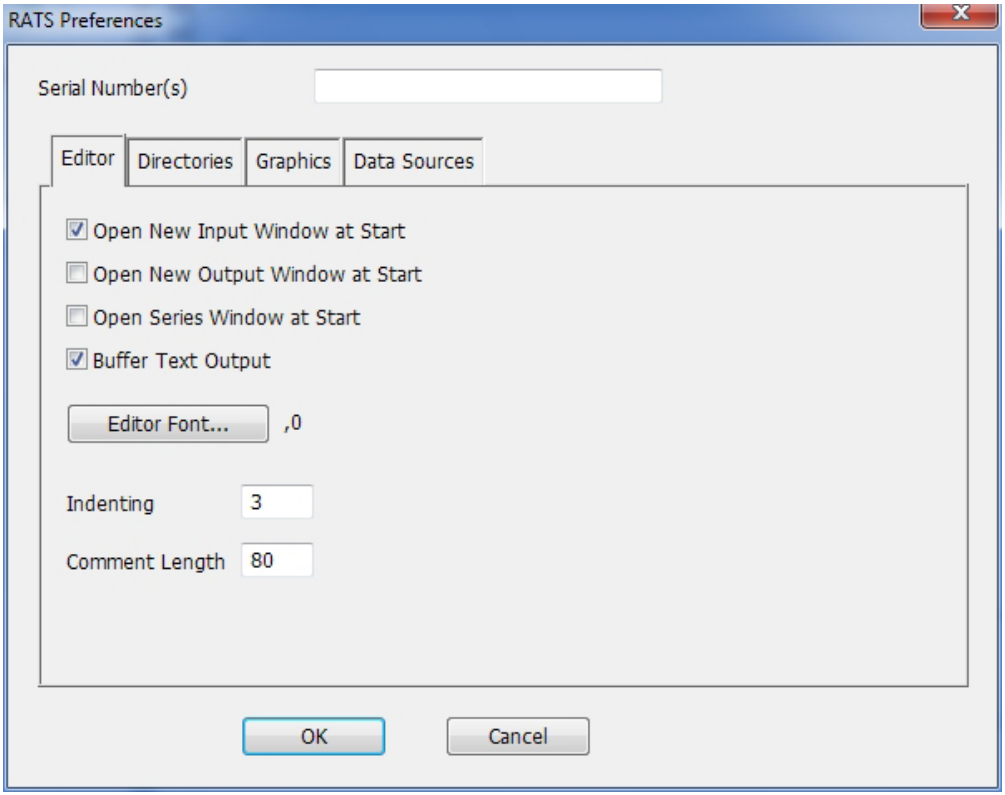
However, virtually all of the functionality of RATSDATA has now been incorporated in RATS itself. So, in most cases you will probably find it easiest to just use the *File–New* and *File–Open* operations in RATS to do this kind of work.

## Series Edit Windows

See page Int-14 in the *Introduction* for details on the icons available when editing a series (either a series in memory or a series on a RATS format file).

## 2.3 Preference Settings: Customizing the RATS Editor

The *Preferences...* operation on the *File* menu (on the *RATS* menu for Macintosh users) lets you customize various aspects of the RATS environment. Selecting the *Preferences* operation brings up this dialog box:




You can use the **Serial Number** field at the top to store your RATS serial number in the registry. This allows you to view the serial number at any time by doing *Help–About RATS* (*RATS–About RATS* on Macintosh). For convenience, you can also store serial numbers for other RATS products (such as the CATS cointegration procedures)—just separate the serial numbers by semi-colon (`;`).

Note: in some cases, the installation program may be able to save the serial number in the registry automatically.

The remaining fields are described on the following pages.

### The Editor Tab

The first tab in the dialog box, labelled "Editor", controls various settings affecting the text editing features of the interactive mode interface.

#### Open New Input Window at Start

This controls whether or not RATS automatically opens up an empty text window, designated as the input window, when you start the program.

#### Open New Output Window at Start

This controls whether or not RATS automatically opens up an empty text window set as the output window when you start the program. If "Input Window" is checked but "Output Window" is not, one new window will open, set as both the input window and the output window. If both boxes are checked, RATS opens one input window and one output window, tiled horizontally.

#### Open Series Window at Start

Turn this on if you want RATS to automatically open the Series Window (displaying the series in memory) when the program starts.

#### Buffer Text Output

Turn this *off* if you want RATS to display each line of output as it is generated. Turn it *on* if you want RATS to save output in a "buffer", and only update the screen periodically, which allows it to execute much more quickly. You will want to leave this on unless you need to see each line of output immediately as it is generated.

#### Editor Font Button

You can change the font and type size used in the RATS editor by clicking on this button. This controls the appearance of text both on the screen and when printed. The current font and size are shown in the dialog to the right of the button.

#### Indenting

This controls how far (in terms of number of characters) lines will be indented if you use the *Indent Lines* operation on the *Edit* menu.

#### Comment Length

This sets the maximum line width (in number of characters) that will be used if you do the *Format Comments* operation on the *Edit* menu.

### The Directories Tab

The "Directories" tab offers the following fields:

#### Default Directory

The Default Directory field allows you to change the default startup directory. If you select a directory here, and answer "Yes" when RATS asks if you want to make the changes permanent, that directory will be the default directory each time you start the program. Note: use the *File–Directory* operation if you just want to change the default directory for the current session.

#### Procedure Directories

If you try to execute a procedure that RATS has not yet compiled, it will automatically search the current directory and the directory containing the RATS executable file for a ".SRC" file with a name matching that of the procedure. You can use this field

to specify additional search directories. So, if you have procedures you use regularly, you may want to collect them in a particular directory or directories, and enter the directory name(s) in this field. See Section 15.2.1 of the *User's Guide* for details.

#### Procedure Library

The Procedure Library field allows you to specify a file of procedures that you want RATS to load automatically each time you start the program or clear the memory.

#### CATS Directory

If you have version 2.0 or later of our CATS cointegration analysis procedure, enter the directory containing your CATS files here. This will allow you to use the *CATS Cointegration* operation on the *Statistics* menu to launch the CATS procedure.

#### Batch Output .ext

This field lets you specify the extension that is appended to output files when running in batch mode.

### The Graphics Tab

The Graphics tab provides the following fields:

#### Fonts for Label Types

You can use this field to set the default font, size, and style for the various labels available on graphs generated by RATS. To change the settings for a label, select (highlight) a label type and then click on the "Change" button. This displays a standard font selection dialog box which you can use to select the font.

Note that you can override these defaults using the **GRPARM** instruction or the *Graph Settings* operation on the *Wizards* menu.

#### One Graph Window Only

If this is on, RATS re-uses the same window each time you generate a new graph. Otherwise, it displays each new graph in its own window.

### The Data Sources Tab

This section contains settings relevant to some common sources of economic data.

#### FRED Key

This is a security key that grants you access (via the *FRED Browser* operation from the *Data/Graphics* menu) to the online FRED® (Federal Reserve Economic Data) database provided by the St. Louis Federal Reserve Bank. RATS ships with a default key that should work. If the Fed ever changes the key, you can use this field to enter the new key. See **research.stlouisfed.org/fred2/** for more information on the database.

#### Haver Analytics Database

If you have files (such as the USECON database) from Haver in their DLX format, you can set this field to point to the directory where the files are located. This allows you

to use **OPEN HAVER** filename and **DATA(FORMAT=HAVER)** instructions to read the data, without having to provide a path on the **OPEN HAVER** instruction.

### CRSP Database

If you have CRSP (Center for Research in Security Prices) data on your system, you can set this field to the directory where those files are located. You can then read data from the CRSP database using **DATA(FORMAT=CRSP)**. See **www.crsp.com** for details.

### Global Insight (Citibase/DRI Basics) Directory

If you have a copy of the Global Insight (formerly DRI/Citibase) Basics Economics database, you can use the "Global Insight Directory" field to tell RATS where these files are located. You can then read data from those files into RATS using the FORMAT=DRI option on the **DATA** instruction.

## 2.4 Using RATS in Batch Mode

### About Batch Mode

In *batch mode*, RATS automatically reads instructions from a text file and stores the resulting output in another (new) file. This is very convenient for running long programs, because you can start RATS and then leave the computer unattended while it executes the program. You can create these programs using the RATS editor or any word-processing program that can save files as plain (unformatted) text. Similarly, you can view the output with RATS or any other word-processor.

### Running Programs in Batch Mode

There are three ways to run WinRATS in batch mode (please see the "ReadMe" file included with MacRATS and the UNIX/Linux installation guide for details on running in batch mode on those platforms).

One method is to start RATS, put it into batch mode using the *Batch Mode* operation on the *File* menu, and then drag and drop your program files from Windows Explorer or a similar utility onto the active RATS application. This is most useful if you have several programs that you want to run.

Another method is to run from a command prompt—either via the *Run* operation on the *Start* menu, or by opening the "Command Prompt" shell. You simply type the name of the RATS application itself, followed by the name of the program you want to execute and the command-line switch "`/RUN`".

Finally, you can execute RATS in batch mode from a short cut icon. This method is most useful if you have a particular program that you need to run frequently.

In any case, RATS will automatically run the specified program(s), and save the output in a new file (or files) with the name *filename.ext*, where *filename* is the name of the input file, and *ext* is the "batch mode extension" specified in the *File–Preferences...* operation in RATS (the default extension is `LIS`). The file(s) will be saved in the same directory as the program file(s).

Details on all three methods follow:

#### Drag and Drop Method

To run a program as a batch job using the drag and drop method, do the following:

1. Start RATS.
2. Switch RATS into batch mode by selecting the *Batch Mode* operation from the *File* menu.
3. Open Windows Explorer and arrange the screen so that you can see both the WinRATS window and the Explorer window on the screen at the same time.
4. Select one or more program files from Explorer, drag the file icon(s) over to the WinRATS window, and release the mouse button.

You can drag and drop the files one at a time, or you can drag and drop several files simultaneously. Note that a check mark appears next to the *Batch Mode* operation on the menu while RATS is in batch mode. To switch back to interactive mode, just select the *Batch Mode* operation again.

If you use batch mode frequently, you may want to create a copy of the WinRATS icon that will automatically start RATS in batch mode. Here's how:

1. Make a copy of the WinRATS program icon (see your Windows documentation or help for details).
2. Right-click on the RATS icon to display the properties dialog for the new icon.
3. Set the icon to "Run minimized" and click on OK.

Now, when you double click on this new icon, RATS will automatically start up with the *Batch Mode* switch turned on. You will need to click on the minimized icon to open it up into a window. You can then drag and drop program files onto the window.

## Command Prompt Method

If you are comfortable with using the Command Prompt shells in Windows, or are used to using the older DOS version in batch mode, you may find this method useful. You simply need to open the Command Prompt window from within Windows, and then type in the name of the RATS executable file (`ratsv9_std` for WinRATS, `ratsv9_pro` or `ratsv9_64` for 32-bit or 64-bit WinRATS Professional), the name of the input file, and the "`/RUN`" switch, separated by spaces. For example, the command line for WinRATS might look like:

```
c:\winrats\ratsv9_std c:\test\myprog.rpf /run
```

This will run the job and save the output to `MYPROG.LIS` (unless you have used *File–Preferences* in WinRATS to change the default output filename extension). You can include commands like this in a DOS batch (.bat) file if desired.

## Short Cut Icon Method

There are several ways to create shortcut icons—see your Windows on-line help or documentation for details. One method is to right-click on the desktop, and select *New–Shortcut* from the pop-up menu.

In the command line, enter the command to run the desired program, just as shown under "Command Prompt Method" above.

Click on the "Next" button, enter a name for the icon, and click on the "Finish" button. This will create the batch job icon on your desktop—you can double-click on this to run the job.

## Command Line Switches

RATS offers several command line "switches" to control batch mode operation—these can be used with both the "Command Prompt" and "Shortcut Icon" methods of running in batch mode:

| | |
|---|---|
| /NOSHOWGRAPHS | Suppresses display of graphs in batch mode. |
| /PRINTGRAPHS | Prints graphs as they are generated. |
| /PLOT=filename | Saves graphs to the specified plot file. Equivalent to putting an **OPEN PLOT** instruction in your program. |
| /DATA=filename | Opens the specified file as the **DATA** unit. Equivalent to an **OPEN DATA** instruction. |
| /COPY=filename | Opens the specified file as the **COPY** unit. Equivalent to an **OPEN COPY** instruction. |
| /PROC=filename | Designates *filename* as a Procedure Library file—the commands in this file will automatically be processed before the specified program file is executed. |

To use a particular switch, just add it to the end of the command line (you can use more than one switch if you wish). For example:

```
c:\winrats\ratsv9_pro myprog.rpf /run /noshowgraphs /plot=graf.rgf
```

executes the instructions in MYPROG.RPF, saves the graphs in the file MYPLOT.RGF, but does not display the graphs to the screen.

# 3. Data Formats

This section describes all of the data file formats supported by RATS, sorted by the name of the FORMAT option used with the file format.

In each section, we provide a brief description of the file format itself, a list of the RATS instructions that can read or write that format, and information on interface operations that can be used with the format. Where applicable, we provide additional details and a short example.

For each format, the "Interface Operations" section dscribes which of the following aspects of the RATS interface can be used with the format in question:

**Data Wizard**
: the Data Wizard operations on the *Data/Graphics* menu that you can use to read data series into memory.

**Series Window**
: a window showing all of the series in memory, which can also be used to read in and export data. Use the *View–Series Window* operation to display this window.

**RATS Data File Window**
: a window showing the contents of a RATS format data file.

**Report Window**
: a window displaying a RATS Report, generated either by a **REPORT** instruction, a WINDOW option on various instructions, or by reloading a Report Window using the *Restore Report* submenu on the *Window* menu.

**Matrix Window**
: a window displaying the contents of an array. Opened using an **MEDIT** instruction or by doing *View–All Symbols* and double-clicking on a matrix variable.

## 3.1 FORMAT=BINARY (Native binary format)

Binary is the internal format used on a computer for representing numbers. It is the fastest way of reading and writing data, but probably also the least useful, since it isn't easily portable from machine to machine or program to program and has no embedded description of the contents (10 columns with 100 rows and 1 column with 1000 look exactly the same). It is an Unlabeled format which can be used for input and output of series and matrices.

### RATS Instructions

| | |
|---|---|
| **data(format=binary)** | read series from binary into memory |
| **store(convert=binary)** | copy series from binary to RATS format |
| **copy(format=binary)** | write series to binary file |
| **prtdata(format=binary)** | write series from RATS format file to binary |
| **read(format=binary)** | read scalars and arrays from a binary file |
| **write(format=binary)** | write scalars and arrays to binary file |

ORG options are required on **DATA**, **STORE**, **COPY** and **PRTDATA**.

### Interface Operations

None

### Details

Binary format is the fastest way of reading and writing data, as it requires no translation from characters to numbers. It also preserves the internal representation of all numbers. Speed, however, is the only real advantage it has over other formats. Its disadvantages tend to outweigh this:

- Because the data file is not in character form, it is very difficult to determine what is on the file should you happen to forget.
- It is impossible to change any number on the file without rewriting the entire file.
- You cannot store dates or series labels on a binary file.

We don't recommend that you use BINARY format unless it's the only way available to communicate data to another program (running on the same type of computer). In reading binary data prepared using another program, you must be very careful to read the data back in exactly the same way as it was written.

- Real-valued data use a 64-bit representation. This is double precision, except on machines with 64-bit single precision.
- With ORG=ROWS, RATS reads (or writes) each *series* with a single call.
- With ORG=COLS, each *observation* is read (written) with a single call.

## 3.2 FORMAT=CDF (Comma delimited text formats)

This is a text file with fields delimited with commas. It's a Labeled Table or Unlabeled format (depending upon the file) for both input and output of series, reports and matrices. For reading series, we recommend that you use the more general FORMAT=PRN instead.

### RATS Instructions

| | |
|---|---|
| **data(format=cdf)** | read series from CDF into memory |
| **store(convert=cdf)** | copy series from CDF to RATS format |
| **copy(format=cdf)** | write series to CDF file |
| **prtdata(format=cdf)** | write series from RATS format file to CDF |
| **read(format=cdf)** | read scalars or arrays from CDF into memory |
| **write(format=cdf)** | write scalars or arrays from memory to CDF file |

ORG options are required on **DATA**, **STORE**, **COPY** and **PRTDATA**.

### Interface Operations

#### Data Wizard

You can read series using *Data (Other Formats)* on the *Data/Graphics* menu.

#### Series Window and RATS Data File Window

You can import series using *File–Import...* or the Import toolbar button. You can export series using *File–Export...*, the Export toolbar button, or the *Export* operation on the contextual menu.

### Details

On **DATA** and **STORE**, you can, if required, use the NOLABELS option to skip automatic label processing and use the **LEFT** and **TOP** options to isolate the data.

See Section 2.8 of the *Introduction*
for more details on the format.

## 3.3 FORMAT=CITIBASE/FORMAT=DRI

This is a Time Series Database format available for series input, available only on the Professional level of RATS. It is the format used for the Basic Economics database (formerly known as Citibase), which is maintained by Global Insight.

### RATS Instructions

**data(format=citibase)** read from Basic Economics into memory

**store(convert=citibase)** copy series from Basic Econ. to RATS format

### Interface Operations

None for reading or writing data. *File–Preferences* sets location of files.

### Details

You must use the *Preferences* operation on the *File* menu to tell RATS where the data are located. With that done, you can read data with

**data(format=citibase,**other options**)** *start end list of series*

Because the series have dates, you can use this to select a sub-sample or to compact or expand to a different frequency.

See Section 2.5 of the *Introduction* for details.

## 3.4 FORMAT=CRSP (CRSP Files)

This is available in the Pro version, but only under Windows and certain UNIX platforms. You need a subscription to CRSP® (Center for Research in Security Prices) data, available from the University of Chicago's Booth School of Business (**www.crsp.com**). It is a Time Series Database format available for series input only.

### RATS Instructions

**data(format=crsp)** read series from CRSP into memory

### Interface Operations

None for reading or writing data.

You can use *File–Preferences* to set the default location of the CRSP files.

### Details

Your workstation must be configured with the proper set of environment variables (CRSP_ROOT=, etc.) and you need to tell RATS where the data are located. You can do that by setting the CRSP Directory setting in the *File–Preferences...* dialog box, or by using an explicit **OPEN CRSP** instruction specifying the path in your program.

You can fetch the price, return, volume, bid and ask prices for stocks. These are done by prefixing the CRSP Permno for the stock with "P" for price, "R" for return, "V" for volume, "B" for bid and "A" for ask. You can ask for any data frequency from daily to annual, and you'll get the aggregation for that frequency produced by CRSP.

For example, the following will get the monthly returns for IBM (permno 12490) and Microsoft (10107) over 1984:4 to 2004:12. Since Microsoft doesn't start trading until 1986, its returns will be NA's until then.

```
open crsp c:\CRSP_Sample_Data\data\aiz\diz\
cal(m) 1984:4
all 2004:12
data(format=crsp) / r12490 r10107
```

## 3.5 FORMAT=DBF (dBase Files)

This is a database format used by dBase and compatible programs. It's a Labeled Table format for both input and output of series.

### RATS Instructions

| | |
|---|---|
| **data(format=dbf)** | read series from DBF into memory |
| **store(convert=dbf)** | copy series from DBF to RATS format |
| **copy(format=dbf)** | write series to DBF file |
| **prtdata(format=dbf)** | write series from RATS format file to DBF |

### Interface Operations

#### Data Wizard

You can read series using *Data (Other Formats)* wizard on the *Data/Graphics* menu.

#### Series Window and RATS Data File Window

You can import series using *File–Import...* or the Import toolbar button. You can export series using *File–Export...*, the Export toolbar button, or the *Export* operation on the contextual menu.

### Details

The data are read from the file based upon the field (column) names. Each record is treated as a separate observation. If the file includes a date field as the first field, RATS will be able to make use of the date information to select a sub-sample, recognize skipped observations, or allow compacting or expanding to a different frequency (Section 2.5 of the *Introduction*). The dates can
either use the DBF date field format, which codes dates as year/month/day, or be entered as a character field with the standard codings like "year:period".

For writing data, each series that you write will be a field in the database. The dBase specification allows for at most ten characters in a field name, so longer names will be truncated. If you ask for dates, RATS will either include a date field (for daily or weekly data), or a character field with coded dates.

### Example

This reads in several series of stock return data from a DBF file:

```
calendar(d) 1996:1:2
allocate 2009:12:31
open data stockdat.dbf
data(format=dbf)
```

## 3.6 FORMAT=DIF (Data Interchange Format)

DIF is a (rather bulky) text format used for transmitting the content of a simple spreadsheet. It's a Labeled Table or Unlabeled format (depending upon the file) for both input and output of series, reports and matrices, available in all versions of RATS.

### RATS Instructions

| | |
|---|---|
| **data(format=dif)** | read series from DIF into memory |
| **store(convert=dif)** | copy series from DIF to RATS format |
| **copy(format=dif)** | write series to DIF file |
| **prtdata(format=dif)** | write series from RATS format file to DIF |
| **read(format=dif)** | read scalars or arrays from DIF into memory |
| **report(format=dif)** | (with ACTION=FORMAT) write a report to DIF file |

ORG options are required on **DATA**, **STORE**, **COPY** and **PRTDATA**.

### Interface Operations

#### Data Wizard

You can read series using *Data (Other Formats)* on the *Data/Graphics* menu.

#### Series Window and RATS Data File Window

You can import series using *File–Import...* or the Import toolbar button. You can export series using *File–Export...*, the Export toolbar button, or the *Export* operation on the contextual menu.

#### Report Windows

You can write reports to a DIF file using *File–Save As*, *File–Export*, *Edit–Copy*; the Export and Copy toolbar items, and the *Export* and *Copy* contextual menu operations.

#### Matrix Windows

You can write the contents of a matrix window to a DIF file using *File-Save As*, *File-Export*; the Save and Export toolbars, and the *Export* contextual menu.

### Details

For copy-and-pasting into DIF, you will probably have to choose the *Paste Special* operation in the target application. RATS copies the full precision of real numbers when copying to the clipboard in DIF format.

## 3.7 FORMAT=DTA (Stata Native Format)

DTA is the native format for Stata™ data files. It's a Labeled Table format available for series input (only), on all RATS versions.

### RATS Instructions

| | |
|---|---|
| **data(format=dta)** | read series from Stata into memory |
| **store(convert=dta)** | copy series from Stata into RATS format |

### Interface Operations

#### Data Wizard

You can read series using *Data (Other Formats)* on the *Data/Graphics* menu.

#### Series Window and RATS Data File Window

You can import series using *File–Import...* or the Import toolbar button.

### Details

Data in a DTA file are stored as named series which will be read or imported using those names. You can read all the series by leaving the series list blank on either **DATA** or **STORE**, or you can request specific series by listing just the ones you want. If you import directly into a data list, you will get all series.

RATS will not recognize any dates on the file.

### Example

```
open data pennxrate.dta
calendar(panelobs=34,a) 1970
data(format=dta) 1//1970:01 151//2003:01 year xrate ppp $
   id capt realxrate lnrxrate oecd g7
```

reads 24 years of panel data for 151 countries.

## 3.8 FORMAT=FAME (FAME Databases)

FAME is a time-series database management program provided by SunGard (see *www.sungard.com/fame*). It is a Time Series Database format for input and output, and is supported by the Professional versions of WinRATS and (on certain platforms) UNIX. In addition to needing the proper version of RATS, you also need a license for the FAME software on the same platform.

### RATS Instructions

| | |
|---|---|
| **data(format=fame)** | read series from FAME into memory |
| **store(convert=fame)** | copy series from FAME to RATS format |
| **copy(format=fame)** | write series to FAME file |

### Interface Operations

On the *Data/Graphics* menu (for Pro versions), there's a *Data Browser–Fame Database* operation. This opens a Series Window showing the contents of the file. You can drag and drop series from that into the Series Window or a RATS Data File window.

### Details

FAME databases can be either local (accessed as standard operating system files) or server. There are slight differences in how you use them.

Local databases are files with a `.DB` extension that you read like any other file:

```
cal(m) 1970:1
open data c:\fame\util\driecon.db
data(format=fame) 1970:1 2003:7 ipsb50001 cusa0
```

For server access, just use the base name of the database on the **OPEN DATA**, and include the additional option `SERVER=`*`server spec`* on the **DATA** or **STORE** instruction. The *`server spec`* is a string with the port, followed by the server node, followed by (if necessary) the user id and password. These are separated from each other by the `|` character. For instance:

```
cal(q) 1980:1
open data driecon
data(format=fame,server="5555|booker") 1980:1 2010:2 gdp gicv
```

You can only add series to a *local* database. Use the `APPEND` option on the **OPEN COPY** instruction to add to an existing file (rather than create a new one):

```
open(append) copy write.db
copy(format=fame) /  x y z
```

## Fame Objects Supported by RATS

RATS can read FAME objects with combinations of SERIES/FORMULA class, NUMERIC/BOOLEAN/PRECISION type, and all frequencies except UNDEFINED. All series created by RATS are written into FAME files as objects with SERIES class and PRECISION type. Missing values, which appear as NC, ND and NA in FAME data bases are translated into %NA in RATS. Missing values in RATS are converted to NA when exporting to FAME data bases.

## Symbolic Name Conversion Between RATS and FAME

RATS and FAME place different restrictions on symbolic variable names, so RATS will not accept some variable names which are legal in FAME. To deal with this problem, RATS allows you to create a "lookup table"—a file containing a RATS name and an associated FAME name for one or more series.

The lookup table file must be an ASCII text file containing two columns of text separated by at least one space. The first column on each line is a RATS series name, while the second column is the associated FAME name.

## Using Lookup Tables

To use a lookup table, you must open the lookup file with an **OPEN FAME** instruction, before executing your **DATA** or **COPY** instruction (FAME is a reserved I/O unit name).

On a **DATA** instruction, you must list only legal RATS series names. Normally, RATS searches the FAME data base for the specified series. If you have opened a lookup table, however, RATS first looks for the series names in the lookup table. For names that *do* appear in the table, RATS searches the FAME data base using the associated FAME name, rather than the RATS name. The series, if found, is then read into memory using the RATS name. For series not listed in the table, RATS searches the FAME data base using the RATS name.

Similarly, when writing data using **COPY**, any series that appear in a lookup table will be written using the "FAME" name, rather than the RATS name.

A lookup table should be a text file with two columns. The RATS name should be the first item on each line, followed by the FAME name (separated by at least one space). If the second column is omitted, the FAME name will be the same as the RATS name (equivalent to not including the name in the file). Anything after the second column is ignored, so you can add comments after the FAME name if you like. The order of series listed in a lookup table file is irrelevant. Both RATS and FAME are case-insensitive to series names, so mixed cases are legal in the lookup table. All lower case characters are converted to upper case.

## 3.9 FORMAT=FORTRAN string

This is an Unlabeled text format for input and output which uses FORTRAN I/O descriptors. It's available on all versions of RATS. It's available in case other methods for dealing with a poorly-formatted text file are inconvenient. However, we would recommend you look first at importing the file into a spreadsheet program and using its line parsing tools.

### RATS Instructions

| | |
|---|---|
| **data(format=**'Fortran format string'**)** | read series using FORTRAN format |
| **copy(format=**'Fortran format string'**)** | write series using FORTRAN format |
| **read(format=**'Fortran format string'**)** | read data using FORTRAN format |
| **write(format=**'Fortran format string'**)** | write data using FORTRAN format |

ORG options are required on **DATA** and **COPY**.

### Interface Operations

None

### Details

FORTRAN format allows you to use the standard FORTRAN I/O formatting codes to indicate the format of the data file. You probably shouldn't try using it unless you are fairly comfortable with FORTRAN formats (described briefly below), *and* the other methods above are cumbersome.

With FORTRAN formats you can tell RATS exactly how the data are organized when using **DATA** or **READ**, or exactly how you want the data to be formatted when using **COPY** or **WRITE**. For example, when used with **COPY**, FORMAT=FREE uses a very wide field for each number so that it can reasonably handle very different magnitudes. The result is that you can sometimes get a lot of extra digits. This takes up space (so you see fewer numbers at a glance) and masses of digits can be hard to read.

If you are unfamiliar with FORTRAN, a brief description is in order. The basic format elements for real numbers take the forms Fw.d, Ew.d and Gw.d. RATS (which only simulates FORTRAN I/O) also allows you to use the Iw format for input.

| | |
|---|---|
| Fw.d | For numbers with a fixed decimal place: a total of **w** positions (total digits to the left and the right of the decimal) with **d** digits to the right of the decimal. Very large and very small numbers will not fit an F format. For example, the number 1000000. cannot be printed as F15.8. |
| Ew.d | For numbers in scientific notation. When writing files, this format is useful because it can handle a number of any magnitude, but it can be difficult to pick out large and small values at a |

| | |
|---|---|
| | single glance. For example, it takes a bit of work to see that the first of 1.343E-02 and 8.547E-03 is the larger. |
| Gw.d | This is a mixture of the F and E formats. If a number can reasonably be printed with the F format, that is used. All others (very large and very small) are displayed in the E format. |
| Iw | For integer numbers. If there is a fractional part to the number, it is ignored. |
| Aw | This is used (by **COPY**) only for the date strings. It prints a character string, left justified, in a field of width w. |
| wX | Indicates w spaces |
| / | Indicates a skip to the next line |

In the description of a line, the fields are separated by commas. You can prefix a field descriptor by a number of repetitions. For instance, F6.2,F6.2,F6.2 can be shortened to 3F6.2.

## The FORMAT Option

The **DATA**, **COPY**, **READ**, and **WRITE** instructions all support the FORTRAN format. The FORMAT option is the same for all four instructions:

```
FORMAT="( format string )"
```

Enclose the format string in quotes, for instance, FORMAT="(11X,4F15.7)". The format string must fit on a single line. If you need an extremely long string, you may want to put the string on its own line:

```
data(org=col,format=  $
 "(f8.5,2x,f8.4,2x,f8.5,2x,f8.3,2x,f8.5,2x,f8.5)"  )  $
 1947:1 1979:4  wage interest stocks mortgage charitable misc
```

## With ORG=ROWS

Each entire series is read with the indicated format: below is a RATS instruction and its FORTRAN language equivalent:

```
data(format="(11x,4f15.7)")  1 100 gnp m1

       read(n,1000)  (gnp(i),i=1,100)
       read(n,1000)  (m1(i) ,i=1,100)
  1000 format(11x,4f15.7)
```

If you have blank lines separating the series, you may have to read the first series with one format, then use a format such as (//(11X,4F15.7)) for the remaining series: this particular format will skip two lines before reading a series.

## With ORG=COLS

The equivalent of a separate FORTRAN READ statement is used for each observation: on the next page is a sample RATS instruction and its FORTRAN equivalent. (You must delete any header lines at the top of the file).

```
data(org=col,format="(f9.4,2x,f10.3,2x,f9.4)")  1 100 gnp m1 ipd
```

```
        do 100 i=1,100
           read(n,1000) gnp(i),m1(i),ipd(i)
 100 continue
1000 format(f9.4,2x,f10.3,2x,f9.4)
```

## Mixed Character and Numeric Data

FORTRAN format codes can also be useful if you have a text file containing both numeric data and text labels. You cannot store character information in SERIES variables, or use **DATA** to read in character/label information, but if the data are well–organized, it may be possible to read the numeric data into series and the text data into LABEL variables. You will probably need to read the file twice: once to get the numeric data and once to get the character data.

Suppose you have the following data file:

```
1.0450  ME     2.0   1.0
2.3210  MS     2.0   2.0
1.8930  MN     2.0   3.0
```

You could use the following program to read this data. First, a **READ** instruction using a FORMAT option skips the numeric data and reads only the two-character label. The data file is rewound (positioned to the top), and then the data are read using a **DATA** instruction which skips over the character information to get the numbers. Finally, a simple loop is used to display formatted output:

```
all 3
declare vect[labels] states(3)
open data mixed.dat
read(unit=data,format="(8x,a2)") states
rewind data
data(format="(f6.4,6x,2f6.1)",org=col) / x y z
do row=1,3
   display states(row) @10 x(row) y(row) z(row)
end
```

## Missing Data

You can use the codes (such as NA) for missing values used by other text files. You can also leave a blank area where the missing values should be and use BLANK=MISSING on the **DATA** instruction. This will interpret the blank area as a missing value (as opposed to a standard practice of reading a zero).

## Using FORTRAN formats: An Example

The following data set was used in Section 2.9 of the *Introduction*. We show here how to use FORTRAN format to simplify the process of reading the data.

```
FIRM A
1996          11.3      11.6      10.9      12.3
1997          13.0      12.8      11.5      12.5
1998          12.9      13.0      13.2      13.6

FIRM B
1996          22.5      21.9      24.3      25.6
1997          21.9      21.8      22.6      23.5
1998          22.5      25.0      24.5      25.4
```

The data rows can be read with `(12X,4F9.1)`. The problem is the other rows: there is only one row preceding the first series (the row with "`FIRM A`"), but two (the blank plus "`FIRM B`") preceding the second. The simplest way to handle this is to add a blank line at the beginning of the file and use

```
cal(q) 1996:1
all 1998:4
open data test.dat
data(format="(//(12x,4f9.1))",org=rows) / firma firmb
print / firma firmb
```

Note that we use **PRINT** to verify that the data has been read properly. How did we come up with '`(//(12X,4F9.1))`'? First, the two slashes tell RATS to skip two lines before each block of data. Next, for a data block where the numbers are regularly spaced (as they are here), just determine the position of the last digit in the *first* data value on the line (column 21 in this case) and the field width of the data (here 9, which is the distance from the end of one number to the end of the next). The 12 in `12X` is just the number of leading positions (21–9=12) to be skipped.

The `4F9.1` indicates that the fields are each nine characters wide (including leading blanks) with 1 digit after the decimal point, and that this format is repeated 4 times per row. After skipping two lines, RATS will read data into `FIRMA` using the `12X,4F9.1` format until it has read the requested number of data points (12 in this case). It then skips two more lines and again uses the `12X,4F9.1` format to read 12 data points into `FIRMB`.

An alternative if you either can't, or don't want to, alter the original file is to use separate instructions so that you can apply different formats in succession:

```
data(format="(/(12x,4f9.1))",org=rows) / firma
data(format="(//(12x,4f9.1))",org=rows) / firmb
```

## 3.10 FORMAT=FRED (Online FRED™ database)

This is the St. Louis Federal Reserve's online FRED database. Available only in the Professional version of RATS.

### RATS Instructions

| | |
|---|---|
| **data(format=fred)** | read series from FRED into memory |
| **store(convert=fred)** | copy series from FRED to RATS format |

### Interface Operations

#### Data Wizard

You can view, graph, and read in series using the FRED browser window, via the *FRED(online)* operation under *Data Browsers* on the *Data/Graphics* menu.

### Details

See Section 2.2 in the *Introduction* for some general information, and see the "FRED Data Browser" topic in the Help for details on using the browser.

See the web site:

research.stlouisfed.org/fred2/

for more details on the database itself.

## 3.11 FORMAT=FREE (Delimited text format, no names/dates)

This is a text file with fields delimited with spaces, tabs or commas that does *not* contain series labels or date information (see FORMAT=PRN on page AT-52 for text files that *do* include series names and, optionally, dates). It's an Unlabeled format for both input and output of series, reports and matrices, available on all versions of RATS.

### RATS Instructions

| | |
|---|---|
| **data(format=free)** | read series from FREE into memory |
| **store(convert=free)** | copy series from FREE to RATS format |
| **copy(format=free)** | write series to FREE file |
| **prtdata(format=free)** | write series from RATS format file to FREE |
| **read(format=free)** | read scalars or arrays from FREE into memory |
| **write(format=free)** | write scalars or arrays from memory to FREE file |

### Interface Operations

#### Data Wizard

You can read series using *Data (Other Formats)* on the *Data/Graphics* menu.

#### Series Window and RATS Data File Window

You can import series using *File–Import...* or the Import toolbar button. You can export series using *File–Export...*, the Export toolbar button, or the *Export* operation on the contextual menu.

### Details

See Section 2.9 of the *Introduction* for details.

## 3.12 FORMAT=HAVER (Haver Analytics native format)

This is a Time Series Database format for accessing data in the Haver Analytics native DLX format. It is available only in the Pro level, and only on Windows. It is for series input. The Haver Analytics databases, which are available for purchase from Estima, are provided in both RATS and DLX formats.

### RATS Instructions

| | |
|---|---|
| **data(format=haver)** | read series from Haver into memory |
| **store(convert=haver)** | copy series from Haver to RATS format |

### Interface Operations

On the *Data/Graphics* menu (for WinRATS Pro), there's a *Data Browser–Haver DLX Database* operation. This opens a window listing the contents of the file. You can drag and drop series from that into the Series Window or a RATS Data File window.

### Details

A DLX database consists of several files in the same directory with a common base name, For instance, the USECON data base (a 12,000 series database with a broad range of key U.S. macroeconomic series) comprises the files USECON.IDX and USECON.DAT. To use **DATA** or **STORE**, you need to open a unit named HAVER with just that base name (in this case USECON), with the full path if required. In the *File–Preference* dialog, in the Data Sources tab, you can set a permanent association of the HAVER unit with a particular database, If you've done that, you don't need to include an **OPEN HAVER** instruction to access that database.

Series are read according to their database names. Each series has its own range and date scheme, so you can choose any subset of entries and can do compaction and expansion as described in Section 2.5 of the *Introduction*.

### Example

The following reads six series (GDP, GDP deflator, gross private domestic investment, unemployment rate, one year T-bill rate and M2) from the USECON data base. The last three series are compacted from the monthly data on the database to quarterly in the workspace.

```
open haver m:\data\haver\usecon
cal(q) 1960:1
data(format=haver) 1960:1 2010:2 gdpq dgdp fq lr fcm1 fm2
```

## 3.13 FORMAT=HTML (HTML table)

HTML is a table-oriented text output (only) which renders the contents as an HTML table for use in a web page. It is available on all versions of RATS. You can apply it to data series and reports.

### RATS Instructions

| | |
|---|---|
| **copy(format=html)** | copy series from memory to an HTML table |
| **prtdata(format=html)** | copy series from a RATS format file to an HTML table |
| **write(format=html)** | write scalars or arrays from memory to HTML file |
| **report(format=html)** | used with the ACTION=SHOW and UNIT options, this writes the report to an HTML table |

### Interface Operations

#### Report Windows

You can write reports to HTML using *File–Save As*, *File–Export*, *Edit–Copy*; the Export and Copy toolbar items, and the *Export* and *Copy* contextual menu operations.

#### Series Window and RATS Data File Window

You can export series using *File–Export...*, the Export toolbar button, or the *Export* operation on the contextual menu.

#### Matrix Windows

You can write the contents of a matrix window to an HTML file using *File-Save As*, *File-Export*; the Save and Export toolbars, and the *Export* contextual menu.

### Details

This produces just a part of a full HTML file, so you will have to insert it into a larger document. From a report window, it copies real numbers as they are displayed on the screen, so format them first to the precision that you want.

## 3.14 FORMAT=MATLAB (MATLAB binary format)

MATLAB is the binary format used by the MATLAB™ software. This can be used in various ways for input of series or matrices on any version of RATS. Depending upon the structure of the file, it can be either a Time Series Database format or an Unlabeled format.

### RATS Instructions

| | |
|---|---|
| **data(format=matlab)** | read series from MATLAB into memory |
| **store(convert=matlab)** | copy series from MATLAB to RATS format |
| **read(format=matlab)** | read scalars or arrays from MATLAB into memory |

### Interface Operations

#### Data Wizard

You can read series from a MATLAB (*.MAT) file using *Data (Other Formats)* on the *Data/Graphics* menu.

#### Series Window and RATS Data File Window

You can import series using *File–Import...* or the Import toolbar button.

### Details

**read(format=matlab)**

Since **READ** is designed to read matrices, reading MATLAB data is straightforward: if there's an $N \times M$ matrix A on the file, **READ A** will dimension A as $N \times M$ and read the data into it. You can read multiple matrices off a single file. If you've saved scalars on the file, you can read those as well.

```
data(format=matlab)
store(convert=matlab)
```

Data series will typically be stored in one of two ways: as separately named column vectors, or as a single matrix with multiple columns. Either way, RATS will not recognize any dates on the file. If the data are in separate columns, read them by name. For the arrangement with multiple columns, use the option SHEET=*matrix name*, then give series names to the columns.

### Examples

The file CBES.MAT has one matrix named PSID. This has 3051 rows by 3 columns. The columns represent (in order) age, education and income. The following will read the data from that file and create three series:

```
open data cbes.bat
data(format=matlab,sheet="psid") 1 3051 age education income
```

The file USMODEL_DATA.MAT has seven column matrices with names DC, DINVE, DY, LABOBS, DW, PINFOBS and ROBS. Each column has 230 entries, representing quarterly data from 1950Q1 to 2007Q2. The following reads data from the file:

```
open data usmodel_data.mat
calendar(q) 1950:1
data(format=matlab) 1950:01 2007:02 dc dinve dy labobs dw $
   pinfobs robs
```

## 3.15 FORMAT=ODBC (ODBC/SQL databases)

This is a Labeled Table format for series input (only), available in the Professional level of RATS, to read data from any database format that supports Open Database Connectivity (ODBC) and SQL.

### RATS Instructions

| | |
|---|---|
| **data(format=odbc)** | read series from a database into memory |
| **store(format=odbc)** | copies series from a database to a RATS format file |

### Interface Operations

None

### Details

Before reading data from a database using ODBC and SQL, you must set up an ODBC "Data Source" for the desired database. The process for doing this varies somewhat depending on the operating system in use. With recent versions of Windows, you can set up a Data Source by opening the Windows Control Panel and double-clicking on the "Data Sources (ODBC)" control. This displays a dialog box you can use to define a "Data Source Name" for your database.

Once you have the Data Source defined, you can open a connection to the database in RATS by using a command of the form:

**open odbc** *dsn*

where "*dsn*" is the Data Source Name for the database.

You can then read data from the database by executing a **DATA** command. On the **DATA** instruction, you can either:

- use the SQL option to provide a short (255 characters or less) SQL string directly on the **DATA** instruction, or
- use the QUERY option to process a more complex SQL queries, either from a separate file or from the lines following the **DATA** instruction.

The **DATA** command creates a table (internally) which is then processed similar to the spreadsheet files in Section 2.8 of the *Introduction*.

For example, the following creates a table with date and total daily sales (summing the SUBTOTAL field by date to create the SALES field), then reads the data into RATS, creating the series SALES as monthly sums.

```
cal(m) 1995:1
open odbc "Sales"
data(format=odbc,compact=sum,
sql="select date,sum(subtot) as sales from invoice order by date") $
   1995:1 2006:12
```

The code below does the same thing using QUERY:

```
cal(m) 1995:1
open odbc "Sales"
open sqlfile "c:\rats\sqlquery.txt"
data(format=odbc,compact=sum,query=sqlfile) 1995:1 2006:12
```

where the file SQLQUERY.TXT contains the following lines:

```
select date,sum(subtot)
as sales from invoice
order by date
;
```

## 3.16 FORMAT=PORTABLE (RATS Portable Format)

Portable format is the text version of the RATS file format. It includes all of the information stored on the RATS file, but in text form. The most important use of is to create archival human–readable copies of RATS format files. It's a Time Series Database format which can be used for input and output in all versions of RATS.

### RATS Instructions

| | |
|---|---|
| **data(format=portable)** | read series from PORTABLE into memory |
| **store(convert=portable)** | copy series from PORTABLE to RATS format |
| **copy(format=portable)** | write series to PORTABLE file |
| **prtdata(format=portable)** | write series from RATS format file to PORTABLE |

### Interface Operations

#### Data Wizard

You can read series using *Data (Other Formats)* on the *Data/Graphics* menu.

#### Series Window and RATS Data File Window

You can import series using *File–Import...* or the Import toolbar button. You can export series using *File–Export...*, the Export toolbar button, or the *Export* operation on the contextual menu.

### Examples

This writes a series out to a file in portable format, including one header (description) line). You can add up to two of those.

```
copy(format=portable,header=1)  / gnp
     GROSS NATIONAL PRODUCT, CURRENT DOLLARS
```

This shows how a portable file is constructed. The series are displayed separately. The key elements for each are the series name (on a separate line), the line containing the frequency and starting and ending dates, the header lines, and the row of === which ends the header.

```
    GNP
     Quarterly data from 1947:01 to 1990:04
     GROSS NATIONAL PRODUCT, CURRENT DOLLARS
================================================================================
1947:01       224.9000000     229.1000000     233.3000000     243.6000000
1948:01       249.6000000     257.1000000     264.0000000     265.5000000
 etc.
```

```
dedit mydata.rat
open copy archive.txt
prtdata(unit=copy)
```

This writes all of the series stored in the RATS data file `MYDATA.RAT` to a text file called `ARCHIVE.TXT`.

## 3.17 FORMAT=PRN (Delimited text formats)

This is a text file with fields delimited with spaces, tabs or commas. It's a Labeled Table or Unlabeled format (depending upon the file) for both input and output of series, reports and matrices, available on all versions of RATS.

### RATS Instructions

| | |
|---|---|
| **data(format=prn)** | read series from PRN into memory |
| **store(convert=prn)** | copy series from PRN to RATS format |
| **copy(format=prn)** | write series to PRN file |
| **prtdata(format=prn)** | write series from RATS format file to PRN |
| **read(format=prn)** | read scalars or arrays from PRN into memory |
| **write(format=prn)** | write scalars or arrays from memory to PRN file |

### Interface Operations

#### Data Wizard

You can read series using *Data (Other Formats)* on the *Data/Graphics* menu.

#### Series Window and RATS Data File Window

You can import series using *File–Import...* or the Import toolbar button. You can export series using *File–Export...*, the Export toolbar button, or the *Export* operation on the contextual menu.

### Details

On **DATA** and **STORE**, you can, if required, use the NOLABELS option to skip automatic label processing and use the **LEFT** and **TOP** options to isolate the data.

See Section 2.8 of the *Introduction* for more details on the format.

## 3.18 FORMAT=RATS (RATS data format)

RATS format is specially designed to deal with time series data. It's a Time Series Database format which is available on all versions of RATS. It supports all of the time-series frequencies supported by RATS, including panel and intra-day data. It also allows you to store series with different frequencies in the same file.

### RATS Instructions

| | |
|---|---|
| **data(format=rats)** | read series from RATS format into memory |
| **store(convert=rats)** | copy series from one RATS format file to another |
| **copy(format=rats)** | write series to RATS file |
| **prtdata(format=rats)** | write series from one RATS format file to another |

### Interface Operations

#### Data Wizard

You can read series by using *File–Open...* to open a RATS format file, selecting the desired series from the RATS Data File window, and using *Data (RATS Formats)* on the *Data/Graphics* menu.

#### Series Window and RATS Data Window

You can use *File–Open* and *File–New...* to open or create a RATS format file. Various toolbar and contextual menu operation are available for generating graphs, editing series, renaming series, and so on. You can bring series into memory from an open RATS file using the Data Wizard described above, or by dragging and dropping series onto the Series Window.

You can write series from memory to the open file by dragging and dropping them from the Series Window onto the RATS Data File Window.

You can import series into the file from another using *File–Import...* or the Import toolbar button. You can export series to another using *File–Export...*, the Export toolbar button, or the *Export* operation on the contextual menu.

### Details

See Section 2.7 of the *Introduction* for more details on creating and using RATS format files.

## 3.19 FORMAT=TEX (TeX table)

TeX is a table-oriented text format for output (only) which renders the contents as a TeX `tabular` environment. You can apply it to data series and reports. It's available on all versions of RATS.

### RATS Instructions

| | |
|---|---|
| **copy(format=tex)** | writes series to a TeX file |
| **prtdata(format=tex)** | copy series from a RATS file to a TeX file |
| **write(format=tex)** | write scalars or arrays from memory to TeX file |
| **report(format=tex)** | used with the `ACTION=SHOW` and `UNIT` options, this writes the report to a TeX file |

### Interface Operations

#### Series Window and RATS Data File Window

You can export series using *File–Export...*, the Export toolbar button, or the *Export* operation on the contextual menu.

#### Report Windows

You can write reports to a TeX file using *File–Save As*, *File–Export*, *Edit–Copy as TeX*, the Export and Copy toolbar items, and the *Export* and *Edit–Copy as TeX* contextual menu operations.

#### Matrix Windows

You can write the contents of a matrix window to a TeX file using *File-Save As*, *File-Export*, *Edit–Copy as TeX*, the Save and Export toolbars, and the *Export* contextual menu.

### Details

To use the TeX table in a document, you need to include the `dcolumn` package, which allows for decimal alignment of columns. This is an example of what is produced:

```
\newcolumntype{.}{D{.}{.}{-1}}
\begin{tabular}{. . . }
\multicolumn{1}{c}{Test} & \multicolumn{1}{c}{Statistic} &
  \multicolumn{1}{c}{P-Value}\\
\multicolumn{1}{l}{Joint} & \multicolumn{1}{r}{4.070} &
  \multicolumn{1}{r}{0.000}\\
\multicolumn{1}{l}{Variance} & \multicolumn{1}{r}{0.121} &
  \multicolumn{1}{r}{0.473}\\
\multicolumn{1}{l}{Constant} & \multicolumn{1}{r}{0.922} &
  \multicolumn{1}{r}{0.004}\\
\multicolumn{1}{l}{X2} & \multicolumn{1}{r}{0.917} &
  \multicolumn{1}{r}{0.004}\\
\multicolumn{1}{l}{X3} & \multicolumn{1}{r}{0.914} &
  \multicolumn{1}{r}{0.004}\\
\end{tabular}
```

Note that if you want to paste the table into a TeX document, you need to use one of the *Copy as TeX* operations. The standard *Copy* operations also produce a straight tab-delimited text copy of the data, which will generally be the one chosen by the Paste operation in the target application. *Copy as TeX* puts only the TeX version into the clipboard.

You will generally enclose the tabular environment produced by RATS inside a table environment which will handle the placement and captioning. If the TeX code on the previous page is saved in the file `testtable.tex`, the following would center it on the page with the caption "Example of TeX Output".

```
\begin{table}[htb]
\centering
\include{testtable}
\caption{Example of TeX Output}
\end{table}
```

## TeX formatting codes

In general, the export procedure will escape any character which would be interpreted as a TeX format code. For instance, the lag notation `Y{1}`, `Y{2}` and `Y{3}` out of a standard regression output will appear as they would in RATS output, as the `{` and `}` get escaped to prevent the standard TeX interpretation.

```
\newcolumntype{.}{D{.}{.}{-1}}
\begin{tabular}{. . . }
\multicolumn{2}{l}{Constant} & \multicolumn{1}{r}{0.0976}\\
\multicolumn{2}{l}{Y\{1\}} & \multicolumn{1}{r}{0.1356}\\
\multicolumn{2}{l}{Y\{2\}} & \multicolumn{1}{r}{0.1769}\\
\multicolumn{2}{l}{Y\{3\}} & \multicolumn{1}{r}{0.1457}\\
\end{tabular}
```

If you want to include special formatting codes, you can edit the information after you've included in your final document. If you want to eliminate that step, you can enclose a string (inside the quotes) with $ characters when you built a `REPORT`. For instance, the following will use $\phi_1$, $\phi_2$ and $\phi_3$ for the three lag coefficients.

```
report(action=define)
report(atrow=1,atcol=1) "1" %beta(1) %stderrs(1)
report(atrow=2,atcol=1) "$\phi _1$" %beta(2) %stderrs(2)
report(atrow=3,atcol=1) "$\phi _2$" %beta(3) %stderrs(3)
report(atrow=4,atcol=1) "$\phi _3$" %beta(4) %stderrs(4)
report(action=format,picture="*.####")
open texfile "regoutput.tex"
report(action=show,format=tex,unit=texfile)
```

## 3.20 FORMAT=TSD (Tab separated text formats)

This is a text file with fields delimited with tabs. It's a Labeled Table or Unlabeled format (depending upon the file) for both input and output of series, reports and matrices, available in all versions of RATS. For reading series, we recommend that you use the more general FORMAT=PRN instead.

### RATS Instructions

| | |
|---|---|
| **data(format=tsd)** | read series from TSD into memory |
| **store(convert=tsd)** | copy series from TSD to RATS format |
| **copy(format=tsd)** | write series to TSD file |
| **prtdata(format=tsd)** | write series from RATS format file to TSD |
| **read(format=tsd)** | read scalars or arrays from TSD into memory |
| **write(format=tsd)** | write scalars or arrays from memory to TSD file |

### Interface Operations

#### Data Wizard

You can read series using *Data (Other Formats)* on the *Data/Graphics* menu.

#### Series Window and RATS Data File Window

You can import series using *File–Import...* or the Import toolbar button. You can export series using *File–Export...*, the Export toolbar button, or the *Export* operation on the contextual menu.

### Details

On **DATA** and **STORE**, you can, if required, use the NOLABELS option to skip automatic label processing and use the **LEFT** and **TOP** options to isolate the data.

See Section 2.8 of the *Introduction* for more details on the format.

## 3.21 FORMAT=WF1 (EViews workfile format)

WF1 is the EViews™ workfile format. It's a Labeled Table format available for series input (only), on all RATS versions.

### RATS Instructions

| | |
|---|---|
| **data(format=wf1)** | read series from WF1 format into memory |
| **store(convert=wf1)** | copy series from WF1 format to RATS format |

### Interface Operations

#### Data Wizard

You can read series using *Data (Other Formats)* on the *Data/Graphics* menu.

#### Series Window and RATS Data File Window

You can import series using *File–Import...* or the Import toolbar button.

### Details

Data in a WF1 file are stored as named series which will be read or imported using those names. You can read all the series by leaving the series list blank on either **DATA** or **STORE**, or you can request specific series by listing just the ones you want. If you import directly into a data list, you will get all series.

Each file has a data frequency and start date, so you can read a subset of entries or change the frequency and get the standard compaction or expansion handling (Section 2.5 of the *Introduction*).

Because these are often "work files" rather than simply data files, they will often include extra series (like residuals) which are derived from the original data series. If you use the Data Wizard, you may want to trim the list of series to include just the data and do any further transformations yourself.

### Example

```
open data yen1kp.wf1
calendar(m) 1973:1
data(format=wf1) 1973:01 1996:07 dm yen
```

## 3.22 FORMAT=WKS (Lotus WKS format)

This is the native format used by the Lotus 1-2-3 spreadsheet and successor programs. While it's no longer the main format used by any current software, there are many legacy datasets which used it. It's a Labeled Table or Unlabeled format (depending upon the file) for both input and output of series, reports and matrices, available in all versions of RATS.

### RATS Instructions

| | |
|---|---|
| **data(format=wks)** | read series from WKS into memory |
| **store(convert=wks)** | copy series from WKS to RATS format |
| **copy(format=wks)** | write series to WKS file |
| **prtdata(format=wks)** | write series from RATS format file to WKS |
| **read(format=wks)** | read scalars or arrays from WKS into memory |
| **write(format=wks)** | write scalars or arrays from memory to WKS file |
| **report(format=wks)** | (with ACTION=FORMAT) write a report to WKS file |

### Interface Operations

#### Data Wizard

You can read series using *Data (Other Formats)* on the *Data/Graphics* menu.

#### Series Window and RATS Data File Window

You can import series using *File–Import...* or the Import toolbar button. You can export series using *File–Export...*, the Export toolbar button, or the *Export* operation on the contextual menu.

#### Report Windows

You can write reports to a WKS file using *File–Save As*, *File–Export*, *Edit–Copy*; the Export and Copy toolbar items, and the *Export* and *Copy* contextual menu operations.

#### Matrix Windows

You can write the contents of a matrix window to a WKS file using *File-Save As*, *File-Export*; the Save and Export toolbars, and the *Export* contextual menu.

The *Copy* operations will generally only allow pasting in WKS format into a spreadsheet.

### Details

On **DATA** and **STORE**, you can, if required, use the NOLABELS option to skip automatic label processing and use the **LEFT** and **TOP** options to isolate the data.

See Section 2.8 of the *Introduction* for more details on the format.

## 3.23 FORMAT=XLS/XLSX (Excel workbook)

These are the spreadsheet format for Microsoft Excel™. It's a Labeled Table or Unlabeled format (depending upon the file) for both input and output of series, reports and matrices, available in all versions of RATS. XLS is the standard format used by Excel through the 2003 version. XLSX is the standard format for Excel 2007 and later.

### RATS Instructions

For all of these, use FORMAT=XLS for Excel 2003 and older formats (BIFF workbooks), and FORMAT=XLSX for Excel 2007 format (zipped XML format)

| | |
|---|---|
| **data(format=xlsx)** | read series from XLSX into memory |
| **store(convert=xlsx)** | copy series from XLSX to RATS format |
| **copy(format=xls)** | write series to XLS file |
| **prtdata(format=xls)** | write series from RATS format file to XLS |
| **read(format=xlsx)** | read scalars or arrays from XLSX into memory |
| **write(format=xls)** | write scalars or arrays from memory to XLS file |
| **report(format=xlsx)** | (with ACTION=FORMAT) write a report to XLSX file |

### Interface Operations

#### Data Wizard

You can read series from XLS or XLSX files using *Data (Other Formats)* on the *Data/Graphics* menu.

#### Series Window and RATS Data File Window

You can import series using *File–Import...* or the Import toolbar button. You can export series using *File–Export...*, the Export toolbar button, or the *Export* operation on the contextual menu.

#### Report Windows

You can write reports using *File–Save As*, *File–Export*, *Edit–Copy*; the Export and Copy toolbar items, and the *Export* and *Copy* contextual menu operations.

#### Matrix Windows

You can write the contents of a matrix window to a file using *File-Save As*, *File-Export*; the Save and Export toolbars, and the *Export* contextual menu.

The *Copy* operations will generally only allow pasting in XLS format into a spreadsheet.

### Details

On **DATA** and **STORE**, you can, if required, use the NOLABELS option to skip automatic label processing and use the **LEFT** and **TOP** options to isolate the data.

See Section 2.8 of the *Introduction* for more details on the format.

# 4. Functions

This chapter lists all of the functions available in RATS, organized alphabetically. The first line provides the function name and argument list along with a brief description of its purpose. A detailed description follows, including a list of the variable types for each argument and the type of value returned by the function.

### %( expression1, expression2, ... ) — Expression evaluation function

Used to group sub-calculations in situations where a comma would be interpreted as an argument separator. For instance:

```
%if( x>=0, %(y=sqrt(x),x), y=%na)
```

if X is non-negative, this returns X while setting Y to the square root of X. If X is negative, this returns a missing value while also setting *y* to the missing value. %(expr1,expr2,…,exprn) returns the value of the final expression in the group.

Argument type: Most expressions. Separate multiple expressions with commas.
Returns type: The type returned by the final expression in the argument list.

### abs(x),%abs(A) — Absolute value of a scalar or matrix

For a scalar argument, ABS returns the absolute value of the scalar. For matrix argument **A**, %ABS returns a matrix **B** such that: $\mathbf{B}(i,j) = \left|\mathbf{A}(i,j)\right|$

Argument type: Real for ABS, array of reals for %ABS
Returns type: Real or array of reals

### %acos( x ) — Arc (inverse) cosine

Returns the inverse cosine of its argument.

Argument type: Real (must have $|x| \le 1$, or returns missing value)
Returns type: Real between 0 and $\pi$

### %allocend() — Default series length

This returns the entry number corresponding to the default series length, which is usually set using the **ALLOCATE** or first **DATA** instruction.

Argument type: None
Returns type: Integer

### %annuity( p, r, n) — Present value of an annuity

Returns the present value of an annuity with payment *p*, interest rate *r*, and term *n*.

Argument types: Reals
Returns type: Real

### %arg( z ) — Argument of a complex number

Returns the argument of a complex number. This is $\tan^{-1}(a/b)$ for $z = a + ib$

Argument type: Complex
Returns type: Real

**%asin( x ) — Arc (inverse) sine**

Returns the inverse sine of its argument

Argument type: Real (must have $|x| \leq 1$, or returns missing value)
Returns type: Real between $-\pi/2$ and $\pi/2$

**%atan( x ) — Arc (inverse) tangent**

Returns the inverse tangent of its argument

Argument type: Real
Returns type: Real between $-\pi/2$ and $\pi/2$

**%avg( A ) — Average of matrix elements**

Returns the average of the elements of **A**. If **A** is symmetric, off-diagonal elements are counted only once.

Argument type: Array
Returns type: Real

**%besselj( n, x) — Bessel function**

Returns the Bessel function of the first kind, for $x$ given order $n$.

Argument types: Integer, Real
Returns type: Real

**%bestrep( A, w ) — "Best" representation**

Returns a string giving a "picture" code (for use in **DISPLAY**, **PRINT** and other output instructions) which "best" represents the values in the array **A** using $w$ positions. In general, this will use all $w$ positions. %MINIMALREP is similar but cuts the number of decimal places if they would be zeros for all numbers.

Argument types: Real array **A** and integer $w$
Returns type: String

**%betainc( x, a, b ) — Incomplete beta function**

Returns the incomplete beta function of $x$ given parameters $a$ and $b$.

Argument types: Real. $0 \leq x \leq 1, a > 0, b > 0$
Returns type: Real

**%bicdf(X,Y,rho) — Cumulative density function of bivariate standard Normal**

Returns $P(x<X,y<Y)$ for a bivariate standard Normal with correlation coefficient *rho*.

Argument type: Real
Returns type: Real

**%bin( x, V ) — Determine appropriate "bin" for a value**

This returns an integer indicating which "bin" the value x would fall in given a vector V of boundary values defining the bins. For example,

```
%bin(2.5,||1.0,2.0,3.0,4.0||)
```

returns integer value 2, as the value 2.5 is in the range (2.0 to 3.0) for the second bin.

Argument types: Real, Vector of Reals
Returns type: Integer

**%binomial( x, n ) — Binomial coefficient**

Returns the binomial coefficient $x(x-1)\ldots(x-n+1)/n!$

Argument types: Real. If $n > x$, $x$ can't be an integer, or a missing value will result.
Returns type: Real

**%block( m, n ) — Get Number of Blocks**

Gives number of blocks (starting with one) when partitioning $m$ integers into blocks of size $n$. For example, `%block(10,5)=2`, `%block(11,5)=3`.

Argument types: Integers
Returns type: Integer

**%blockdiag( VR ) — Create a block diagonal matrix**

Returns a matrix formed by placing the elements of **VR** (a `VECTOR` of `RECTANGULAR` arrays) in blocks down the diagonal, with all other elements of the created matrix set to zero. Note that the matrices in **VR** do not have to be square. The ~\ operator (diagonal concatenation) is usually easier to use.

Argument type: Vector of Rectangulars
Returns type: Rectangular array

**%blockglue( G ) — Matrix concatenation**

The argument is an array of `RECTANGULARS`. `%blockglue()` concatenates these in both dimensions to form a single large matrix. The ~ and ~~ operators (horizontal and vertical concatenation) are usually easier to use.

Argument types: Array of (rectangular) arrays
Returns type: Rectangular array

**%blocksplit( A, I ) — Matrix partitioning**

Returns a `RECTANGULAR` of `RECTANGULAR`s formed by splitting the input array into blocks with dimensions given by the elements of the index array **I**.

Argument types: Real array, Vector of Integers
Returns type: Rectangular array of rectangular arrays

### %boxcox(x,y) — Box-Cox transformation

Returns the Box-Cox transformation of $x$ and $y$, which is given by the formula: $\left(x^{y}-1\right)/y$, or $\log x$ for y=0.

Argument types: Reals
Returns type: Real

### %bqfactor(S,L) — Blanchard-Quah factorization

Returns the Blanchard-Quah factorization of **S** (covariance matrix of residuals) with **L** as the matrix of sums of lag coefficients. The factorization is computed as:

$$\mathbf{L}\left(\mathbf{L}^{-1}\mathbf{S}\mathbf{L}'^{-1}\right)^{1/2}$$

where the matrix square root is the Choleski factorization. Note that after doing an **ESTIMATE**, **S** is available as the reserved variable %SIGMA, and **L** is available as %VARLAGSUMS.

Argument types: Symmetric, Rectangular
Returns type: Rectangular

### %cabs( z ) — Complex absolute value

Returns the absolute value of a complex number $z$. The result is real.

Argument type: Complex
Returns type: Real

### %cal(year,period) — Entry number of specified date

Returns the entry number corresponding to the specified period of the year, given the current **CALENDAR** setting. It is equivalent to the expression year:period. For frequencies such as daily and weekly that are not specified in periods per year, %cal(year,period) returns the entry for that "period" (day or week) of the year.

Argument types: Integers
Returns type: Integer

### %calendar() — Save current CALENDAR setting

Returns the current **CALENDAR** setting, which can be saved into a variable. That variable can be used with the RECALL option on **CALENDAR** to reset the **CALENDAR**.

Argument types: None
Returns type: VECTOR[INTEGER]

### %cdf(x) — Cumulative density function of standard Normal

Returns $\Phi(x)$, where $\Phi$ is the CDF of the standard Normal.

Argument type: Real
Returns type: Real

**%cexp( z ) — Complex $e^z$**

Returns complex $e^z$.

Argument type: Complex
Returns type: Complex

**%chisqr(x,r) — Chi-squared tail probability**

Returns the *tail probability* of a $\chi^2$ with $r$ degrees of freedom, that is, the probability that a $\chi^2$ random variable with $r$ degrees of freedom exceeds $x$.

Argument types: Real
Returns type: Real

**%chisqrdensity( x, r ) — Chi-squared density**

Returns the density at $x$ of a $\chi^2$ with $r$ degrees of freedom

Argument types: Real. $x$ must be non-negative, $r$ must be positive
Returns type: Real

**%chisqrnc(x,r, nc) — Non-central chi-squared CDF**

Returns the CDF of $\chi^2$ with $r$ degrees of freedom for a non-central chi-square, with non-centrality $nc$. Note that this is the CDF, not the tail probability.

Argument types: $x$, $nc$ are Reals, $r$ is Integer, with $x \geq 0$ and $r > 0$
Returns type: Real

**%chisqrncdensity( x, r, nc ) — Non-central chi-squared density**

Returns the density at $x$ of a non-central $\chi^2$ with $r$ degrees of freedom and noncentrality $nc$.

Argument types: $x$, $nc$ are Reals, $r$ is Integer, with $x \geq 0$ and $r > 0$
Returns type: Real

**%choice( label ) — Map Label to Option Choice**

Allows you to specify an argument for a "choice" option on a built-in instruction by supplying the appropriate text label as the argument on `%CHOICE`. For example, you can select the BFGS estimation method on any applicable instruction by using `METHOD=%CHOICE("BFGS")`. Useful in **PROCEDURE**s.

Argument types: Label (must match a "choice" exactly)
Returns type: Integer (but can only be used as the argument for a choice option)

**%clock( m, n ) — Modified modular division**

Does a modified form of modular division, returning [($m$–1) modulo $n$] + 1. Maps to 1,...,$n$.

Argument types: Integer
Returns type: Integer

**%clog( z ) — Complex natural log**

Returns complex $\log_e z$.

Argument type: Complex
Returns type: Complex

**%closestdate(y, m, d, dow) — Observance date (closest day of week)**

Returns the number of the day within a month that is the closest occurrence of the specified day of the week (*dow)* to the date specified by *y,m,d*. For example, Martin Luther King Day is observed in the U.S. on the Monday closest to January 20th. Days of the week are handled as Monday=1 through Sunday=7, so:

```
compute day = %closestdate(2007,1,20,1)
```

returns "22", indicating the closest Monday occurs on 2007:1:22.

Argument types: Integer
Returns type: Integer

**%closestweekday(y, m, d) — Observance date (closest Monday–Friday)**

Returns the day number within the month of the weekday closest to the date specified by *year*,*month*,*day*. For example:

```
dis %closestweekday(2004,7,4)
```

returns "5", because the closest weekday to that day was July 5 (a Monday).

Argument types: Integer
Returns type: Integer

**%cm( x ) — Convert centimeters to inches**

Returns the number of inches in *x* centimeters.

Argument types: Real
Returns type: Real

**%cmplx( real, imag ) — Complex number from real and imaginary parts**

Converts two real numbers into a complex number. The first argument is the real part and the second is the imaginary part.

Argument types: Real
Returns type: Complex

**%cols(A) — Number of columns of a matrix**

Returns the number of columns in a matrix. `%cols(A)` returns the value 1 for arrays of type `VECTOR` (use `%rows()` to get the dimension of a `VECTOR`)

Argument type: Any matrix type
Returns type: Integer

**%compress( A, v ) — Compress empty rows out of an array**

Returns an array with the same number of columns as **A**, but with all rows removed for which the corresponding element of the vector **v** is zero. **A** and **v** must have the same number of rows.

Argument types: **A** is Rectangular or Symmetric array, **v** is a Vector
Returns type: Same as **A**

**%concat( first, last ) — Concatenate two labels**

Concatenates two label variables. This can also be done as *first+last*.

Argument types: Labels
Returns type: Same as argument type

**%conjg( z ) — Complex conjugate**

Returns the complex conjugate of *z*.

Argument type: Complex
Returns type: Complex

**%const( x ) — Fill matrix with a constant value**

In an expression `A = %const(x)`, fills all elements of the array **A** with the value *x*. See also `%FILL` and `%ZEROS`.

Argument type: Real
Returns type: Matrix of reals (must be declared and dimensioned ahead of time)

**%corr( A, B ) — Correlation coefficient**

Computes the correlation coefficient of the two arrays.

Argument type: Real arrays, must have same dimensions ($N\times1$ and $1\times N$ are considered equivalent).
Returns type: Real

**%corrtocv( C, V) — Convert correlation matrix to covariance matrix**

Converts a correlation matrix (**C**) and a vector of variances (**V**) into a covariance matrix.

Argument types: Symmetric, Vector
Returns type: Symmetric

**cos( x ) — Cosine function**

Returns the cosine of *x*, where *x* is in radians. If *x* is a matrix, transforms all elements.

Argument type: Real, or Real matrix
Returns type: Same as argument

**%cosh( x ) — Hyperbolic cosine**

Returns the hyperbolic cosine of its argument $\frac{\exp(x)+\exp(-x)}{2}$

Argument type: Real
Returns type: Real

**%cot( x ) — Cotangent function**

Returns the cotangent of $x$.

Argument type: Real
Returns type: Real

**%cov( A, B ) — Covariance of two arrays**

Returns the covariance of two arrays: $\frac{1}{NM}\sum_{ij}\left(\mathbf{A}_{ij}-\bar{\mathbf{A}}\right)\left(\mathbf{B}_{ij}-\bar{\mathbf{B}}\right)$

Argument types: Real arrays. **A** and **B** must have the same dimensions (RATS considers $N\times 1$ and $1\times N$ to be equivalent).
Returns type: Real

**%cputime()— CPU Timer**

Returns the CPU's internal time in seconds. You can use this to determine the running time of a program, by comparing results of calls to `%CPUTIME` made at the beginning and at the end of the program.

Argument types: None
Returns type: Integer

**%csc( x ) — Cosecant function**

Returns the cosecant of $x$.

Argument type: Real
Returns type: Real

**%csqrt( z ) — Complex Square root**

Returns the complex square root of $z$.

Argument type: Complex
Returns type: Complex

**%cvtocorr( V ) — Converts a covariance matrix to correlations**

Returns the correlation matrix **C** created from a covariance matrix **V**:

$$\mathbf{C}_{ij}=\mathbf{V}_{ii}\Big/\sqrt{\mathbf{V}_{ii}\mathbf{V}_{jj}}$$

Argument type: Symmetric
Returns type: Symmetric

**%cxadj( Z ) — Complex matrix adjoint (conjugate transpose)**

Returns the (conjugate transpose) adjoint of a complex matrix.

Argument type: Complex matrix
Returns type: Complex matrix

**%cxdiag( Z ) — Complex diagonal matrix**

Returns the diagonal complex matrix from a complex vector or $1\times N$ rectangular.

Argument type: Complex vector or $1\times N$ rectangular
Returns type: Complex matrix

**%cxeigdecomp( Z ) — Complex eigen decompostion**

Returns the eigen decomposition of a Hermitian complex matrix. The return is a `VECT[CMATRIX]` with two components—the first are the eigenvalues, the second the eigenvectors.

Argument type: Complex matrix
Returns type: Vector of Complex arrays, with 2 elements

**%cxinv( Z ) — Complex matrix inverse**

Returns the complex inverse of a complex array.

Argument type: Complex matrix (must be $N\times N$)
Returns type: Complex matrix

**%cxsvd( Z ) — Complex singular value decomposition**

Returns the singular value decomposition of a complex ($m\times n$) matrix **Z**. This is a `VECTOR` of three complex matrices, **U**, **W** and **V**, where $\mathbf{Z}=\mathbf{U}\ diag(\mathbf{W})\ \mathbf{V}^*$ (where * denotes the conjugate transpose).

**U** and **V** are column-orthonormal matrices, that is $\mathbf{U}^*\mathbf{U}=\mathbf{I},\mathbf{V}^*\mathbf{V}=\mathbf{I}$. All elements of **W** (the *singular values* of **Z**) are real (though they're stored as complex) and are sorted from largest to smallest. **W** has dimensions $\min(m,n)$ by 1; **U** and **V** are dimensioned to conform with that.

Argument type: Complex matrix
Returns type: Vector of Complex arrays

**%dateandtime( ) — Date and time stamp**

Returns a string with the current date and time.

Argument type: None
Returns type: String

**%datelabel( t ) — Date label for a given date/entry number**

Returns the date label of entry *t*, formatted as RATS does on, for instance, a **PRINT** instruction. The argument can be specified as a date or as an integer entry number.

Argument type: Integer (entry number or date expression)
Returns type: Label

**%day( t ) — Day of month for entry T**

Returns the day of the month (1–31) of entry *t*. Defined only if the **CALENDAR** in effect has a "year month day" format.

Argument type: Integer (entry number or date expression)
Returns type: Integer

**%daycount( t ) — Number of days in period T**

Returns the number of days in entry *t*. For instance, if your **CALENDAR** is set for monthly data, `%DAYCOUNT(2010:11)` is 30. With quarterly data, `%DAYCOUNT(2010:2)` is 91.

Argument type: Integer (entry number or date expression)
Returns type: Integer

**%ddivide( A, v ) — Divide columns of a matrix by vector elements**

Returns the result of dividing the columns of **A** by corresponding elements in `VECTOR` **v**.

Argument type: Rectangular or Symmetric, Vector
Returns type: Rectangular or Symmetric

**%decomp( S ) — Choleski decomposition**

Returns the Choleski decomposition of a `SYMMETRIC` matrix. For a matrix **S**, this returns a lower-triangular `RECTANGULAR` array **F**, such that $\mathbf{FF}'=\mathbf{S}$. **S** must be positive semi-definite. See also the `%PSDFACTOR` function.

Argument type: Symmetric array
Returns type: Rectangular array

**%defined( name ) — Status of procedure parameter or option**

Returns the status of the specified procedure parameter or option. Returns the value 1 if a value was specified for the option or parameter when the procedure was executed (always returns 1 for options which have been assigned default values), otherwise returns a 0. See page UG–475 in the *User's Guide* for details.

Argument type: Parameter or option name
Returns type: Integer

**%density( x ) — Standard normal density function**

Standard Normal density function.

Argument type: Real
Returns type: Real

**%det( A ) — Determinant of an array**

Computes the determinant of a symmetric or $N\times N$ rectangular matrix **A**.

Argument type: Symmetric (p.s.d.) or square rectangular array of reals.
Returns type: Real

**%diag( A ) — Create diagonal matrix from a 1–dimensional array**

Creates a square diagonal matrix from a vector. **A** may be either a VECTOR or an $N\times 1$ or $1\times N$ RECTANGULAR array.

Argument type: Vector or $N\times 1$ or $1\times N$ Rectangular array of reals
Returns type: Rectangular array

**%digamma( x ) — Digamma function**

Returns the digamma function of $x$. The digamma is the derivative of $\log\Gamma(x)$, and is usually denoted $\psi(x)$.

Argument type: Real (positive)
Returns type: Real

**%dims( A ) — Dimensions of a matrix**

Returns the number of rows and columns in an array, as a VECTOR of integers, with the number of rows in entry 1, and the number of columns in entry 2.

Argument type: Any matrix (Vector, Rectangular, or Symmetric)
Returns type: Vector of Integers

**%dlminit( A, SW, F, Z ) — Generate initial conditions for a state-space DLM**

This returns a full solution for the initial conditions for a state space model with the given input matrices. It returns a VECT[RECT] with first component being the (finite) covariance matrix, the second the mean, and the third the diffuse covariance matrix. %PSDINIT solves a more limited problem for strictly stationary transition matrix **A**.

Argument type: Array
Returns type: Vector of Rectangulars

**%dlmgfroma( A ) — Transforming a state-space DLM to stationarity**

This analyzes a state transition matrix **A** from a state-space model and produces the "G" matrix which transforms the model to its stationary states.

Argument type: Array
Returns type: Array

**%dmills( x ) — Derivative of the inverse Mills' ratio**

Returns the derivative evaluated at $x$ of the inverse Mills' ratio for the normal $(\phi/\Phi)$. The inverse Mills' ratio itself is computed by the %MILLS function.

Argument type: Real
Returns type: Real

**%dmult( A, v ) — Multiply columns of a matrix by vector elements**

Returns the result of multiplying the columns of **A** by corresponding elements in VECTOR **v**.

Argument type: Rectangular or Symmetric, Vector
Returns type: Rectangular or Symmetric

**%do( i, n, m, expr) — Internal do loop**

Loops over the expression beginning with $i=n$ as long as $i \leq m$. See the descriptions of **EWISE** and **DO** for examples.

Argument types: *i* must be an integer variable, *n* and *m* are integer values or expressions, *expr* can be any expression
Returns type: None. Should only be done as part of a more complex calculation

**%dow( year, month, day) — Day of week from expanded date**

Returns the day of the week (Monday=1 through Sunday=7) for the date given by *year*,*month*,*day*.

Argument types: Integer
Returns type: Integer

**%dot( A, B ) — Dot product of two arrays**

Returns the "dot" product of **A** and **B**.

Argument types: Vectors or Rectangular arrays of reals. **A** and **B** must have the same dimensions (RATS considers $N \times 1$ and $1 \times N$ to be equivalent).
Returns type: Real

**%easter( year ) — Returns date of (Western) Easter Holiday**

Returns the date of the Easter holiday for *year* in the Western Christian tradition as a number of days after March 22 (the earliest possible Easter date).

Argument types: Integer
Returns type: Integer

**%eigdecomp( S ) — Eigen decomposition of a symmetric matrix**

Returns a 2-vector of RECTANGULAR arrays giving an eigen decomposition of the symmetric array **S**. The first array returned is the $N\times 1$ matrix of eigenvalues, and the second is the $N\times N$ matrix of eigenvectors (in the columns, normalized to unit length).

Argument type: Symmetric matrix
Returns type: Vector of Rectangular arrays

**%eqncoeffs( equation ) — Coefficients of an equation**

Returns a VECTOR with the current coefficients for the specified equation. You can refer to equations by their name or number. Use 0 as the argument to get the coefficient vector from the most recent regression.

Argument type: Equation name or equation number
Returns type: VECTOR of reals

**%eqndepvar( equation ) — Get dependent variable of an equation**

Returns the series number of the dependent variable of the specified equation. Use zero as the argument to get the dependent variable in the most recent regression.

Argument type: Equation name or equation number
Returns type: Integer

**%eqnhandle( equation ) — Get "handle" of an equation**

Returns the integer number which can be used to represent the equation.

Argument type: Equation name or equation number
Returns type: Integer

**%eqnlag( equation , lag ) — Lags right hand side of an equation**

Returns a copy of an equation with all right hand side variables lagged an additional "lag" periods.

Argument types: Equation name or equation number, integer
Returns type: Equation

**%eqnlagpoly( equation , series ) — Extract lag polynomial**

Extracts the lag polynomial for the selected series from an equation. *Equation* can be an equation name or number. Use zero to refer to the most recent regression. This returns a polynomial as a VECTOR of dimension $(n+1)$, where $n$ is the degree of the lag polynomial. The form of the polynomial depends upon which variable is extracted. See Section 5.2 in the *Additional Topics* PDF for details.

Argument types: Equation name or number, series.
Returns type: Vector of Reals

**%eqnprj( equation , t ) — Compute projected values of an equation**

Returns the projected (fitted) value of *equation* at time period *t*. Use 0 for *equation* to use the explanatory variables from the most recent regression. Use `%EQNVALUE` if you want the value given a set of coefficients that you supply rather than the current coefficient values of the equation.

Argument types: Equation name or number, integer or date.
Returns type: Real

**%eqnreglabels( equation ) — Get regressor labels for an equation**

Returns a `VECTOR[LABELS]` with the labels of the regressors (explanatory variables) in an equation, formatted as shown in regression output (e.g., `series{lag}` to show lags). Use zero as the argument to get the labels for the most recent regression.

Argument type: Equation name or equation number
Returns type: Vector of Labels

**%eqnresid( equation, entry ) — Residual value of equation at entry t**

Returns the residual value of the equation at the specified date or entry number.

Argument type: Equation, date or entry number
Returns type: Real

**%eqnrvalue( equation, entry, coeffs ) — Residual of equation given coefficient vector**

Returns the residual value of the equation at the specified date or entry number, given a set of coefficient values provided by the vector *coeffs*.

Argument type: Equation, date or entry number, vector of reals
Returns type: Real

**%eqnserieslag( series, lag ) — Create equation "bit"**

Returns an `EQUATION` with the explanatory variable *`series{lag}`* with a coefficient of 1.0. This can be used in building or adjusting an equation.

Argument type: Series name or number, integer
Returns type: Equation

**%eqnsetcoeffs( equation, C ) — Reset coefficients of an equation**

This sets the coefficients of *equation* to the vector *C*.

Argument type: Equation, Vector of Reals
Returns type: Equation (the first argument)

**%eqngetidentity( equation ) — Returns setting of "identity" tag**

This returns the value of the identity tag for the equation. It returns 1 if the equation is an identity, 0 if it is not.

Argument type: Equation
Returns type: Integer

**%eqnsetidentity( equation, tag) — Set the "identity" tag of an equation**

This sets the "identity" tag for *equation*. This determines whether or not the equation is treated as an identity. To turn on the tag and make the equation an identity, use the value 1 for *tag*. Use the value 0 to turn off the tag.

Argument type: Equation, Integer
Returns type: Equation (the first argument)

**%eqnsetresids( equation, r ) — Set the residuals for an equation**

This sets the residuals for *equation* to be the series *r*.

Argument type: Equation, Series
Returns type: Equation (the first argument)

**%eqnsetvariance( equation, v ) — Reset variance of an equation**

This sets the residual variance of *equation* to the value *v*.

Argument type: Equation, Real
Returns type: Equation (the first argument)

**%eqnsize( equation ) — Number of explanatory variables in equation**

Returns the number of explanatory variables in *equation*.

Argument type: Equation name or number
Returns type: Integer

**%eqntable( equation ) — List of variables/lags in an equation**

Returns a `RECTANGULAR[INTEGER]` array listing the explanatory variables in *equation*. The array has dimensions 2 × *number of explanatory variables*. Row one lists the series numbers of the explanatory variables, row two lists the lags. Use 0 for *equation* to get the table of variables in the most recent regression.

Argument type: Equation name or equation number
Returns type: $2\times N$ Rectangular array of integers

**%eqnvalue( equation, t, coeffs ) — Value of an equation**

Returns the value of the linear *equation* at time period *t*, given *coeffs*—a set of coefficients supplied as a vector. Use 0 for *equation* to use the explanatory variables from the most recent regression. Use `%EQNPRJ` if you want the value of an equation given the current coefficient values of the equation rather than user-inputted values.

Argument type: Equation name or number, entry number, vector of reals
Returns type: Real

**%eqnvariance( equation ) — Variance of an equation**

Returns the variance associated with an equation (generally the residual variance from when it is estimated.)

Argument type: Equation name or number
Returns type: Real

**%eqnxvector( equation , t ) — Extract an X(t) vector for an equation**

Returns a `VECTOR` with the "X" variables from the equation for time period *t*. Use 0 for *equation* to pull the explanatory variables from the most recent regression.

Argument types: Equation name or number, integer
Returns type: Vector

**execpath() — Return path to RATS executable file**

This returns the path to the directory containing the RATS executable file in use. This can be useful if you need to reference a subdirectory of that path (for compiling a procedure file, for example) or need to verify the location of the executable file.

Argument type: None
Returns type: String

**exp(x),%exp( x ) — Scalar or elementwise exponentiation function**

For a scalar real *x*, `exp()` returns $e^x$. For an array **A**, `%exp()` returns an array **B**, such that $\mathbf{B}_{ij}=\exp(\mathbf{A}_{ij})$.

Argument type: Scalar real or integer, or array of reals or integers
Returns type: Same as argument type

**%factorial( x ) — Factorial function**

Returns the factorial $x!=x(x-1)(x-2)\ldots1$ or, more generally $\Gamma(x+1)$.

Argument type: Real. *x* cannot be a negative integer.
Returns type: Real

### %file( fname ) — Get I/O Unit from File Name

Returns an I/O unit for a given file name string. Note: function call or variable set equal to function call must be preceded by the `&` dereferencing character, which tells rats to evaluate the contents of the argument, rather than treating it as a literal string. For example:

```
compute tempunit = %file("c:\myfiles\salesdata.xls")
data(unit=&tempunit),format=xls,org=col)
```

Argument type: String, must be valid file name (can include path)
Returns type: I/O unit

### %fill( rows, cols, value) — Create constant matrix

Creates a matrix with dimensions *rows* by *cols*, with all elements set to *value*.

Argument type: *rows*,*cols* are Integer, *value* is Real
Returns type: Rectangular

### fix( x ) — Convert real to integer by truncation

Converts a real value to an integer value by eliminating the fractional part (not by rounding—use `%round(x,n)` for that). Note that RATS will not automatically convert reals to integers in an expression.

Argument type: Real
Returns type: Integer

### float( n ) — Convert integer to real

Explicitly converts an integer to a real. See page Int–42 in the *Introduction* for details.

Argument type: Integer
Returns type: Real

### %floatingdate( y, m, dow , n) — Day of month for floating holiday

Returns the day of the month of the *n*th occurrence of the specified day of the week (*dow*, coded as Monday=1 to Sunday=7) in the year *y*, month *m*. Use a negative value of *n* to count from the end. `%floatingdate(2010,11,4,4)` is the fourth Thursday of November, 2010, while `%floatingdate(2010,11,4,-1)` is the final Thursday.

Argument types: Integer
Returns type: Integer

### %frac( x ) — Fractional part of a real number

Returns $x$–$[x]$, where $[x]$ is the first integer value less than $x$. For example, `%frac(9.7)` returns .7, `%frac(-9.7)` returns .3.

Argument type: Real
Returns type: Integer

**%fractiles( A, F ) — Compute fractiles**

Returns a VECTOR with fractiles of the elements of the array **A**. The list of desired fractiles are provided by the VECTOR **F**.

Argument type: **A** is a Real array, **F** is a Vector
Returns type: Vector with the same dimension as **F**.

**%freqend() — Number of frequency ordinates**

Returns the number of ordinates set by the **FREQUENCY** instruction.

Argument type: None
Returns type: Integer

**%freqsize( n ) — Compute recommended number of complex ordinates**

Returns the recommended number of ordinates for a **FREQUENCY** instruction, for $n$ actual data points.

Argument type: Integer
Returns type: Integer

**%ftest( x, n, m ) — F-test tail probability**

Returns the tail probability for an $F(n,m)$.

Argument types: Real
Returns type: Real

**%gamma( x ) — Gamma function**

Returns the gamma function of $x$. See also %LNGAMMA.

Argument type: Real
Returns type: Real

**%gammainc( x, a ) — Incomplete gamma function**

Returns the incomplete gamma function of $x$ given parameter $a$.

Argument types: Real, $x \geq 0$, $a > 0$
Returns type: Real

**%gedcdf( x, s ) — CDF of the GED distribution**

Returns the CDF of the GED (generalized error distribution) with shape $s$ at $x$.

Argument types: Real
Returns type: Real

**%getenv( name ) — Get environment string**

For a given operating system environment variable *name*, this returns the associated environment string. Returns a blank string if the environment variable is not defined in the operating system.

Argument types: String or Label
Returns type: String

**%gevcdf( x, k, mu, sigma ) — CDF of Generalized Extreme Value**

Returns the CDF of the Generalized Extreme Value distribution, for tail index *k*, location parameter *mu*, and scale parameter *sigma*.

Argument types: Real
Returns type: Real

**%ginv( A ) — Generalized inverse of a Matrix**

Computes the generalized (Moore-Penrose) inverse of **A**. This is a matrix $\mathbf{A}^+$ which solves $\mathbf{AA}^+\mathbf{A}=\mathbf{A}$ and $\mathbf{A}^+\mathbf{AA}^+=\mathbf{A}^+$. **A** does not have to be square; $\mathbf{A}^+$ will have the same dimensions as **A**.

Argument type: Rectangular array (*m* x *n*)
Returns type: Rectangular array (*n* x *m*)

**%gpcdf( x, k, mu, sigma ) — CDF of Generalized Pareto distribution**

Returns the CDF of the Generalized Pareto distribution, for tail index *k*, location parameter *mu*, and scale parameter *sigma*.

Argument types: Real
Returns type: Real

**%grparm() — Get Current Graph Label Settings**

Returns a vector of integers identifying the current font and type size settings for graph labels. Used with the RECALL option on **GRPARM**.

Argument types: None
Returns type: Vector of integers

**%gsortho( A ) — Gram-Schmidt orthonormalization**

Returns a column-wise Gram-Schmidt orthonormalization of **A**

Argument types: Rectangular
Returns type: Rectangular with same dimensions as **A**.

**%iabs( n ) — Integer absolute value**

Returns the absolute value of the integer *n*.

Argument type: Integer
Returns type: Integer

**%identity( size ) — Create identity matrix**

Creates an identity matrix with dimensions *size* x *size*. *size* must be an integer value.

Argument type: Integer
Returns type: Rectangular array of reals

**%idiv( m, n ) — Integer division**

Integer *m* divided by integer *n*, returning an integer result. Any remainder is ignored.

Argument type: Integer
Returns type: Integer

**%if( x, y, z ) — Evaluate a conditional expression**

Returns *y* if *x* is non-zero, returns *z* if *x* is zero. *x* must be integer or real, but *y* and *z* can be any expressions as long as they are matching types. Often used in **SET** instructions with *x* as a logical expression (that is, if *x* is true, return *y*, otherwise return *z*).

Example: `set bigx = %if(x>=.5 , 1.0, -1.0)`

Argument type: *x*=Logical expression or scalar (zero = false, non-zero = true); *y*,*z* = Any matching types
Returns type: Type of *y* and *z*.

**%imag( z ) — Imaginary part of complex number**

Returns the imaginary part of a complex number *z*.

Argument type: Complex
Returns type: Real

**%imax( m, n ) — Maximum of two integers**

Returns the maximum of the two integer arguments.

Argument types: Integer
Returns type: Integer

**%imin( m, n ) — Minimum of two integers**

Returns the minimum of the two integer arguments.

Argument type: Integer
Returns type: Integer

**%index( V ) — Sorting index for a VECTOR**

Returns a `VECTOR` of `INTEGERS` which gives the sorting index for the vector **V**; that is, a `VECT[INT]` such that `V(%INDEX(V(i)))` would sort **V** into increasing order.

Argument type: Vector, or $N\times 1$ Rectangular
Returns type: Vector of Integers

**%indiv( t ) — Individual within a panel set**

When a panel data **CALENDAR** is set, this returns the number of the individual of which entry $t$ is a part.

Argument type: Integer
Returns type: Integer

**%innerxx( A ) — Inner cross product**

Returns $\mathbf{A}'\mathbf{A}$ for the matrix $\mathbf{A}$. See also %OUTERXX.

Argument types: Real matrix
Returns type: Symmetric

**%instlist() — List of instruments**

Returns the current list of instruments as a regressor list (a VECTOR[INTEGER] array). %INSTTABLE returns the same information, but in the form of a $2 \times n$ array.

Argument types: None
Returns type: Vector of Integers

**%insttable() — Table of instruments**

Returns the table (a $2 \times n$ array of integers) for the current list of instruments. Each column in the return array represents one *series{lag}* pair, with the series in the first row and lag in the second.

Argument types: None
Returns type: Rectangular array of Integers

**%instxvector(entry) — Extract an X(t) for the instrument set**

Returns a VECTOR with the current instruments for time period *entry*.

Argument types: Integer
Returns type: Vector

**inv( A ) — Inverse of a matrix**

Returns the inverse of the non-singular matrix $\mathbf{A}$. $\mathbf{A}$ can be SYMMETRIC or $N \times N$ RECTANGULAR. See %GINV for generalized inverses.

Argument type: Symmetric or square rectangular matrix
Returns type: Rectangular

**%invchisqr( p, r ) — Inverse chi-squared tail probability**

Returns the argument $x$ for which the *tail probability* of a chi-square distribution with $r$ degrees of freedom is $p$. In particular, %chisqr(%invchisqr(p,r),r)=p

Argument types: Real, with $0 \leq p \leq 1$ and $r > 0$
Returns type: Real

**%invchisqrnc( p, r, nc ) — Inverse CDF non-central chi-squared**

Returns the argument $x$ for which the CDF of a non-central chi-square distribution with $r$ degrees of freedom and non-centrality $nc$ is $p$. In particular, `%chisqrnc(%invchisqrnc(p,r,nc),r,nc)=p`

Argument types: $p$ and $nc$ Real, $r$ an Integer. With $0 \le p \le 1$ and $r > 0$
Returns type: Real.

**%invftest( p, n, m ) — Inverse F-test tail probability**

Returns the value $x$ for which the tail probability of an $F$ with $n$ and $m$ degrees of freedom is $p$. In particular, `%ftest(%invftest(p,n,m),n,m)=p`.

Argument types: Real, with $0 \le p \le 1$. $n$ and $m$ must be >0
Returns type: Real

**%invged( p, c ) — Inverse CDF of GED**

Returns the inverse CDF of the generalized error distribution (GED) with shape parameter $c$ (and variance 1). The kernel of the density function is $\exp\left(-|x|^{(2/c)}/2\right)$.

Argument type: Real. $0 \le p \le 1$, $c$ is positive.
Returns type: Real

**%invgev( p, k, mu, sigma ) — Inverse CDF of Generalized Extreme Value**

Returns the inverse CDF of the Generalized Extreme Value distribution, for probability $p$, tail index $k$, location parameter $mu$, and scale parameter $sigma$.

Argument type: Real
Returns type: Real

**%invgp( p, k, mu, sigma ) — Inverse CDF of the Generalized Pareto**

Returns the inverse CDF of the Generalized Pareto distribution, for probability $p$, tail index $k$, location parameter $mu$, and scale parameter $sigma$.

Argument type: Real
Returns type: Real

**%invnormal( p ) — Inverse normal distribution**

Returns the $x$ for which the CDF of a standard Normal distribution is $p$. In particular, `%cdf(%invnormal(p))=p`

Argument type: Real. $0 \le p \le 1$
Returns type: Real

**%invtcdf( p, d ) — Inverse CDF of the t distribution**

Returns the inverse CDF of the $t$ distribution, for the probability $p$ and degrees of freedom $d$.

Argument type: Real
Returns type: Real

**%invttest( p, r ) — Inverse t test**

Returns the $x$ for which the two-tailed probability of a $t$ distribution with $r$ degrees of freedom is $p$. In particular, `%ttest(%invttest(p,r),r)=p`

Argument types: Real, with $0 \leq p \leq 1$ and $r > 0$ .
Returns type: Real

**%isimpson( X, F ) — Numerical integral by Simpson's Rule**

Returns a numerical integral computed using Simpson's Rule. **X** is the vector of grid points (which should be equally spaced in increasing order) and **F** is a corresponding vector of function values.

Argument type: Vectors of Reals (should be same dimension)
Returns type: Real

**%itrapezoid( X, F ) — Numerical integral by Trapezoidal Rule**

Returns a numerical integral computed using the trapezoidal rule. **X** is the vector of grid points (in increasing order, not necessarily equally spaced) and **F** is a corresponding vector of function values.

Argument type: Vectors of Reals (should be same dimension)
Returns type: Real

**%julian( t ) — Number of days from Jan 1, 1901 or Jan 1, 0001**

Returns the number of days to entry $t$ from Jan. 1, 1901 if your **CALENDAR** year is specified using two digits, or from Jan. 1, 0001 if it is specified using four digits.

Argument type: Integer
Returns type: Integer

**%keys( hash ) — Get keys from hash table of strings**

Hash variables are an aggregate variable type where the elements stored in the hash are referenced by character strings, or "keys". This function returns a list of the keys (strings) stored in the *hash* variable.

Argument types: Hash
Returns type: Vector of Strings

**%kroneker( A, B ) — Kroneker product of two arrays**

Returns the Kroneker product of arrays **A** and **B**. For **A**($m$,$n$) and **B**($p$,$q$), this computes an $mp \times nq$ matrix **A**⊗**B**.

Argument types: Any matrix
Returns type: Rectangular

**%kronid( A, B ) — Kroneker identity function**

Computes (**A**⊗**I**)**B**. **A** must be a square matrix; the first dimension of **B** must be a multiple of the dimension of **A**.

Argument type: **A** = Symmetric or square rectangular matrix, **B**= any matrix with first dimension a multiple of the dimension of **A**.
Returns type: Rectangular

**%kronmult( A, B, C ) — Specialized Kroneker multiplication**

Computes (**A**⊗**B**)**C** where **A**⊗**B** is the Kroneker product of **A** and **B**.

Argument types: Any matrices with compatible dimensions
Returns type: Rectangular

**%l( s ) — Get label of a series**

Returns the label of series with the name or number *s*.

Argument type: Series name or number
Returns type: Label

**%label( variable ) — Get label of a series or other variable**

Returns the label attached to the specified variable (that is, the name of the specified series, equation, etc.). Unlike `%L`, if you use a series number, you must use a type modifier to tell RATS that you want the label of a series.

Argument type: Variable name, series or equation number
Returns type: Label

**%left( s, n ) — Left substring**

Returns the leftmost $n$ characters of the string *s*.

Argument types: *s* is a string or label. $n$ is integer.
Returns type: String

**%lnbeta( a, b ) — Natural log of the beta function**

Returns the natural log of the beta function B($a$,$b$).

Argument type: Real, non-negative
Returns type: Real

**%lngamma( x ) — Natural log of the Gamma function**

Returns the natural log of $\Gamma(x)$.

Argument type: Real, non-negative
Returns type: Real

**%lnlogistic( x ) — Natural log of the logistic CDF**

Returns $\log\left(\exp(x)/\left(1+\exp(x)\right)\right)$, that is, the log of the logistic CDF.

Argument type: Real
Returns type: Real

**log( x ) — Natural log function**

Returns the natural log of $x$ ($\log_e x$).

Argument type: Real
Returns type: Real

**%log( A ) — Elementwise natural log function**

Elementwise log function. This returns an array **B** such that $\mathbf{B}_{ij}=\log_e(\mathbf{A}_{ij})$.

Argument type: An array
Returns type: An array of same type and dimension as **A**

**%logbetadensity( x, $\alpha$, $\beta$ ) — Log Density of Beta**

Returns the log density of the beta distribution for $x$, given $\alpha$ and $\beta$.

Argument type: Real
Returns type: Real

**%logcdf( v, x ) — Natural log of the normal CDF**

Returns $\log\left(\Phi\left(x/\sqrt{v}\right)\right)$ where $v$ is the variance, and $\Phi$ is the normal CDF.

Argument type: Reals
Returns type: Real

**%logconcdensity( $\hat{\Sigma}$ , T ) — Log concentrated MV normal density**

Returns the log concentrated density for a (multivariate) normal with covariance matrix $\hat{\Sigma}$ and number of observations $T$:

$$-\frac{Tk}{2}\left(\log\left(2\pi\right)+1\right)-\frac{T}{2}\log\left|\hat{\Sigma}\right|$$

Argument types: $\hat{\Sigma}$ is $k\times k$ Symmetric (or scalar Real), $T$ is an Integer.
Returns type: Real

**%logdensity( A, v ) — Log multivariate normal density**

Returns the log density for a (multivariate) normal with covariance matrix **A** evaluated at the deviations from mean of **v**:

$$-(1/2)\ (k\log 2\pi+\log\left|\mathbf{A}\right|+\mathbf{v}'\mathbf{A}^{-1}\mathbf{v})$$

Argument types: **A** is $k\times k$ Symmetric, **v** is a Vector (or $k\times 1$ or $1\times k$ Rectangular). Both can be scalars as well.
Returns type: Real

**%logdensitycv( $\Sigma$, $\hat{\Sigma}$ , T ) — Log multivariate normal density**

Returns the log of multivariate normal density as a function of $\Sigma$, $\hat{\Sigma}$ , and the number of observations ($T$).

$$-\frac{Tk}{2}\log\left(2\pi\right)-\frac{T}{2}\log\left|\hat{\Sigma}\right|-\frac{T}{2}\ \text{trace}\ \Sigma^{-1}\hat{\Sigma}$$

Argument types: $\Sigma$ and $\hat{\Sigma}$ are $k\times k$ Symmetric (or scalar Reals), $T$ is an integer.
Returns type: Real.

**%logdensitydiag( V, u ) — Log multivariate normal density with diagonal covariance**

Similar to %LOGDENSITY except that the covariance matrix is diagonal, and V is a vector with the diagonal elements.

Argument type: Both VECTOR of the same dimension.
Returns type: Real

**%logdetxx( S ) — Log determinant of a symmetric matrix**

Returns $\log(|\mathbf{S}|)$ for a symmetric matrix **S**.

Argument type: Symmetric
Returns type: Real

**%logdirichlet( x, d ) — Log Density of Dirichlet**

Returns the log density of Dirichlet$\left(\mathbf{x}|\mathbf{d}\right)$. **x** and **d** must have the same dimensions, the **d**'s must be positive, and the **x**'s must sum to one.

Argument types: Vectors, must be of the same dimension.
Returns type: Real

**%loggammadensity( x, c, b) — Log Gamma density**

Returns the log density evaluated at *x*, for a gamma distribution with shape parameter *c* and scale *b*.

Argument type: Real. *c* and *b* must be positive.
Returns type: Real

**%loggeddensity( x, c, v ) — Log GED density**

Returns the log density evaluated at *x*, for a GED (generalized error distribution) with shape parameter *c* and variance *v*. Note that the scale parameter for the distribution is determined indirectly from the shape and the variance.

Argument type: Real. *c* and *v* must be positive.
Returns type: Real

**%loggevdensity( x, k, mu, sigma ) — Log GEV density**

Returns the log of the density function for the Generalized Extreme Value distribution for *x*, given tail index *k*, location parameter *mu*, and scale parameter *sigma*.

Argument type: Real
Returns type: Real

**%loggpdensity( x, k, mu, sigma ) — Log Generalized Pareto density**

Returns the log of the density function for the Generalized Pareto distribution for *x*, given tail index *k*, location parameter *mu*, and scale parameter *sigma*.

Argument type: Real
Returns type: Real

**%logistic( x, b ) — Logistic function**

Returns the logistic function: $1/\left(1+be^{-x}\right)$. This is "safeguarded" to avoid overflows.

Argument types: Real
Returns type: Real

**%lognegbin( x, r, p ) — Log of the Negative binomial density**

Returns the log of the negative binomial density of *x* given *r* and *p*.

Argument types: Reals
Returns type: Real

**%logpoissonk( mean, k ) — Log of Poisson probability**

Returns the log of $P(x=k|m=mean)$ for a Poisson process with the mean given by *mean*.

Argument types: Reals
Returns type: Real

**%logtdensity( V, U, nu ) — Log Multivariate t density**

Returns the log of the multivariate *t*-density with *nu* degrees of freedom and covariance matrix **V,** evaluated at the vector of deviations from the mean **U**. Note that **V** is the covariance matrix of the *t* distribution itself, not of the underlying multivariate Normal on which it is based.

Argument type: **V** is Symmetric, **U** is Vector (same dimension as **V**), *nu* is a positive-valued Real (**V** and **U** can also both be reals to evaluate the univariate log density).
Returns type: Real.

**%logtdensitystd( V, U, nu ) — Log Multivariate t density**

Returns the log of the standardized multivariate *t*-density with *nu* degrees of freedom and covariance matrix **V,** evaluated at the vector of deviations from the mean **U**. This is equivalent to: `%logtdensity(v*nu/(nu-2),u,nu)`

Argument type: **V** is Symmetric, **U** is Vector (same dimension as **V**), *nu* is a positive-valued Real (**V** and **U** can also both be reals to evaluate the univariate log density).
Returns type: Real.

**%ltinv( L ) — Lower triangular inverse**

For a packed matrix **L**, this returns the inverse of the lower-triangular portion.

Argument type: `PACKED` matrix of reals
Returns type: `PACKED` matrix of reals

**%ltouterxx( L ) — Lower triangular outer product**

For a packed matrix **L**, this returns the `SYMMETRIC` $\mathbf{LL}'$.

Argument type: `PACKED` matrix of reals
Returns type: `SYMMETRIC` matrix of reals

**%matpeek( A, Coords ) — Extracting entries from a sparse matrix**

Returns a vector formed from extracting from the matrix **A** the row,column elements from the 2 x *m* `RECT[INTEGER]` **Coords** in the matrix **A**.

Argument types: **A** is Rectangular, **Coords** is a Rectangular of Integers
Returns type: Vector

**%matpoke( A, Coords, v ) — Filling entries in a sparse matrix**

Copies data from the VECTOR **v** (dimension *m*) to the *row*,*column* pairs elements specified by the 2 x *m* RECT[INTEGER] **Coords**. The remaining entries of **A** are unaffected.

Argument types: **A** is Rectangular, **Coords** a Rectangular of Integers, **V** a Vector
Returns type: Rectangular (returns modified **A**)

**%max( x, y ) — Maximum of two reals**

Returns the maximum of two real values, *x* and *y*.

Argument types: Real
Returns type: Real

**%maxindex( V ) — Location of maximum value in a VECTOR**

Returns the element number at which the maximum value of a VECTOR is attained.

Argument type: Vector
Returns type: Integer

**%maxvalue( A ) — Get maximum value of an array**

Returns the maximum value of the array **A**.

Argument type: Any array
Returns type: Real

**%mid( s, m, n ) — Middle substring**

Returns the *n* characters from the string *s* starting at position *m*. (Count positions beginning at 1). If $n \leq 0$, all characters from position *m* on are included.

Argument types: *s* is a string or label, *m* and *n* are integers
Returns type: String

**%mills( x ) — Inverse Mills' ratio**

Returns the inverse Mills' ratio for a normal ($\phi/\Phi$) evaluated at *x*.

Argument type: Real
Returns type: Real

**%min( x, y ) — Minimum of two reals**

Returns the minimum of two real values, *x* and *y*.

Argument types: Reals
Returns type: Real

**%minimalrep( A, w ) — Determines "minimal" representation**

Returns a string giving a "picture" code for use by **PRINT**, **DISPLAY** or other output instructions which provides the "minimal" representation for the elements of **A** using at most $w$ positions. This will be shorter than the representation given by %BESTREP if all elements of **A** can be shown to their full precision with only a few decimal places.

Argument types: **A** is a real matrix, $w$ is a positive integer
Returns type: String

**%minindex( V ) — Location of minimum value in a VECTOR**

Returns the element number at which the minimum value of a VECTOR is attained.

Argument type: Vector
Returns type: Integer

**%minus( A ) — Negative ("minus") values of an array**

Returns negative elements of array, with positive values replaced with zeros: $\mathbf{B}_{ij} = \min\left(\mathbf{A}_{ij}, 0\right)$

Argument type: Real or Real array
Returns type: Same as argument

**%minvalue( A ) — Get minimum value of an array**

Returns the minimum value of the array A.

Argument type: Real array
Returns type: Real

**%mod( m, n ) — Modulo arithmetic**

Returns the remainder from dividing $m$ by $n$ (integer arithmetic). See also %CLOCK(m,n) which does modulo arithmetic but returns values between 1 and $n$.

Argument type: Integer
Returns type: Integer

**%modelcompanion( M ) — Companion matrix for a model**

Returns the companion matrix for a model. The model must only include linear equations—the function will not work if the model includes non-linear formulas.

Argument type: Model
Returns type: Real array

**%modeldepvars( M ) — Get dependent variables from a model**

Returns a VECTOR[INTEGER] which gives the series numbers for the dependent variables of the equations or formulas in a model.

Argument type: Model
Returns type: Vector of Integers

**%modeleqn( M, n ) — Get equation from a model**

Returns a copy of the equation at position *n* in model *M*.

Argument types: *M* is a Model, *n* is an Integer
Returns type: Equation

**%modelfind( M, s ) — Locate an equation in a model**

Returns the position in model *M* of the equation or formula which has the series *s* as its dependent variable. Returns 0 if there is none.

Argument type: Model, Integer
Returns type: Integer

**%modelgetcoeffs( M ) — Get coefficients from a model**

Returns a RECTANGULAR with the coefficients from the equations of model *M*. Each column of the output array will have the coefficients from a different equation.

Argument type: Model
Returns type: Rectangular

**%modelgetvcv( M ) — Get covariance matrix from a model**

Retrieves the covariance matrix of model *M*.

Argument type: Model
Returns type: Symmetric

**%modellagmatrix( model, k ) — Lagged Dependent Variable Coefficents**

This returns the $N\times N$ matrix of the coefficients for lag $k$ of the dependent variables of a model (usually a VAR model).

Argument types: *model* is a MODEL, *k* is an integer
Returns type: Integer

**%modellagsums( M ) — Lag sums for a VAR model**

Computes the lag sums for a VAR model *M*. This is the matrix $\mathbf{I}-\sum_{s=1}^{l}\Phi_s$ where $\Phi_s$ is the matrix of VAR coefficients for lag *s*

Argument type: Model
Returns type: Rectangular

**%modellabel( M, i ) — Label of dependent variable in a model**

Returns the label of the dependent variable of the equation or formula at position *i* in model *M*.

Argument type: Model
Returns type: Label

**%modellargestroot( M ) — Dominant root of model**

Returns the absolute value of the largest root of the companion matrix of *M*.

Argument type: Model
Returns type: Real

**%modelpoke( M, i, equation ) — Replace an equation in a model**

This replaces the equation at position *i* in model *M* with *equation*.

Argument type: Model, Integer, Equation
Returns type: Model

**%modelsetcoeffs( M, C ) — Set coefficients from a model**

Sets the coefficients of the equation of a model. The coefficients are taken from the array **C**, with the coefficients for equation *i* in the model being taken from column *i* of **C**.

Argument type: *M* is a Model, **C** is a Rectangular array
Returns type: None

**%modelsetvcv( M, S ) — Set covariance matrix of a model**

Sets **S** as the residual covariance matrix of model *M*.

Argument type: Model, Symmetric
Returns type: None

**%modelsize( M ) — Number of equations in a model**

Returns the number of equations or formulas in a model.

Argument type: Model
Returns type: Integer

**%modelsubstect( M ) — Substitute out error correction terms**

Substitutes the error-correction terms out of a model, producing a model written in terms of the original variables only.

Argument type: Model
Returns type: Model

**%month( t ) — Month number (1–12) of entry T**

Returns the month corresponding to entry *t*, where 1 is January, 2 is February, etc.

Argument type: Integer
Returns type: Integer

**%mqform( A, B ) — Matrix quadratic form operation**

Returns $\mathbf{B}'\mathbf{AB}$.

Argument types: **A** must be $N\times N$ Rectangular or Symmetric; **B** can be any array that allows the $\mathbf{B}'\mathbf{AB}$ operation, given the dimension of **A**.
Returns type: Symmetric array if **A** is Symmetric, otherwise Rectangular

**%mqformdiag( A, B ) — Diagonal of matrix quadratic form**

Returns a `VECTOR` containing the diagonal elements of $\mathbf{B}'\mathbf{AB}$.

Argument types: Same as for `%mqform(a,b)`, described above
Returns type: Vector

**%mscalar( x ) — Create a scalar matrix**

Creates a scalar matrix equal to the identity matrix times the value $x$. You can only use this in an expression such as `A=%MSCALAR(x)`, where **A** is an $N\times N$ matrix which has already been declared and dimensioned.

Argument type: Real
Returns type: Symmetric or Rectangular array

**%mspexpand( P ) — Markov switching probability function**

Takes the reduced size $(N-1)\times N$ matrix of transition probabilities **P** (parameterized that way for estimation purposes) and expands it to a full $N\times N$ matrix.

Argument type: Array
Returns type: Array

**%negbin( x, r, p ) — Negative binomial density**

Returns the negative binomial density $\binom{r+x-1}{x}p^r(1-p)^x$

Argument types: Reals
Returns type: Real

**%noprec( x ) — Tests for loss of precision**

Returns 1 or 0 depending upon whether $x$ is "machine-zero".

Argument type: Real
Returns type: Real (0 or 1 value)

**%normsqr( A ) — Squared norm of a matrix**

Returns the sum of squared elements of **A**.

Argument types: Real array
Returns type: Real

**%nullspace( A ) — Orthonormal basis for null space**

Returns an orthonormal matrix $\mathbf{A}^{\perp}$ which forms a basis for the null space of **A**. %NULLSPACE is similar to %PERP, but it does not assume that the input matrix has full rank in its smaller dimension. As a result, the dimensions of the output matrix will depend upon the rank of **A**, not just upon its dimensions.

Argument types: Rectangular array
Returns type: Rectangular array

**%nyblomtest( x, p ) — Nyblom fluctuations distribution**

Returns the (approximate) tail probability of a standard Nyblom (1989) fluctuations test with *p* components. This is computed using a saddlepoint approximation, and should be quite accurate in the tails.

Argument types: *x* is a non-negative real, *p* is a positive integer
Returns type: Real

**%outerxx( A ) — Outer cross product**

Returns $\mathbf{AA}'$ for the matrix **A**. Use %INNERXX(A) to get $\mathbf{A}'\mathbf{A}$.

Argument types: Real array
Returns type: Symmetric array

**%ovcheck( x ) — Checks for potential overflow**

Returns a missing value if *x* is so large as to probably be the result of an unstable calculation. This will typically force the parameters generating a function evaluation to be rejected as out-of-range. If *x* is a "normal" number, %OVCHECK returns *x*.

Argument type: Real
Returns type: Real (either *x* or missing value)

**%panelobs( ) — Size of panel data time dimension**

Returns the number of periods per individual for panel data as was set on the **CALENDAR** instruction.

Argument type: None
Returns type: Integer

**%panelsize() — Number of Individuals in Panel Set**

Returns the number of individuals given the current **CALENDAR(PANELOBS=...)** setting and total number of observations. %PANELOBS() gives the number of time periods.

Argument type: None
Returns type: Integer

**%parmset( ) — Returns the default PARMSET**

This returns the current default parameter set.

Argument type: None
Returns type: Parameter Set

**%parmslabels( P ) — Extract parameters from a PARMSET**

Returns the labels of the variables in a parameter set.

Argument type: Parameter set
Returns type: `VECTOR[STRINGS]`

**%parmspeek( P ) — Extract parameters from a PARMSET**

Returns the parameters from a `PARMSET` in vector form.

Argument type: Parameter set
Returns type: Vector

**%parmspoke( P, v ) — Put vector into a PARMSET**

Resets the parameters in `PARMSET` **P** from the vector *v*.

Argument types: Parameter set, Vector
Returns type: Vector (returns *v*)

**%patchmat( Base, Source ) — Replace NA's in a matrix**

Returns the matrix formed by patching over (replacing) any NA's in the **Base** matrix with the corresponding entries from the **Source** matrix. **Base** and **Source** must have conforming dimensions.

Argument types: Any matrix (both must have same dimensions)
Returns type: Matrix of same type/dimensions as **Base**

**%patchzero( Base ) — Replace NA's in a matrix with zeros**

Returns the matrix formed by patching over (replacing) any NA's in the **Base** matrix with zeros.

Argument types: Any matrix
Returns type: Matrix of same type/dimensions as **Base**

**%payment( a, r, n ) — Required payments for an annuity**

Returns the required payment for an annuity with principal $a$, interest rate $r$, and term $n$ ($a$, $r$, and $n$ are real values).

Argument types: Reals
Returns type: Real

**%period( t ) — Period within a panel set**

For entry $t$, returns the time period within either the individual (for panel data) or the day (for intra-day data), or year for other calendars.

Argument type: Integer
Returns type: Integer

**%perp( A ) — Perp operator**

Returns an orthonormal matrix $\mathbf{A}^{\perp}$ which forms a basis for the null space of $\mathbf{A}$. If $m>n$ (that is, $\mathbf{A}$ has more rows than columns), this is a basis for the left null space, so $\mathbf{A}^{\perp\prime}\mathbf{A}=0$. $\mathbf{A}^{\perp}$ is $m\times(m-n)$. If $m<n$, it will be a basis for the right null space, $\mathbf{A}\mathbf{A}^{\perp}=0$, with $\mathbf{A}^{\perp}$ being $n\times(n-m)$.

Argument types: Rectangular array
Returns type: Rectangular array

**%plus( A ) — Non-negative ("plus") values of an array**

Returns positive elements of array, with negative values replaced with zeros: $\mathbf{B}_{ij}=\max(\mathbf{A}_{ij},0)$.

Argument type: Real or Real array
Returns type: Same as argument

**%poisson( mean, k ) — Cumulative Poisson probability**

Returns $P(x\leq k|m=mean)$ for a Poisson process with mean given by *mean*.

Argument types: Reals
Returns type: Real

**%poissonk( mean, k ) — Poisson probability**

Returns $P(x=k|m=mean)$ for a Poisson process with mean given by *mean*.

Argument types: Reals
Returns type: Real

**%polyadd( V1, V2 ) — Add two polynomials**

Adds two polynomials, represented by the vectors **V1** and **V2**, where an $n+1$ vector $\mathbf{V}$ represents an $n$-degree polynomial, in the form $\mathbf{V}(1)+\mathbf{V}(2)x^1+\ldots+\mathbf{V}(n+1)x^n$.

Argument types: **V1** and **V2** are Vectors
Returns type: Vector

**%polycxroots( V ) — Complex roots of a polynomial**

Returns all roots of a polynomial represented as described under `%POLYADD`.

Argument type: Vector
Returns type: Vector of Complex

**%polydiv( V1, V2, d ) — Divide two polynomials**

Divides polynomial **V1** by polynomial **V2** (where **V1** and **V2** are VECTORS) up to degree *d*.

Argument types: **V1** and **V2** are Vectors, *d* is an integer
Returns type: Vector

**%polymult( V1, V2 ) — Multiply two polynomials**

Multiplies two polynomials, represented by the vectors **V1** and **V2**.

Argument types: Vectors
Returns type: Vector

**%polyroots( V ) — Roots of a polynomial**

Returns all real roots of a polynomial represented by the vector **V**.

Argument type: Vector of Reals
Returns type: Vector of Reals

**%polysub( V1, V2 ) — Subtract two polynomials**

Subtracts two polynomials, represented by the vectors **V1** and **V2**.

Argument types: Vectors
Returns type: Vector

**%polyvalue( V, x ) — Evaluate a polynomial**

Returns the value of a polynomial (represented by the vector **V**) for a given value of *x*.

Argument types: Vector, Real
Returns type: Real

**%psddiag( S , B ) — Diagonalizing a symmetric matrix**

For the positive semi-definite SYMMETRIC **S**, returns a lower triangular matrix **L** such that $\mathbf{LSL}'$ is diagonal with 1's and 0's on the diagonal. **B** is used to determine when a calculation has produced a zero (to machine precision). If **S** is full rank, %PSDDIAG(S,B) will give $\mathbf{LSL}'=\mathbf{I}$.

Argument types: Symmetric (both should be positive semi-definite)
Returns type: Rectangular

**%psdfactor( A , I ) — Factoring symmetric matrix**

Returns the Choleski factor of the positive semi-definite SYMMETRIC **A** with ordering given by the index list in the VECTOR[INTEGER] **I**. The function %DECOMP(A) is equivalent to %PSDFACTOR(A,||1,2,...,n||)

Argument types: **A** is Symmetric, **I** is a Vector of Integers
Returns type: Rectangular

**%psdinit( A , W ) — Stationary covariance matrix**

Returns a SYMMETRIC matrix **V** such that $\mathbf{AVA}' + \mathbf{W} = \mathbf{V}$. This is the stationary covariance matrix for a state-space model. %DLMINIT can solve a generalized problem with unit roots in the transition matrix **A**.

Argument types: **A** is $N \times N$ Rectangular; **W** is $N \times N$ Symmetric
Returns type: Symmetric

**%psubmat( A, startrow, startcol, B ) — Copy values into an array**

Copies information in matrix **B** into matrix **A**, starting at (*startrow*,*startcol*) of **A**. Copied values overwrite any existing values in the affected entries of **A**. **A** must be sufficiently large to allow all entries in **B** to be copied into existing dimensions of **A**.

Argument type: **A** and **B** matrices, *startrow* and *startcol* integers
Returns type: None

**%psubvec( VA, position, VB ) — Copy values into a vector**

Puts the information in vector **VB** into vector **VA**, beginning at entry *position* of **VA**. The copied values will overwrite any existing values in the affected entries of **VA**. Note that **VA** must be sufficiently large enough to allow all entries of **VB** to be copied into the existing dimensions of **VA**.

Argument types: **VA** and **VB** are Vectors, *position* is an integer
Returns type: None

**%pt( V, t, X ) — Filling a matrix of series**

**V** is an array of SERIES, **X** should be an array of reals of the same size and shape. The $t$ elements of the series in **V** are filled with the corresponding values of vector **X**.

Argument types: **V** is an array of Series, $t$ an integer, **X** an array of reals
Returns type: None

**%qform( A, B ) — Evaluate a quadratic form**

Returns $\mathbf{B}'\mathbf{AB}$, where **B** is either a VECTOR or an $N \times 1$ or $1 \times N$ RECTANGULAR array.

Argument types: Vector or Rectangular array
Returns type: Real

**%qformd( A, B ) — Evaluate a quadratic form with a diagonal matrix**

Returns $\sum_i A_i B_i^2$ where **A** and **B** are either conforming VECTORS or $N \times 1$ or $1 \times N$ RECTANGULAR arrays.

Argument types: Vector or Rectangular array
Returns type: Real

**%qformdpdf( D, x ) — PDF of a diagonal quadratic form**

Returns $P(\mathbf{e}'\mathbf{De} < x)$ where $\mathbf{D}$ is a diagonal matrix (represented by a vector).

Argument types: Vector (diagonal elements of matrix), Real
Returns type: Real

**%qforminv( S, v ) — Inverse quadratic form**

Returns $\mathbf{v}'\mathbf{S}^{-1}\mathbf{v}$ where $\mathbf{S}$ is positive semi-definite symmetric.

Argument types: Symmetric, Vector
Returns type: Real

**%qformpdf( A, x ) — PDF of a quadratic form**

Computes $P(\mathbf{e}'\mathbf{Ae} < x)$ for a quadratic form in Normal(0,1) variables, where $\mathbf{A}$ is SYMMETRIC and $\mathbf{e}$ is i.i.d. This can be used for ratios of quadratic forms as well, since $P(\mathbf{e}'\mathbf{Ae}/\mathbf{e}'\mathbf{Be} < x) = P(\mathbf{e}'(\mathbf{A} - x\mathbf{B})\mathbf{e} < 0)$.

Argument types: Symmetric, Real
Returns type: Real

**%qrdecomp( A ) — QR decomposition**

Returns a VECTOR[RECTANGULAR] with two elements, the $\mathbf{Q}$ and $\mathbf{R}$, respectively, of $\mathbf{QR} = \mathbf{A}$, where $\mathbf{Q}$ is orthonormal $(\mathbf{QQ}' = \mathbf{I})$ and $\mathbf{R}$ is upper triangular.

Argument types: Rectangular array, which must be square
Returns type: 2-element Vector of Rectangular arrays.

**%ran( x ) — Random normal draw**

Returns draws from a Normal distribution with mean zero and standard deviation $x$. If you use this in an expression where the left side of the expression is an array that has already been declared and dimensioned (A=%RAN(1.0)), this will fill all elements of the array with random draws. Otherwise, it returns a single real value.

Argument types: Real
Returns type: Real or array of reals, depending on context of expression.

**%ranbeta( a, b ) — Random beta draw**

Returns draws from a Beta distribution with shape parameters $a$ and $b$. If you use this in an expression where the left side of the expression is an array that has already been declared and dimensioned (A=%RANBETA(3.0,5.0)), this will fill all elements of the array with random draws. Otherwise, it returns a single real value.

Argument types: Real. Both must be positive.
Returns type: Real or array of reals, depending on context of expression.

**%ranbranch( P ) — Random selection of a branch**

Returns a random integer from {1,..,dim(**P**)} where the elements of **P** are the (unscaled) probabilities of the choices.

Argument types: Array of reals
Returns type: Integer

**%ranchisqr( d ) — Random chi-squared draw**

Returns draws from a chi-squared distribution with $d$ degrees of freedom. If you use this in an expression where the left side of the expression is an array that has already been declared and dimensioned (`A=%RANCHISQR(10.0)`), this will fill all elements of the array with random draws. Otherwise, it returns a single real value.

Argument types: Real
Returns type: Real or array of reals, depending on context of expression.

**%rancombo( n , k ) — Random combination**

Returns a `VECTOR[INTEGER]` which is a random combination (draw without replacement) of $k$ items from {1,...,$n$}.

Argument types: $n$ and $k$ are integers, both positive
Returns type: Vector of Integers with dimension $k$

**%randirichlet( C ) — Random Dirichlet**

Returns a draw from a Dirichlet distribution. The Dirichlet is a generalization of the beta distribution. C is an $n$-vector of positive shape parameters. The return is an $n$-vector of probabilities (positive values summing to one).

Argument type: Vector of Reals
Returns type: Vector of Reals

**%ranflip( p ) — Random draw from Bernoulli distribution**

Returns random draws (zeros or ones) from a Bernoulli distribution with probability $p$. If you use this in an expression where the left side of the expression is an array that has already been declared and dimensioned (`A=%RANFLIP(0.6)`), this will fill all elements of the array with random draws. Otherwise, it returns a single draw.

Argument types: Real
Returns type: Real or array of reals (zeros or ones)

**%rangamma( r ) — Random gammas**

Returns draws from a gamma distribution with shape parameter $r$. If you use this in an expression where the left side of the expression is an array that has already been declared and dimensioned (`A=%RANGAMMA(10.0)`), this will fill all elements of the array with random draws. Otherwise, it returns a single real value.

Argument type: Real. $r > 0$
Returns type: Real or array of reals, depending on context of expression.

**%rangrid( X, F ) — Random draw from approximate distribution**

Returns a random draw from a distribution approximated by the grid. **X** is the vector of grid points (in increasing order, not necessarily equally spaced) and **F** is a corresponding vector of (relative) densities.

Argument type: Vectors of Reals (should be same dimension)
Returns type: Real

**%raninteger( l , u) — Random integer**

Returns an integer drawn randomly from $[l, u]$.

Argument types: Integer
Returns type: Integer

**%ranks( A ) — Ranks of the elements of an array**

Returns an array with the same size and type as **A** with the ranks of the corresponding elements of **A**. Ranks are in increasing order beginning with 1. Ties are assigned the average of the covered ranks.

Argument type: Real array
Returns type: Real array

**%ranlogkernel() — Log kernel density from recent draw**

Returns the log kernel density from the most recent draw of random numbers.

Argument types: None
Returns type: Real

**%ranmat( m, n) — Rectangular matrix of random normal draws**

Returns an $m \times n$ matrix of random draws from a standard Normal(0,1) distribution.

Argument types: Integer
Returns type: Rectangular

**%ranmvkron( FS, FX ) — Draw from multivariate Normal regression**

Returns a draw from a multivariate Normal regression. Inputs are a factor of the regression $\Sigma$ matrix (**FS**) and a factor of the $(\mathbf{X}'\mathbf{X})^{-1}$ matrix (**FX**).

Argument types: Rectangular
Returns type: Rectangular (dim **FX** $\times$ dim **FS**).

**%ranmvkroncmom(C, H, P, M) — MV normal regression, CMOM variant**

Returns a draw from a multivariate Normal regression given a cross-moment matrix **C**, regression precision matrix (**H**), prior precision matrix (**P**) and prior mean (**M**).

Argument types: Symmetric, Symmetric, Symmetric, Vector or Rectangular
Returns type: Same as fourth argument

**%ranmvnormal( F ) — Draw from random multivariate Normal**

Returns a draw from a multivariate Normal with mean 0 and covariance matrix $\mathbf{FF}'$, that is, the argument provides a factor of the covariance matrix.

Argument types: Rectangular
Returns type: Vector with the same number of rows as **F**.

**%ranmvpost( P1, M1, P2, M2) — Draw from a MV Normal posterior**

Returns a draw from a multivariate Normal posterior combining multivariate Normals, with the given precisions (**P1** and **P2**) and means (**M1** and **M2**).

Argument types: Symmetric, vector, symmetric, vector.
Returns type: Vector with the same dimensions as **P1** and **P2**.

**%ranmvpostcmom( C, h, P, M) — MV Normal posterior, CMOM variant**

Returns a draw from a multivariate Normal posterior combining multivariate Normals, given a cross-moment matrix **C**, a regression precision (*h*), prior precision matrix (**P**) and prior mean (**M**).

Argument types: Symmetric, Real, Symmetric, Vector
Returns type: Vector with the same dimension as **M**.

**%ranmvt( F, nu ) — Random draw from multivariate t**

Returns a random draw from a multivariate *t* distribution, where *nu* is the degrees of freedom and F is a factor of the covariance matrix (the covariance matrix is $\mathbf{FF}'$) .

Argument types: *F* is a rectangular array, *nu* is real.
Returns type: Symmetric

**%ranpermute( n ) — Random permutation**

Returns a `VECTOR[INTEGER]` which is a random permutation (reordering) of {1,...,*n*}.

Argument types: Integer
Returns type: Vector of Integers, where the vector has *n* elements.

**%ransphere( n ) — Random draw on the unit sphere**

Returns a draw made uniformly from the unit sphere in $n$ dimensions.

Argument type: Integer
Returns type: Vector

**%rant( d ) — Random draw from Student t distribution**

Returns a random draw from a $t$ distribution with $d$ degrees of freedom.

Argument type: Real
Returns type: Real, unless in the specific form `A=%RANT(D)`, where **A** is a real array with dimensions already set, in which case a joint multivariate $t$ draw is placed into the elements of **A**.

**%rantruncate( mu, sig, low, up ) — Random draw from truncated Normal**

Returns a draw from a Normal with mean *mu* and standard deviation *sig* truncated to the interval [*low*,*up*]. *low* or *up* can be `%NA` to indicate an interval unbounded in that direction. If you use this in an expression where the left side of the expression is an array that has already been declared and dimensioned (`A=%RANTRUNCATE(...)`), this will fill all elements of the array with random draws. Otherwise, it returns a single real value.

Argument types: Real
Returns type: Real or array of reals, depending on context of expression.

**%ranwishart( n, r ) — Random Wishart matrix**

Returns a draw from a $n \times n$ Wishart distribution with identity covariance matrix and shape parameter $r$.

Argument types: $n$ is integer, $r$ is a positive real. $r$ must be greater than $n$.
Returns type: Symmetric

**%ranwishartf( F, r ) — Random Wishart with given covariance matrix**

Returns a draw from a Wishart distribution with covariance matrix $\mathbf{FF}'$ and shape parameter $r$.

Argument types: **F** is Rectangular, $r$ is a positive real. $r$ must be greater than dim(**F**).
Returns type: Symmetric

**%ranwisharti( F, r ) — Random inverse Wishart**

Returns a draw from an inverse Wishart distribution with covariance matrix $\mathbf{FF}'$ and shape parameter $r$.

Argument types: **F** is Rectangular, $r$ is a positive real number greater than dim(**F**).
Returns type: Symmetric

**%ratsversion() — Get version number of RATS**

This returns the version number of the copy of RATS you are using (for example, 8.00).

Argument type: None
Returns type: Real

**%real( z ) — Get real part of complex number**

Returns the real part of a complex number *z*.

Argument type: Complex
Returns type: Real

**%regend() — Return regression ending entry**

Returns the ending entry/date of the most recently completed regression or estimation.

Argument type: None
Returns type: Integer

**%reglist() — Get regressor list**

Returns the regressor list from the most recent regression, as a `VECTOR` of `INTEGERS`. You can use `%RLADDONE` and related functions to modify the regressorlist.

Argument type: None
Returns type: Vector of Integers

**%regload(R) — Reload saved regression**

Loads into memory a regression saved using `%REGSAVE()`. Subsequent instructions such as **`TEST`**, other hypothesis test operations, and **`PRJ`** will apply to this regression.

Argument type: Regression (Vector of Integers created using `%REGSAVE()`)
Returns type: None

**%regsave() — Save a regression**

Saves the most recent regression, as a `VECTOR` of `INTEGERS`.

Argument type: None
Returns type: Vector of Integers (with coded information)

**%regstart() — Return regression starting entry**

Returns the starting entry/date of the most recently completed regression or estimation.

Argument type: None
Returns type: Integer

**%reshape( A, r, c ) — Change the dimensions of an array**

Returns the contents of **A** with dimensions changed (usually from rectangular to vector or vice versa). *r* and *c* provide the number of rows and columns, respectively, in the new array. See also %VEC and %VECTORECT.

Argument type: **A** is an array, *r* and *c* are integers
Returns type: *r* × *c* RECTANGULAR

**%right( s, n ) — Right substring**

Returns the rightmost *n* characters from the string *s*. Use the %MID function to get the rightmost characters from a known position within the string.

Argument types: *s* is String or Label, *n* is a positive Integer
Returns type: String

**%rladdlag( R, S, L ) — Add a lagged regressor to a regressor list**

Adds a lag of a series to an existing regressor list. See also %rladdone().

Argument types: *R* is a regressor list (Vector of Integers), *S* is a series, and *L* is an integer lag number
Returns type: Regressor list (Vector of Integers)

**%rladdlaglist( R, S, L ) — Add list of lags to a regressor list**

Adds a series with a list of lags to a regressor list. This is identical to %RLADDLAG(), except *L* is a list of lag numbers (as a Vector of Integers), rather than a single lag.

Argument types: *R* is a regressor list (Vector of Integers), *S* is a series, and *L* is a Vector of Integers with a list of lags
Returns type: Regressor list (Vector of Integers)

**%rladdone( R, S ) — Add a regressor to a regressor list**

Adds a series (unlagged) to an existing regressor list.

Argument type: *R* is a regressor list (Vector of Integers), *S* is a series
Returns type: Regressor list (Vector of Integers)

**%rlconcat( R1, R2 ) — Concatenate two regressor lists**

Concatenates two regressor lists into a single list.

Argument types: *R1* and *R2* are regressor lists (Vector of Integers)
Returns type: Regressor list (Vector of Integers)

**%rlcount( R ) — Number of regressors in list**

Returns the total number of variables in a regressor list once lag fields are expanded.

Argument type: *R* is a regressor list (Vector of Integers)
Returns type: Integer

**%rlempty( ) — Empty Regressor List**

Returns an empty regressor list; this is identical to a `VEC[INT]` with dimension 0.

Argument type: None
Returns type: Vector of Integers

**%rlfromeqn( Eqn ) — Get a regressor list from an equation**

Returns the regressor list from the specified equation.

Argument type: Equation
Returns type: Regressor list (Vector of Integers)

**%rlfromtable( Table ) — Get a regressor list from a regression table**

Returns a regressor list from a regression table (such as generated by `%EQNTABLE` or `%INSTTABLE`).

Argument type: *Table* is a regression table—a $2 \times n$ array of integers where each column in the array represents one `series{lag}` pair, with the series number in the first row and lag number in the second row.
Returns type: Regressor list (Vector of Integers)

**%rlverify( R ) — Verify a regressor list**

Verifies that a list is a valid regressor list. Useful when creating your own regressor lists or modifying existing regressor lists.

Argument type: *R* is a regressor list (Vector of Integers)
Returns type: Integer: 1 if a regressor list is good, 0 if not.

**%round( x, n ) — Rounding**

Returns *x* rounded to *n* decimal places. Negative values of *n* round to positive powers of 10, that is, `%round(1536,-2)=1500`.

Argument type: *x* is real, *n* is integer
Returns type: Real

**%rows( A ) — Number of rows in a matrix**

Returns the number of rows in a matrix **A**.

Argument type: **A** can be any type of matrix.
Returns type: Integer

**%rsscmom( C, V ) — Residual sum of squares from a cross product**

This returns the residual sum of squares given a (properly structured) cross moment matrix **C** and a coefficient vector **V**.

Argument type: Symmetric $(K+1)\times(K+1)$, Vector $(K)$.
Returns type: Real

**%s( L ) — Series from label**

Returns the series number for the series whose name is the label L. If such a series doesn't yet exist, it will be created. For instance, `SET %S("TREND"+I) = T^I`, will, if I is 2, create series TREND2 equal to a squared trend.

Argument type: Label variable or expression
Returns type: Series number (integer)

**%scalar( A) — Get first element of a matrix**

Returns **A**(1,1) (or **A**(1) for a VECTOR).

Argument type: **A** can be any real-valued array.
Returns type: Real

**%scol( S, col ) — Series from Column in Array of Series**

Returns a vector of series handles for a column of a matrix of series.

Argument type: **S** is an array of series, *col* is an integer
Returns type: VECTOR of series handles

**%sec( x ) — Secant function**

Returns the secant of *x*.

Argument type: Real
Returns type: Real

**%selseries() — Return series selected in the Series List**

This returns the list of series selected in the *Series List* window.

Argument type: None
Returns type: Vector of Integers with series numbers.

**%seq( m, n ) — Creates an Integer Sequence**

Returns a VECTOR[INTEGER] with the sequence from *m* to *n*. (*m*,*n*) can be in either order.

Argument type: Integers
Returns type: Vector of Integers

**%seqa( start, incr, n ) — Creates a Vector with a Real-Valued Sequence**

Returns a VECTOR of dimension *n* filled with the sequence starting with *start* and incremented by *incr* for each subsequent entry.

Argument type: *start*, *incr* are Reals, *n* is an Integer
Returns type: Vector of Reals

**%seqrange( lower, upper, n ) — Creates a Vector with a Real-Valued Sequence**

Returns a VECTOR of dimension *n* filled with a sequence of constant-magnitude increments from *lower* to *upper*.

Argument type: *upper*, *lower* are Reals, *n* is an Integer
Returns type: Vector of Reals

**%seriesentry( series, entry ) — Panel data series entry references**

When working with panel data, references to a series entry in a **SET** or **FRML** command will return "NA" (missing value code) if the entry referenced is in a different individual than the entry being set or evaluated—this is done to prevent lag computations from "lagging across" individuals. If you want the actual value of the series entry, regardless of whether it is in the current individual, use "%seriesentry(*series*,*entry*)" rather than just "*series*(*entry*)".

Argument type: *series* is a series, *entry* a date or integer entry number.
Returns type: Real

**%sigmacmom( C, B ) — Sum of squared error matrix from cross product**

This returns the sum of squared multivariate error matrix, given a (properly structured) cross moment matrix **C** and a coefficient matrix **B**.

Argument type: Symmetric $(K+N)\times(K+N)$, Rectangular $N\times K$
Returns type: Symmetric $N\times N$

**%sign( x ) — Sign function**

Returns 1 if *x* is positive, –1 if it's negative and 0 if it's zero.

Argument type: Real
Returns type: Real

**sin( x ) — Sine function**

Returns the sine of *x*, where *x* is in radians. If *x* is a matrix, transforms each element.

Argument type: Real, or real-value matrix.
Returns type: Same as argument

**%sinh( x ) — Hyperbolic sine**

Returns the hyperbolic sine of its argument: $\left(\exp(x)-\exp(-x)\right)/2$

Argument type: Real
Returns type: Real

**%size( A ) — Size of an array**

Returns the total number of elements in an array

Argument type: Any matrix (Vector, Rectangular, or Symmetric)
Returns type: Integer

**%slike( template ) — Series Handles by Template**

Returns a vector of series handles matching a name template.

Argument type: LABEL, supports # or * as wildcards
Returns type: VECTOR of series handles

**%solve( A, B ) — Solves linear equations**

Returns solution $\mathbf{x}$ of $\mathbf{Ax} = \mathbf{B}$. $\mathbf{A}$ is an $N \times N$ real matrix, $\mathbf{B}$ is an $N$-element vector.

Argument types: Square matrix (Rectangular or Symmetric), Vector
Returns type: Vector

**%sort( A ) — Sorts an array**

Returns an array which has the same size and shape as **A**, but with the elements of **A** sorted in increasing order. If **A** is RECTANGULAR, the sorted values are stored by columns.

Argument types: Real matrix or vector
Returns type: Real matrix or vector

**%sortc( A, c ) — Sort on a column**

Returns a copy of array **A** sorted based upon the values in column *c*. This normally sorts in increasing order. Make *c* negative to sort on a column in *decreasing* order.

Argument type: Rectangular array, Integer
Returns type: Rectangular

**%sortcl( A, list ) — Sort on multiple columns**

Returns a copy of array **A** sorted on the values of the columns in the list. The list should be a VECTOR of INTEGERS, or a "in-line matrix" of the form ||*col1*,...,*coln*||. The first column listed is the primary key, the others are used in order to break ties. Use the negative of a column number to have that column sort in decreasing order.

Argument type: Rectangular array, Vector of Integers
Returns type: Rectangular

**sqrt( x ) — Square root function**

Returns the square root of *x*.

Argument type: Real
Returns type: Real

**%sqrt( A ) — Elementwise square root**

Elementwise square root function. This returns an array $\mathbf{B}_{ij} = \sqrt{\mathbf{A}_{ij}}$

Argument type: Real array
Returns type: An array of the same type and dimension as **A**

**%srow( S, row ) — Series from Row in Array of Series**

Returns a vector of series handles for a row of a matrix of series.

Argument type: **S** is an array of series, *row* is an integer
Returns type: VECTOR of series handles

**%stereo( V ) — Stereo projection**

Returns the conversion of a general $\nu-1$ element vector to an $n$ vector on the unit sphere using the stereo projection.

Argument type: Vector
Returns type: Vector

**%strcmp( s1, s2 ) — Compare two strings**

Compares two strings. Returns 0 if they are identical, 1 if *s1*>*s2* in lexicographical order, and –1 if *s1*<*s2*. Note that the return for equal strings is 0, not 1.

Argument types: Strings
Returns type: Integer

**%strcmpnc( s1, s2 ) — Compare two strings disregarding case**

Compares two strings without regard to case. Returns 0 if they are identical, 1 if *s1*>*s2* in lexicographical order, and –1 if *s1*<*s2*. Note that the return for equal strings is 0, not 1.

Argument types: Strings
Returns type: Integer

**%strescape( s , chars , escape ) - Escape special characters**

Returns a copy of s with any character from chars escaped by prefixing with escape.

Argument type: Strings
Returns type: String

**%strfind( s , f ) - Find substring**

Searchs string *s* for string *f*, returning the position in *s* if found and zero otherwise.

Argument type: Strings
Returns type: Integer

**%string( n ) — Convert integer value to a string**

Returns the character string corresponding to the integer $n$ (for instance, %STRING(110) returns "110").

Argument type: Integer
Returns type: String

**%strlen( s ) — Returns the length of a string**

Returns the length (number of characters) of the string *s*.

Argument type: String
Returns type: Integer

**%strlower( s ) — Convert string to lower case**

Returns a copy of the string *s* with all characters in lower case.

Argument type: String
Returns type: String

**%strmatch( s, pattern ) — Test string for pattern match**

Tests the string *s* for a match with *pattern*, returning 1 for a match and 0 for a failure to match. The *pattern* can include the following "wildcard" characters: * for matching 0 or more characters, ? for matching exactly one character, # for matching exactly one digit. Any other character must be an exact match (case is ignored for letters). For example, the pattern "*###" matches any string ending with three digits.

Argument type: Strings
Returns type: Integer

**%strrep( s, n ) — Repeats a string**

Returns a string that consists of the input string *s* repeated *n* times.

Argument type: String, integer
Returns type: String

**%strtrim( s ) — Trim a string**

Returns a copy of a string with leading and trailing spaces and quotes removed.

Argument type: String
Returns type: String

**%strupper( s ) — Convert string to upper case**

Returns a copy of the string *s* with all characters in upper case.

Argument type: String
Returns type: String

**%strval( x, pc ) — String showing a numerical value**

This returns a string that displays the real value *x* in the picture code *pc*. For instance, `%strval(11.9,"*.##")` will return the string 11.90.

Argument type: Real, string
Returns type: String

### %sum( A ) — Sum of array elements

Returns the sum of all elements in the array **A**. For SYMMETRIC arrays, it computes the sum of the lower triangle only. This can also be applied to SERIES variables, in which case the sum will be computed over the defined range of the series, or the range set by an **SMPL** instruction. Any missing observations will be omitted from the computation.

Argument type: Real array or series
Returns type: Real

### %sumc( A ) — Sums columns of an array

This computes the sums of each column in an array. For an $M\times N$ array, this returns an $N\times 1$ array where element $n$ contains the sum of all the elements in column $n$ of the array **A**. Any missing observations will be omitted from the computation.

Argument type: Rectangular array
Returns type: Rectangular

### %sumr( A ) — Sums rows of an array

This computes the sums of each row in an array. For an $M\times N$ array, this returns an $M\times 1$ array where element $m$ contains the sum of all the elements in row $m$ of the array **A**. Any missing observations will be omitted from the computation.

Argument type: Rectangular array
Returns type: Rectangular

### %svdecomp( A ) — Singular Value Decomposition

Returns a VECTOR[RECTANGULAR] with three elements, the **U**, **W** and **V**, respectively, of $\mathbf{A}=\mathbf{U}\left[\text{diag}\left(\mathbf{W}\right)\right]\mathbf{V}'$, where **U** and **V** are column orthonormal, and **W** is the vector of diagonal elements in the SVD. The singular values will be sorted in descending order.

Argument types: Rectangular array, can be any dimensions
Returns type: Vector of Rectangulars, with three elements in the vector

### %sweep( A, k ) — Sweep function of a matrix

Returns the result of sweeping matrix **A** on pivot $k$. **A** must be a square RECT or SYMM array. $k$ is any integer from 1 to $N$. **A** is an $N\times N$ matrix. `B = %SWEEP(A,k)` returns:

$$\begin{aligned}
\mathbf{B}(k,k) &= 1.0/\mathbf{A}(k,k) \\
\mathbf{B}(i,k) &= -\mathbf{A}(i,k)/\mathbf{A}(k,k) && i\neq j \\
\mathbf{B}(k,j) &= \mathbf{A}(k,j)/\mathbf{A}(k,k) && j\neq k \\
\mathbf{B}(i,j) &= \mathbf{A}(i,j)-\mathbf{A}(i,k)\mathbf{A}(k,i)/\mathbf{A}(k,k) && \text{otherwise}
\end{aligned}$$

Note that `%SWEEP(%SWEEP(A,k),k)= A`. See Section 5.1 in the *Additional Topics* PDF for more on the use of the sweep functions.

Argument type: Symmetric or square Rectangular
Returns type: Square Rectangular array (even if **A** is Symmetric)

**%sweeplist( A, list ) — Sweep function of a matrix**

Returns the result of sweeping matrix **A** on the pivots given by the VECTOR[INTEGER] list. The sweep function is described with %SWEEP.

Argument type: Symmetric or square Rectangular array, Vector of Integers
Returns type: Square Rectangular array (even if **A** is Symmetric)

**%sweeptop( A, k ) — Sweep function of a matrix**

Returns the result of sweeping matrix **A** on all pivots 1,...,*k*. The sweep function is described under %SWEEP.

Argument type: Symmetric or square Rectangular array, integer
Returns type: Square Rectangular array (even if **A** is Symmetric)

**%symmcol( n ) — Get column number from packed position**

Returns the column number corresponding to element *n* in a packed symmetric or packed lower triangular matrix.

Argument type: Integer ($n \geq 1$, $n \leq m(m+1)/2$)
Returns type: Integer

**%symmpos( row, col ) — Get position of value in symmetric array**

Returns the position in a packed symmetric matrix for a given row and column in the unpacked matrix.

Argument type: Integer (*row* and *col* $\geq 1$)
Returns type: Integer

**%symmrow( n ) — Get "unpacked" row from symmetric array**

Returns the row number corresponding to element *n* in a packed symmetric matrix.

Argument type: Integer ($n \geq 1$, $n \leq m(m+1)/2$)
Returns type: Integer

**%tableextract( table, list ) — Extract subtable**

Returns a subtable with the items in the listed positions of the argument table. The list would usually be generated using %TABLEFINDALL or %TABLEFINDMISS. For example, to create a regression table containing regressors used in BIGTABLE that are not included in SMALLTABLE, you could do:

```
compute tablediff = %TABLEFINDMISS(bigtable,smalltable)
compute newtable = %TABLEEXTRACT(bigtable,tablediff)
```

Argument type: Regression table (2 x *n* Rectang. of Integers), Vector of Integers
Returns type: Regression table

**%tablefind( table, s, l ) — Locate a single variable in a table**

For series *s* and lag *l*, this locates the requested variable (series{lag}) in regression table *table*. It returns the position of the variable, or zero if it's not present.

Argument types: *table* is a regression table (2 x *n* Rectangular of Integers), *s* is a Series, and *l* is an Integer lag number.
Returns type: Integer (column number)

**%tablefindall( table1, table2 ) — Locate variables in a table**

Locates in *table1* the series,lag pairs from *table2*. It returns a `VECTOR[INTEGER]` with the positions of the regressors in *table1* that also appeared in *table2*.

Argument types: Regression Table (2 x *n* Rectangular of Integers)
Returns type: Vector of Integers (position numbers)

**%tablefindlags( table, s ) — Locate lags of a series in a table**

Returns the list of lags of series *s* which appear in *table* .

Argument types: *table* is a regression table (2 x *n* Rectangular of Integers), *s* is a series.
Returns type: Vector of Integers (list of lags). Could be dimension 0.

**%tablefindmiss( table1, table2 ) — Locates variables missing from table**

Returns a `VECTOR` of `INTEGERS` which shows the positions of variables in *table1* that are not also in *table2*.

Argument type: Regression tables (2 x *n* Rectangular of Integers)
Returns type: Rectangular of Integers

**%tablefromrl( R ) — Create table from regressor list**

Creates a regressor table from a regressor list.

Argument types: *R* is a regressor list (Rectangular of Integers)
Returns type: A regression table (2 x *n* Rectangular of Integers)

**%tablemerge( table1, table2 ) — Merge two regressor tables**

Merges two regression tables into a single regression table. Duplicates are included just once.

Argument types: Regression tables (2 x *n* Rectangular of Integers)
Returns type: Regression table

**tan( x ) — Tangent function**

Returns the tangent of *x*, where *x* is in radians. If applied to a matrix, all elements are transformed.

Argument type: Real, or real-value matrix.
Returns type: Same as argument

**%tanh( x ) — Hyperbolic tangent**

Returns the hyperbolic tangent of its argument:

$$\frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$$

Argument type: Real
Returns type: Real

**%tcdf( x, d ) — CDF for t distribution**

Returns the CDF at $x$ for a $t$ distribution, with degrees of freedom $d$.

Argument types: Real
Returns type: Real

**%tcdfnc( x, d, nc ) — CDF for non-central t distribution**

Returns the CDF at $x$ for a non-central $t$ distribution, with degrees of freedom $d$, and non-centrality parameter $nc$.

Argument type: Real
Returns type: Real

**%tdensity( x, d ) — t density**

Returns the density function at $x$ for a $t$ distribution with $d$ degrees of freedom.

Argument type: Real. $d$ must be positive.
Return type: Real

**%tdensitync( x, d, nc ) — Non-central t distribution**

Returns the density function at $x$ for a non-central $t$ distribution, with degrees of freedom $d$, and non-centrality parameter $nc$.

Argument type: Real
Returns type: Real

**%testdiff( base, test ) — Test difference of two arrays**

Returns the convergence criterion used by the non-linear estimation instructions applied to *base* and *test*. The formula is:

$$\max_i \left\{ \min\left[ \left| \left(test_i - base_i\right)/base_i \right|, \left| test_i - base_i \right| \right] \right\}$$

Argument types: Real vectors of same dimension
Returns type: Real

**%today() — Today's entry**

Returns the entry number in the current calendar which covers today's date (that is, the system date of the computer at run time).

Argument type: None
Return type: Integer

**tr( A ) — Transpose of a matrix**

Returns the transpose of matrix **A**.

Argument type: Real $M\times N$ Array
Return type: Real $N\times M$ Array

**%trace( A ) — Trace of a matrix**

Returns the trace of a SYMMETRIC or $N\times N$ matrix **A**.

Argument type: Symmetric or square Rectangular array
Return type: Real

**%tradeday( t, dow ) — Number of trading day occurrences**

Returns the number of occurrences of weekday *dow* (Monday=1, Tuesday=2, …, Sunday=7) in period *t*. For instance, January 2010 started on a Friday, and so has five of each of Friday, Saturday and Sunday, and four of the other days. With a monthly calendar, `%TRADEDAY(2010:1,1)` is 4 while `%TRADEDAY(2010:1,5)` is 5.

Argument type: Integer
Return type: Real

**%trigamma( x ) — Trigamma function**

Returns the trigamma function of *x*. The trigamma is the second derivative of $\log\Gamma(x)$, and is usually denoted $\psi'(x)$. ($\psi(x)$ is the digamma function).

Argument type: Real (positive)
Returns type: Real

**%tsign( x, y ) — Sign transfer**

Returns $|x|$ if *y* is positive, $-|x|$ if *y* is negative, 0 if *y* is zero.

Argument types: Real
Return type: Real

**%ttest( x, r ) — Two-tailed t test**

Returns the two-tailed probability of a *t* with *r* degrees of freedom, that is, the probability that a *t*-distributed random variable with *r* degrees of freedom exceeds *x* in absolute value.

Argument types: Real
Returns type: Real

**%uniform( l, u ) — Random draw from a uniform distribution**

Returns a random draw from a uniform distribution ranging from (the real values) $l$ to $u$. If you use this in an expression where the left side of the expression is an array (that is, `A=%UNIFORM(0.0,1.0)`), this will fill all elements of the array with random draws. Otherwise, it returns a single real value.

Argument types: Real
Returns type: Returns an array of reals or single real value, depending on context.

**%unit( x ) — Unit circle value**

Returns the unit circle value $\exp(ix)$.

Argument type: Real
Returns type: Complex

**%unit2( n, T ) — Unit circle value for 2π(n-1)/T**

Returns the unit circle value $\exp\left(2\pi i(n-1)/T\right)$.

Argument types: Integers
Returns type: Complex

**%unitfn( s ) — Get file name for I/O unit**

Returns the filename currently assigned to the specified I/O unit (supplied as a string). For example, if you have done

```
open data mydata.xls
```

then `%unitfn("data")` would return "`mydata.xls`".

Argument type: String (unit name)
Returns type: String

**%unitfnext( s ) — Get extension of file name for I/O unit**

Returns the extension (characters after the last period) of the filename associated with the specified I/O unit. For example, if you have done

```
open data mydata.xls
```

then `%unitfnext("data")` would return "`xls`".

Argument type: String (unit name)
Returns type: String

**%unitfnpath( s ) — Get path of file for I/O unit**

Returns the path name (not including the filename) of the file associated with the specified I/O unit. For example, if you have done

```
open data c:\rats\projects\mydata.xls
```

then `%unitfnpath("data")` would return "`c:\rats\projects`".

Argument type: String (unit name)
Returns type: String

**%unitfnroot( s ) — Get root name of file for I/O unit**

Returns the main part of the filename, not including the last period or any extension, for the file associated with the specified I/O unit. For example, if you have done

```
open data mydata.xls
```

then `%unitfnroot("data")` would return "`mydata`".

Argument type: String (unit name)
Returns type: String

**%unitv( n, i ) — Generating a unit vector**

This generates the *i*th unit *n*-vector—a vector with *n* elements, with element *i* set to one, and the other elements set to zero.

Argument types: Integers
Returns type: Vector

**%valid( x ) — Check for valid (non-missing) value**

Returns a value of 1 if and only if (all elements of) *x* are *not* the system missing value (`%NA`). In all other cases, it returns a zero. Use it to test for missing values. If applied to a matrix, all elements must be valid to get a 1 return value.

Argument type: Real or Real-valued matrix.
Returns type: Real

**%value( s ) — Value from string**

Returns the value of a number coded into the string *s*.

Argument type: String
Returns type: Real

**%vec( A ) — Vectorizing an array**

Returns a `VECTOR` containing the values of array **A**. If **A** is `RECTANGULAR`, **A** is stripped out by columns; if `SYMMETRIC` or `PACKED`, by the rows of the lower triangle.

Argument type: Real array
Returns type: `VECTOR`

**%vectorect( V, r ) — Create a RECTANGULAR from a Vector**

Reformats the values of the vector **V** into a Rectangular array with $r$ rows. Elements 1 through $r$ of the vector go into column 1 of the rectangular array, elements $r+1$ through $2r$ go into column 2, and so on. This is the "inverse" of the %VEC function applied to a RECTANGULAR array with $r$ rows.

Argument type: Vector
Returns type: Rectangular

**%vectosymm( V, r ) — Create a SYMMETRIC from a Vector**

Reformats the values of the vector **v** into a SYMMETRIC with $r$ rows, filling the symmetric array row by row. This is the "inverse" of the %VEC function applied to a SYMMETRIC with $r$ rows.

Argument type: Vector of Reals
Returns type: Symmetric of Reals

**%weekday( t ) — Day of week of entry T**

Returns the day of the week (1=Monday through 7=Sunday) of entry $t$.

Argument type: Integer
Returns type: Integer

**%wfractiles( A, W, F ) — Compute fractiles**

Returns a VECTOR with fractiles of the elements of the array **A** with (un-normalized) weights provided by **W**. **W** needs to have the same shape as **A**. The list of desired fractiles are provided by the VECTOR F.

Argument type: **A** and **W** are real arrays, **F** is a Vector
Returns type: Vector

**%xcol( A, i ) — Extract column of a matrix**

Extracts column $i$ from matrix **A**.

Argument type: Real matrix
Returns type: Real matrix ($N\times 1$)

**%xdiag( A ) — Extract diagonal from a matrix**

Extracts the diagonal from matrix **A**.

Argument type: Real matrix
Returns type: Real matrix ($N\times 1$)

**%xrow( A, i ) — Extract row**

Extracts row $i$ from matrix **A**.

Argument type: Real matrix
Returns type: Real matrix ($1\times N$)

**%xsubmat( A, startrow, endrow, startcol, endcol ) — Extract a matrix**

This extracts an $M \times N$ rectangular matrix from matrix **A**. For example, to extract the upper-left 4x4 elements of **A**, you would use `%xsubmat(A,1,4,1,4)`.

Argument type: **A** matrix, and integer-valued row and column numbers
Returns type: Real matrix

**%xsubvec( V, startrow, endrow ) — Extract a vector**

Extracts elements *startrow* through *endrow* of vector **V**.

Argument type: **V** is a Vector, *startrow* and *endrow* are integers
Returns type: Real matrix

**%xt( V, t ) — Extract from an array of SERIES**

From an array of `SERIES`, it extracts the entries $t$ of the component series to form an array with the same size and shape as **V**.

Argument type: **V** is an array of Series, $t$ is an integer.
Returns type: Real matrix ($N \times 1$)

**%year( t ) — Year number**

Returns the calendar year corresponding to entry $t$.

Argument type: Integer
Returns type: Integer

**%ymdaycount( y, m ) — Number of days in specified year/month**

Returns the number of days in month $m$ of year $y$. For example, `%ymdaycount(2012,9)` returns 30, the number of days in September, 2012.

Argument type: Integers
Returns type: Integer

**%ymddow( y, m, d ) — Day of week for specified date**

Returns the day of the the week (1=Monday,...,7=Sunday) for the date $y$:$m$:$d$. For example, `%ymddow(2012,9,1)` returns 6, as September 1st, 2012 is a Saturday.

Argument type: Integers
Returns type: Integer

**%ymdjulian( y, m, d ) — Julian date for specified date**

Returns the Julian date (number of days since January 1st, 0001) for the date $y$:$m$:$d$.

Argument type: Integers
Returns type: Integer

**%zeros( rows, cols ) — Create a zero matrix**

Returns a matrix with dimensions *rows*×*cols*, with all elements equal to zero.

Argument types: Integers
Returns type: Rectangular of Reals

**%zlag( t, x ) — Transfer function for lag**

Returns $\exp(-i2\pi(t-1)x/N)$ where $N$ is the number of frequencies set on the **FREQUENCY** instruction. As $t$ runs from 1 to $N$, this gives the appropriate Fourier transform for the lag operator $L^x$.

Argument type: Integer ($t$) and real ($x$)
Returns type: Complex

**%ztest( x ) — Two-tailed standard normal probability**

Returns the two-tailed probability for Standard Normal, that is, the probability that a Normally distributed random variable exceeds $x$ in absolute value.

Argument type: Real
Returns type: Real

# 5. More on Functions

Here we take a closer look at some useful RATS functions.

**Using %SWEEP**

**Polynomials**

## 5.1 The Sweep Family of Functions

The %SWEEP function, and related %SWEEPTOP and %SWEEPLIST functions, are very useful, and probably don't get as much use as they deserve.

B=%SWEEP(A,k) is the following:

$$\begin{aligned}
&\mathbf{B}(k,k) = 1/\mathbf{A}(k,k) \\
&\mathbf{B}(i,k) = -\mathbf{A}(i,k)/\mathbf{A}(k,k) && \text{for } i \neq k \\
&\mathbf{B}(k,j) = \mathbf{A}(k,j)/\mathbf{A}(k,k) && \text{for } j \neq k \\
&\mathbf{B}(i,j) = \mathbf{A}(i,j) - \mathbf{A}(i,k)\mathbf{A}(k,j)/\mathbf{A}(k,k) && \text{otherwise}
\end{aligned}$$

%SWEEP is reversible: %SWEEP(%SWEEP(A,k),k)=A.

The point of %SWEEP is the following: if **A** begins as a cross-product matrix of a set of variables, successive sweeps will do a regression. If we fix 1 as the dependent variable, after sweeping on 2 and 3, **B**(2,1) and **B**(3,1) will be the regression coefficients and **B**(1,1) will be the residual sum of squares. Note that the sweep operator does *not* produce a symmetric matrix.

The "sweep" functions are:

| | |
|---|---|
| **%SWEEP(A,k)** | Returns the result of "sweeping" **A** on pivot $k$. |
| **%SWEEPTOP(A,k)** | Returns the result of "sweeping" **A** on each of the first $k$ rows. |
| **%SWEEPLIST(A,iv)** | Returns the result of "sweeping" **A** on each of the pivots listed in the VECT[INTEGER] **IV**. |

Consider the properties of the sweep function when applied to a cross product matrix:

1. The submatrix defined by the rows and columns of the pivots is $(\mathbf{X}'\mathbf{X})^{-1}$ from a regression on those variables.
2. The submatrix defined by the non-pivot columns with the pivot rows are the projection (regression) coefficients of the non-pivots on the pivots.
3. The submatrix defined by the pivot columns with the non-pivot rows are minus the projection coefficients.
4. The submatrix defined by the non-pivots by non-pivots gives the sums of the outer product of the projection residuals. (When divided by the number of observations, this will be the estimated covariance matrix of residuals).

If we take a regression of trend on constant,

```
set trend 1 10 = t
cmom
# constant trend
disp %sweep(%cmom,1)
```

The output is:

| | | | |
|---|---|---|---|
| $\left(\mathbf{X}'\mathbf{X}\right)^{-1}$ matrix | 0.10000 | 5.50000 | (regression coeff) |
| (–regression coeff) | -5.50000 | 82.50000 | (Residual Sum of Squares) |

One of the exercises in Hayashi (2000) is to do a Monte Carlo analysis of unit root tests, doing one Dickey-Fuller test allowing for intercept and one with intercept and trend. The sweep functions offer an efficient way to do this because the two regressions can be done by sweeping first on the lag and intercept, then again on the trend.

```
cmom
# y y{1} constant trend
```

We sweep on 2 and 3 to regress on `Y{1}` and `CONSTANT`. The regression coefficients are going to be in column 1 (non-pivot column). The sum of squared residuals is in (1,1). The *t*-stat can be put together using these and the (2,2) element.

```
compute ss=%sweeplist(%cmom,||2,3||)
compute tmu=(ss(2,1)-1)/sqrt(ss(1,1)*ss(2,2)/(%nobs-2))
```

Now sweep on the trend as well and repeat

```
compute ss=%sweep(ss,4)
compute ttau=(ss(2,1)-1)/sqrt(ss(1,1)*ss(2,2)/(%nobs-3))
```

## 5.2 Working With Polynomials

RATS provides a suite of functions for manipulating polynomials. These are represented as a VECTOR of the $\|v(1),\ldots,v(n+1)\|$, which denotes the polynomial:

$$v(1)+v(2)x+\ldots+v(n+1)x^n$$

| | |
|---|---|
| **%EQNLAGPOLY(EQN,S)** | Extracts a lag polynomial from an equation. |
| **%POLYADD(P1,P2)** | Adds two polynomials. |
| **%POLYCXROOTS(P)** | Returns a VECTOR[COMPLEX] with all roots of **P**, represented as complex numbers. |
| **%POLYSUB(P1,P2)** | Subtracts two polynomials. |
| **%POLYDIV(P1,P2,D)** | Divides **P1** by **P2**, expanded out to degree *d*. |
| **%POLYMULT(P1,P2)** | Multiplies two polynomials. |
| **%POLYROOTS(P)** | Returns a VECTOR with all the real roots of **P**. |
| **%POLYVALUE(P,X)** | Returns the value of **P** at *x*. |

The most important of these is %EQNLAGPOLY(eqn,series), which pulls a lag polynomial out of an equation or regression (the most recent regression if *eqn*=0). If *series* is the dependent variable, it extracts the standard normalization for the autoregression polynomial, that is, 1–lag polynomial from the right side.

For example, this implements an ARDL (autoregressive-distributed lag) model

```
linreg(define=ardl) logc
# logc{1 to 3} logy{0 to 3} seas{0 to -3}

compute arpoly = %eqnlagpoly(0,logc)
compute dlpoly = %eqnlagpoly(0,logy)

*
* Compute the long run effect
*
compute ardllt=%polyvalue(dlpoly,1)/%polyvalue(arpoly,1)

*
* Compute 8 lags of the expansion (degree 7 polynomial)
*
compute ardlest=%polydiv(dlpoly,arpoly,7)
```

## 5.3 Date Functions . . .

The following functions can be applied after you've done a **CALENDAR**. Most take as argument an entry number (n in the descriptions) and return information about the date for that entry within the **CALENDAR** scheme that you've set. For functions with more complicated argument sets, we have used . . as an abbrevation.

See Chapter 4, the on-line help or the *Function Wizard* for details on these,

| | |
|---|---|
| %MONTH(n) | returns the month of entry n |
| %CALENDAR() | Saves the current calendar setting into a variable. For example, COMPUTE CAL = %CALENDAR(). This allows you to later recall the calendar setting using **CALENDAR** with the RECALL option. |
| %CLOSESTDATE(..) | Returns the day within a month closest to a given day of the week. Useful for identifying holidays that fall on a specific day of the week, close to a certain calendar date. |
| %CLOSESTWEEKDAY(..) | Returns the weekday closest to the date specified. |
| %DATEANDTIME() | returns a string with today's date and time for date stamping your output. |
| %DATELABEL(n) | returns the output date label, such as "2001:1:31" |
| %DAY(n) | returns the day of entry n |
| %DAYCOUNT(n) | returns the number of days in entry n. For example, %daycount(2005:2) returns 28. |
| %DOW(n) | Returns the day of the week (Monday=1 through Sunday=7) for the requested date. |
| %EASTER() | This will only work with a seven day per week **CALENDAR** setting. It returns the date of the Easter holiday in the Western Christian tradition as a number of days after March 22. |
| %FLOATINGDATE(..) | Returns the day number of the *n*th occurrence of the specified day of the week in the given month. Useful for determining the date of floating holidays defined based in this manner, such as the Thanksgiving holiday in the U.S. |
| %INDIV(n) | returns the individual's number for an entry in a panel data set. |
| %JULIAN(n) | returns the number of days from January 1, 0001 to entry n using the modern calendar. |
| %PANELOBS() | Returns the number of periods per individual for panel data. |

| | |
|---|---|
| %PANELSIZE() | returns the number of individuals in a panel data set. (This requires an **ALLOCATE** instruction or other instruction defining the workspace length). |
| %PERIOD(n) | returns the time period of entry n within a panel data set. |
| %TODAY() | returns the entry number in the current calendar which corresponds to today's date. This allows you to write a program which automatically adjusts its analysis depending upon when it's run. |
| %TRADEDAY(n,d) | returns the number of times day "d" (a numeric code from 1 for Monday through 7 for Sunday) occurs in entry n. For example, for monthly data, %tradeday(2005:2,1) returns the number of Mondays in February, 2005. |
| %WEEKDAY(n) | returns the day of the week (1 = Monday, 2=Tuesday, ...) |
| %YEAR(n) | returns the year of entry n |

# 6. Statistical Techniques

Statistical practices change over time. RATS has been around in some form for almost 30 years, beginning its life as a mainframe program on computers with less power than a modern cell phone. Techniques which were commonly employed in the 1980's often aren't even taught today, or have been replaced by modernized versions. We include here a description of some regression procedures which are no longer used enough to merit space in the main documentation, but might still be of some interest to some users. We also include a description of several less-used options on the RATS estimation routines.

## 6.1 Polynomial Distributed Lags

Polynomial distributed lags (or Almon lags or PDL's) are the principal form of "hard" restriction used for estimating distributed lags. See Section 2.8 of the *User's Guide* for a more complete discussion of the topic of distributed lags.

For the equation:

$$(1) \quad y_t = \sum_{l=L_0}^{L} \beta_l X_{t-l} + u_t$$

an unconstrained third degree PDL (lags 1 to $L$) takes the form:

$$(2) \quad \sum_{k=1}^{L} \left(a + bk + ck^2 + dk^3\right) X_{t-k}$$

where $a$, -, $c$ and $d$ are the free parameters, that is, $\beta_l$ is constrained to follow a 3rd order polynomial. If you expand this sum, $a$ is the coefficient on the sum of $X_{t-k}$, $b$ on $kX_{t-k}$, etc. We use **ENCODE** to create these four series, which we then use in the regression.

- We impose an end constraint (polynomial is zero for lag $L$+1) by writing the polynomial as $\left(k-\left(L+1\right)\right)\left(a+bk+ck^2\right)$.
- We impose near (polynomial is zero for lag –1) and far constraints using $\left(k-\left(L+1\right)\right)\left(k+1\right)\left(a+bk\right)$.

The trick in handling the constrained PDL's is to get the **EWISE** instruction, which creates the encoding matrix **R**, correct. Because lag 0 ($k$=0) corresponds to the first row in **R** (J=1 in **EWISE**), the roots should be at J=0 ($k$=–1) and J=$L$+2 ($k$=$L$+1).

As mentioned in the *User's Guide* section, you need to be very cautious in interpreting the results from a PDL, since the $t$-statistics will usually be very high. This is because they are testing whether a coefficient can be made zero, *while maintaining the hard shape constraint*.

PDL's have fallen out of favor mainly because the basic distributed lag like (1) usually leaves very high serial correlation in the residuals. If you allow for richer dynamics by adding lagged dependent variables (the ARDL model), you will generally get results that are easier to interpret.

## Example 6.1 Polynomial Distributed Lags

This example program is supplied with RATS on the file PDL.RPF. We show both the direct use of the **ENCODE** and UNRAVEL method for estimating a restricted regression, and the use of the **@PDL** procedure for doing this type of analysis.

```
open data haversample.rat
calendar(m) 1947
data(format=rats) 1947:1 2007:4 fltg ftb3
set shortrate = ftb3
set longrate  = fltg
*
* Run all estimates starting in 1951:1.
*
smpl 1951:1 *
declare rect r
*
* PDL with no end constraints
*
dim r(4,25)
ewise r(i,j)=j^(i-1)
encode(results=enc) r
# shortrate{0 to 24}
linreg(unravel) longrate
# constant enc
*
* Same estimation using the PDL procedure
*
@pdl(graph) longrate
# shortrate 0 24 3
*
* PDL with far constraint
*
dim r(3,25)
ewise r(i,j)=(j-26)*j^(i-1)
encode(results=enc) r
# shortrate{0 to 24}
linreg(unravel) longrate
# constant enc
*
* Same estimation using the PDL procedure
*
@pdl(constrain=far,graph) longrate
# shortrate 0 24 3
```

## 6.2 Ridge and Mixed Estimators

Ridge regression (Hoerl and Kennard, 1970) and mixed estimation (Theil, 1971, pp. 347-352) are related procedures which are fairly easy to implement within RATS. These are Bayesian-like estimation techniques, in the sense that there are priors which will generate them as their posterior modes. See Chapter 16 of the User's Guide (Gibbs Sampling in particular) for more precise methods of handling Bayesian estimation in RATS.

A ridge estimator of $\beta$ in $y = X\beta + u$ is defined as

$$\hat{\beta} = \left(\mathbf{X}'\mathbf{X} + k\mathbf{I}\right)^{-1}\mathbf{X}'\mathbf{y} \tag{3}$$

for some positive $k$. Least squares is the special case of $k$=0. Setting $k$>0 has the effect of "shrinking" the estimate of the coefficients toward the origin, although individual coefficients may move away from zero.

In mixed estimation, non-sample information about $\beta$ is put in the form

$$\mathbf{r} = \mathbf{R}\beta + \mathbf{v}, \quad \mathrm{Cov}\left(\mathbf{v}\right) = \mathbf{V} \tag{4}$$

The mixed estimator (assuming $\mathrm{Cov}\left(\mathbf{u}\right) = \sigma^2\mathbf{I}$) is

$$\hat{\beta} = \left(\mathbf{X}'\mathbf{X} + \sigma^2\mathbf{R}'\mathbf{V}^{-1}\mathbf{R}\right)^{-1}\left(\mathbf{X}'\mathbf{y} + \sigma^2\mathbf{R}'\mathbf{V}^{-1}\mathbf{r}\right) \tag{5}$$

The ridge estimator is a special case of this where $\mathbf{R} = \mathbf{I}$, $\mathbf{r} = 0$, and $\mathbf{V}$ is scalar.

### Computation Tools

Note that both techniques estimate $\beta$ as $\left(\mathbf{X}'\mathbf{X} + \mathbf{Q}\right)^{-1}\left(\mathbf{X}'\mathbf{y} + \mathbf{q}\right)$. The tools for implementing such estimators are the **CMOMENT** and **LINREG(CMOMENT)** instructions:

- **CMOMENT** computes the matrices $\mathbf{X}'\mathbf{X}$ and $\mathbf{X}'\mathbf{y}$ using the actual data. In practice, these will actually be segments of the same matrix. We then replace these matrices with $\mathbf{X}'\mathbf{X} + \mathbf{Q}$ and $\mathbf{X}'\mathbf{y} + \mathbf{q}$.
- **LINREG(CMOMENT)** computes the regressions using the altered versions of $\mathbf{X}'\mathbf{X}$ and $\mathbf{X}'\mathbf{y}$.

To simplify this process, on **CMOMENT** you should list the variables as regressors first, followed by the dependent variable. That way, the matrix %CMOM, as computed by **CMOMENT** will have the form

$$\begin{bmatrix} \mathbf{X}'\mathbf{X} & \mathbf{y}'\mathbf{X} \\ \mathbf{X}'\mathbf{y} & \mathbf{y}'\mathbf{y} \end{bmatrix}$$

(actually just the lower triangle of this), so $\mathbf{X}'\mathbf{X}$ is the top corner, and $\mathbf{X}'\mathbf{y}$ is all but the last element of the last row.

## Ridge Regression

Ridge regression is especially easy because all we need to do is add a constant to the diagonal elements of . The simplest way to do this is with a **DO** loop:

```
cmoment
# constant x1 x2 x3 x4 y
linreg(cmoment) y                                    Least squares
# constant x1 x2 x3 x4
do row=1,5
  compute %cmom(row,row)=%cmom(row,row)+k            Add to diagonals 1-5
end do row
linreg(cmoment) y                                    Ridge regression
# constant x1 x2 x3 x4
```

Choosing an appropriate value for $K$ is the real problem with ridge regression. See, for example, the discussion in Judge, Griffiths, Hill and Lee (1980).

## Mixed Estimation

We need to construct the matrices **R**, **r** and **V** in some fashion. **r** is often zero and **V** is almost always diagonal, and is usually a scalar matrix (a constant times the identity), which simplify these considerably.

Since we do not know $\sigma^2$, in general, we must use an estimate $s^2$ of it. We do a preliminary **LINREG** to get the estimate $s$ of $\sigma$, then use matrix commands to replace

$\mathbf{X}'\mathbf{X}$ by $\mathbf{X}'\mathbf{X}+s^2\mathbf{R}'\mathbf{V}^{-1}\mathbf{R}$, and

$\mathbf{X}'\mathbf{y}$ by $\mathbf{X}'\mathbf{y}+s^2\mathbf{R}'\mathbf{V}^{-1}\mathbf{r}$.

Where **V** is scalar (say $\lambda^2\mathbf{I}$), this can be done quite simply (if the %CMOM array has been arranged as described on the previous page) by stacking **R** on top of **r** into **Rr** and then adding to %CMOM the array $\left(s^2/\lambda^2\right)\left(\mathbf{Rr}\right)'\left(\mathbf{Rr}\right)$. This is what is done in Example 6.2. If **V** is more general, it's necessary to change the $\mathbf{X}'\mathbf{X}$ and $\mathbf{X}'\mathbf{y}$ matrices separately. This is done most easily using the **OVERLAY** instruction.

Because the general case involves a bit of extra programming, we provide with RATS a procedure named **@MIXED** (on the file MIXED.SRC) which does the entire calculation. Its syntax is

```
@mixed( options )  depvar  start  end
# explanatory variables in regression format
```

The options are

```
capr=R matrix
lowr=r matrix
scalar=scalar for V being a scalar matrix
V=V matrix in general
```

## Example 6.2 Shiller's Smoothness Prior

Shiller (1973) proposed a method for distributed lags which reduces the variability in the lags seen in least squares estimates, while imposing a less stringent shape restriction than the PDL. His smoothness prior (of degree $d$) takes the form:

$$(6) \quad (1-L)^{d+1}\beta_i = v_i$$

where $\beta$ is the vector of lag coefficients. We do this using mixed estimation (Section 5.12). Since the *dummy observations* have a fairly simple form (in particular, the **V** matrix is scalar), we will do the estimates directly.

**FMATRIX** creates these dummy observations. Note that the rows of **R** in $\mathbf{R}\beta = \mathbf{v}$ take precisely the **FMATRIX** form with the option DIFF=$d$+1. We use several values for the weight put on these.

This example uses a first degree prior ($d$=1) with lag zero of SHORTRATE left free. Thus, there are 22 rows in the **R** matrix: the 24 affected lags minus ($d$+1). **FMATRIX** starts at entry (1,3) because the CONSTANT and lag zero are the first two regressors in the cross moment matrix. Since we need the original %CMOM array each time through the **DOFOR POWER** loop, we store it in the array CMOMHOLD.

The example is supplied on the file SHILLER.RPF. See also SHILLERGIBBS.RPF, which does the same model using Gibbs sampling.

```
open data haversample.rat
calendar(m) 1947
data(format=rats) 1947:1 2007:4 fltg ftb3
set shortrate = ftb3
set longrate  = fltg
cmom
# constant shortrate{0 to 24}  longrate
linreg(cmom) longrate
# constant shortrate{0 to 24}

declare rect dummy(22,27)
declare symm cmomhold(27,27) dummyxx(27,27)
fmatrix(diff=2) dummy 1 3
compute cmomhold=%cmom
compute dummyxx =tr(dummy)*dummy

compute scalefac=%seesq/.01
dofor [real] power = 1.0 2.0 4.0 8.0
   compute kfactor=power*scalefac
   compute %cmom  =cmomhold+kfactor*dummyxx
   compute title=$
    "Shiller Smoothness Prior with k="+%strval(kfactor,"*.##")
   linreg(cmom,title=title) longrate
   # constant shortrate{0 to 24}
end dofor
```

## 6.3 Robust Estimation by Iterated Weight Least Squares

Least absolute deviations (LAD) is fairly complicated technically. An alternative way of handling this is to use a criterion function with tail behavior similar to LAD, but is more similar to least squares near zero, and thus is differentiable. For example, the following uses a convex function, twice differentiable everywhere, which is asymptotically $|u|$.

$$(7) \quad \min_{\beta} \sum_t \left(c^2 + u_t(\beta)^2\right)^{1/2}$$

$c$ is some constant. As $c \Rightarrow 0$, this approaches LAD. The first-order necessary conditions (FONC's) for solution are

$$(8) \quad \sum_t \frac{X_t' u_t}{\left(c^2 + u_t^2\right)^{1/2}} = 0$$

which are the same as the FONC's for weighted least squares with SPREAD=$\left(c^2 + u_t^2\right)^{1/2}$ We can thus estimate $\beta$ by iterated weighted least squares.

Using the results from Section 2.2 in the *User's Guide*, we can compute the covariance matrix as $\mathbf{A}^{-1}\mathbf{B}\mathbf{A}^{-1}$, where

$$(9) \quad \mathbf{B} = \sum_t X_t' X_t \frac{u_t^2}{\left(c^2 + u_t^2\right)}$$

$$(10) \quad \mathbf{A} = \sum_t X_t' X_t \frac{c^2}{\left(c^2 + u_t^2\right)^{3/2}} \qquad \text{(Note: we changed the sign on this)}$$

This is included in **ROBUST.RPF** (Example 6.3) along with LAD estimates.

## Example 6.3 Robust Estimation

```
open data zellner.prn
data(format=prn,org=columns) 1 25 valueadd capital labor nfirm

set logy = log(valueadd)
set logk = log(capital)
set logl = log(labor)
```

*Compute linear regression, and the standardized residuals.*

```
linreg logy / resids
# constant logk logl
prj(xvx=px)
set stdresids = resids/sqrt(%seesq*(1-px))
```

*Rerun the regression, omitting the outliers (defined here as observations with standardized residuals greater than 2.5 in absolute value.*

```
linreg(smpl=abs(stdresids)<=2.5) logy
# constant logk logl
```

*Now do LAD estimator.*

```
rreg logy
# constant logk logl
```

*Iterated WLS M-estimator*

```
compute c = sqrt(%seesq)
dec vector beta0
```

*Start with least squares (spread = constant). Do 10 iterations or until convergence.*

```
set spread = 1.0
do iters=1,10
   compute beta0=%beta
   set spread = sqrt(c^2+resids^2)
   linreg(noprint,spread=spread) logy / resids
   # constant logk logl
   if %testdiff(beta0,%beta)<.0001
      break
end do iters
disp "Iterations Taken" iters
```

*Compute the sandwich estimator for the covariance matrix.*

```
set f      = resids/spread
set fprime = c^2/spread^3
mcov(matrix=b,lastreg) / f
mcov(matrix=a,lastreg,nosquare) / fprime
linreg(create,lastreg,form=chisquared,covmat=%mqform(b,inv(a)),$
   title="Iterated Weighted Least Squares")
```

## 6.4 Miscellaneous Regression Topics

### The Option DFC

**dfc=***Degrees of Freedom Correction*

The DFC option is used to enter the degrees of freedom lost *in the dependent variable* in previous processing.

When you use regression residuals as the dependent variable, the standard errors of the estimated coefficients of the regression will be understated, because the first regression extracts some degrees of freedom. The *Degrees of Freedom Correction* should be equal to the number of regressors in the first regression, minus the number of those regressors which are repeated in the second regression.

```
linreg  ip / detrend
# constant  trend  seasonal{ -10 to 0 }      13 regressors
linreg(dfc=12) detrend                       dfc=13-1
# constant  intrate                          CONSTANT repeated
```

### The ENTRIES Option

**entries=***number of supplementary card entries to process* **[all]**

You can use the ENTRIES option on any instruction which has a supplementary card in regression format: **LINREG**, **CMOMENT**, **STWISE**, **PREGRESS**, **AR1**, **EQUATION** (applies to the supplementary card listing the MORE variables), **LDV**, **DDV**, **PRB**, **LGT**, **MAKE**, **ENCODE**, **MCOV**, **EXCLUDE** and **SUMMARIZE**. You can also use it for the supplementary card for **TEST**.

ENTRIES lets you put one of these instructions inside a loop and change the number of regressors on the supplementary card to use. ENTRIES is largely obsolete because you can use the **ENTER** instruction or one of the "reglist" functions to build supplementary cards in a VECTOR[INTEGER].

An *entry* is any of the objects on the supplementary card, except for the control characters # and $. You must count the { and } of a lag field. Thus M1{1 TO 10} counts as 6 entries (M1, {, 1, TO, 10, and }). RATS will use the number of entries specified by the ENTRIES option, starting from the first entry on the card.

The code below is part of a procedure which computes something similar to a vector autoregression, but with varying number of lags among the variables. NUMVAR is the number of variables in the system, VARS is a VECTOR[INTEGERS] with the list of variables, and LAGS is a VECTOR[INTEGERS] with the number of lags for each. This can handle anything up to a six variable system simply by changing NUMVAR.

```
do j=1,numvar
   linreg(entries=6*numvar+1) vars(j)
   # constant vars(1){0 to lags(1)}  vars(2){0 to lags(2)} $
      vars(3){0 to lags(3)}  vars(4){0 to lags(4)}  $
      vars(5){0 to lags(5)}  vars(6){0 to lags(6)}
end do j
```

## 6.5 Band Spectrum Regression

Another type of frequency domain regression procedure is Engle's (1974) band spectrum regression. In this procedure, the "filtering" procedure is to zero out certain bands in the Fourier transforms of the dependent variable and regressors. This is basically identical to Hannan efficient, except that a masking series is used instead of the one computed there. For instance, to include just the lowest 3/4 of frequencies, use

```
cset 1 = t<=nords*3/8.or.t>nords*5/8
```

in place of the **FFT,** **CMULT**, **WINDOW** and **CSET** instructions.

## Example 6.4 Deseasonalization Using Spectral Techniques

Sims (1974b) describes a procedure to deseasonalize data by removing all power from frequencies in a band about the seasonals. This is a frequency domain filtering procedure where the transfer function is a seasonal mask constructed using **CMASK**. The width of the seasonal bands depends upon the width of the peaks in the spectrum. The data in this example are Canadian retail sales. The example file is FREQDESEASON.RPF.

```
open data oecdsample.rat
calendar(m) 1960
```

*Data are Canadian retail sales*

```
data(format=rats) 1960:1 2007:3 canrett canretts
```

*Transform to logs and take the trend out*

```
set logret = log(canrett)
filter(type=hp) logret / removed
set prepped = logret-removed

compute nords=2*%freqsize(2007:3)
frequency 2 nords

rtoc
# prepped
#    1
fft 1
```

*Create bands of width $\pi/16$. Leave the low frequencies in.*

```
cmask 2 / (nords/12) (nords/32) (nords/12)+1
cmult 1 2
ift 1
```

*Send the deseasonalized data back, add back the removed trend and exponentiate.*

```
ctor 1960:1 %allocend()
# 1
# deseason

set deseason  = exp(deseason+removed)
```

*Graphs the deseasonalized series with the "officially" adjusted series.*

```
graph 2
# canretts
# deseason
```

# 7. Deprecated Features

This section documents several instructions that are rarely used and thus are no longer included in the *Reference Manual*. We also describe some options and parameters that are still supported by other instructions, but which have been superseded by newer parameters or (in most case) options.

## 7.1 Deprecated Options and Parameters

This section lists options and parameters which are still recognized by RATS, but have been superseded by newer features. In many cases, we've replaced parameters with options, as it is usually easier to remember the appropriate option name than to remember the proper sequence of parameters. In other cases, these have been obsoleted by entirely different (and better) ways of doing things. Unless otherwise noted, RATS will still recognize the older parameters, but we recommend that you use the new options.

### ALLOCATE

Prior to Version 6 of RATS, the **ALLOCATE** command was required in any program that used data series. It is now optional. This is no longer needed, as you can now set the default range by supplying explicit *start* and *end* dates (or entry numbers) on your first **DATA**, **SET**, or **SMPL** instruction.

### AR1

A fifth parameter used in versions prior to 7.0 for saving the coefficients has been deprecated. Coefficients are now saved automatically in the vectors %BETA and %BETASYS.

The DIFFERING option was renamed as HETEROGENOUS in Version 7. The name DIFFERING is still recognized, but not recommended.

### ERRORS

The *steps* and *VCV Matrix* parameters used in older versions have been replaced by the STEPS and CV options, respectively.

### ESMOOTH

Two parameters (*forecasts* and *steps*) used in older versions have been replaced by the FORECASTS and STEPS options described below.

### ESTIMATE

Two additional parameters (*residuals* and *coeffs*) used in older versions have been replaced by the RESIDUALS and COEFFS options.

### IMPULSE

The *steps*, *shock to* and *VCV Matrix* parameters used in older versions have been deprecated. These are replaced by the STEPS, COLUMN, and CV options, respectively.

### INCLUDE

The *comments* parameter used in older versions has been replaced by the COMMENTS option.

## KALMAN

The parameters *residual*, *coeffs*, and *printflag* used in older versions have been replaced by options. The first two parameters, for storing the residuals and coefficients, have been replaced by the RESIDS and COEFFS options (you can also use COHISTORY to save coefficients). The functionality of the third parameter, used to control printing of output based on a logical expression, has been incorporated into the PRINT option .

## MRESTRICT

Two additional parameters (*resids* and *coeffs*) used in older versions have been deprecated. RATS still recognizes these, but they are unnecessary—the residuals and coefficients are now saved automatically in the %RESIDS series and %BETA vector, respectively. You can also use the COEFF option to save the coefficients.

## MVFRACTILE

The **MVFRACTILE** instruction functionality has been incorporated into the **MVSTATS** instruction.

## RENAME

The *comments* parameter (and accompanying text cards) used in older versions has been replaced by the COMMENTS option.

## RREG

A fifth parameter called *coeffs*, which saved coefficients into a series, has been deprecated. It will still function in Version 7, but is not recommended, as the coefficients are automatically saved in the %BETA vector.

## SEASONAL

Two additional parameters (*seasonal* and *period*) used in older versions have been replaced by the SPAN and PERIOD options.

## SIMULATE

Three additional parameters used in older versions have been replaced by options. The *steps* and *start* parameters are replaced by the STEPS, FROM, and TO options. The *VCV Matrix* parameter is replaced by the CV option.

## STEPS

Two additional parameters (*steps* and *start*) used in older versions have been replaced by the STEPS, FROM, and TO options.

### STWISE

A fifth parameter, *coeffs*, still works but is not recommended. The coefficients are automatically saved in the vector %BETA.

### SUR

The current options CV and CVOUT were called ISIGMA and OUTSIGMA, respectively, in previous versions. The old names are still recognized in Version 8.

### THEIL

Two additional parameters used in older versions, *steps* and *data end*, have been replaced by the STEPS and TO options.

### X11

Two additional parameters, *adjusted* and *factors*, used in earlier versions have been replaced by options of the same names.

## 7.2 Deprecated Instructions

This section lists (in alphabetical order) some obsolete or little-used instructions that are still recognized by RATS, but are no longer documented in the *Reference Manual*.

Descriptions begin on the next page.

# CNTRL — Tools for Controlling Interactive Procedures

**CNTRL** is an old instruction used for controlling various aspects of program execution. It is largely obsolete now that RATS includes other instructions, such as **USERMENU**, **MESSAGEBOX**, and **SELECT** for controlling interactive procedures.

**cntrl ( options )** *pausemessage*

## Parameter

*pausemessage* This is a string of characters which is reproduced in the dialog box or prompt when you do **CNTRL(PAUSE)**. Note that the newer **MESSAGEBOX** and **INFOBOX** instructions are generally more useful for interactive programs.

## Options

**page/[nopage]**

**CNTRL(PAGE)** inserts a page break in the output file. You should only use this for programs you intend to run in batch mode.

**pause/[nopause]**

Pauses and waits for a user response before continuing. This will be a dialog box, or a simple message to hit a key, depending upon the environment. You can use the *pause message* parameter to add a message to the user prompt. As noted above, the **MESSAGEBOX** and **INFOBOX** instructions offer more flexible alternatives.

**user/[nouser]**

Takes input from the keyboard until the user types a **RETURN** instruction.

**interactive/[nointeractive]**

Use this at the beginning of a program which is run as a batch program but uses some interactive features.

## Notes

If you are writing an interactive procedure that is to be executed in batch mode, you will probably want to use the instruction **ENVIRONMENT NOECHO** to suppress the echoing of lines as they are processed. Let the user see only what you want her to see.

# DEFAULT — Changing Default Options

Most instruction options have a default setting, which RATS uses unless you specify otherwise. The **DEFAULT** instruction lets you change one or more of these default values temporarily. The changes remain in effect until you quit RATS, or change the defaults again with another **DEFAULT** instruction. This can be very useful if a program repeatedly uses a particular instruction with the same set of options.

```
default  instruction name(options)
```

## Parameters

The parameter field for **DEFAULT** looks like an instruction plus its option field, for instance, **GRAPH(NODATES)**. You can change more than one option for a given instruction with a single **DEFAULT**, but you need a separate **DEFAULT** for each separate instruction whose defaults you wish to change.

**DEFAULT** only affects the options you specify—other default settings for the instruction remain unchanged.

## Examples

```
default  print(nodates)
default  graph(nodates)
```

These two instructions set NODATES as the default option for **PRINT** and **GRAPH**. Subsequent **PRINT** and **GRAPH** instructions will, by default, label output by entry number rather than dates.

```
default linreg(noprint,smpl=regsampl)
```

This changes two of the default options for the **LINREG** instruction

- The default print option becomes NOPRINT, so you would need to include a PRINT option in a **LINREG** instruction to have it display the output.
- The SMPL=REGSAMPL option means that subsequent **LINREG** instructions will use REGSAMPL as the *SMPL series*.

# DELETE — Removing Series from a RATS Data File

**DELETE** removes the specified series from the currently open RATS data file (see the instruction **DEDIT**). *You must to use a* **SAVE** *instruction at some point to make these changes permanent.*

**delete** *list of data file series*, separated by blanks

## Wizard

If you open the RATS format file with *Open* on the *File* menu, the *Delete* and *Cut* operations on the *Edit* menu will delete the selected series.

## Parameters

*list of series* The list of series which you want to delete from the file.

## Example

```
dedit pricedat.rat
delete  turcpi  grccpi  prtcpi  islcpi
save
```

removes the series TURCPI, GRCCPI, PRTCPI and ISLCPI from the RATS data file PRICEDAT.RAT.

## Notes

When you do a **DELETE** instruction, it is likely that the length of your file will not change. RATS simply marks the old records as unavailable, but does not physically change the size of the file.

## See Also . . .

| | |
|---|---|
| *Intro.* Section 2.7 | RATS format data files |
| **DEDIT** | Opens (or creates) a RATS format data file for editing. |
| **STORE** | Adds data from one or more series to the RATS file you are editing |
| **INCLUDE** | Adds data from a single series to the RATS file you are editing. |

# EDIT — Screen Data Editing

**EDIT** allows you to edit a series stored on a RATS format file, or create a new series on the file, using a spreadsheet-style editing screen. Alternatively, you can use the instructions **UPDATE** and **STORE**: **UPDATE** is useful for minor changes, while **STORE** replaces the entire data series.

The **EDIT** command is only available when running RATS in interactive mode.

```
edit  series to edit
```

## Wizards

You can also edit series on a RATS file by using the *Open* or *New* operations on the *File* menu to open or create a RATS file, and double-clicking on the series you want to edit or clicking on the "new series" toolbar icon. You can edit series stored in memory (rather than on a RATS file) by doing *View–Series Window* and double-clicking on the series you want to edit.

## Parameters

*series* — The list of series names you want to edit or create. If listing multiple names, separate each by one or more blank spaces. If you don't list any series, RATS will display a scrolling list of all the series on the file. You can select a series to edit from the list, or enter a new series name to create a new series.

## Option

**modal/[nomodal]**

With MODAL, an **EDIT** editing window functions like a modal dialog box. Execution of any pending instructions is suspended until the user closes the editing window and new instructions can be executed. This can be useful when you want the program to stop and wait for the end user to enter data before continuing.

## Opening a RATS Format File

Before you can **EDIT** a series stored on a RATS format file, you need to open or create the file using the *File-Open* or *File-New—RATSData Window* operations, or the **DEDIT** or **ENVIRONMENT RATSDATA=** instructions.

## Editing a Series

Once you have opened or created a RATS data file, you can edit or create a series by issuing a command like:

```
edit usargdps
```

RATS will display a series edit window which you can use to edit the values in the series. See page Int-14 in the *Introduction* for details on using the series editing window.

## Creating a New Series

If you supply a new series name on the **EDIT** instruction, RATS will ask if you want to create the series. If you say "Yes", the program will display a dialog box allowing you to select the frequency and start date ("Undated" is one of the choices), and then display the edit window for that series.

## Saving Changes

When you are done editing, just close the editing window. You will see a dialog box asking if you want to save the changes to the series. Answer "Yes" if you want to keep the changes.

**Important Note**: You still need to save the changes to the file itself by issuing a **SAVE** instruction if you opened the file using **DEDIT**, or by using *File–Save* if you opened the file using *File–Open* or *File–New*. If you try to quit out of RATS before saving changes to the data file, the program will ask if you want to save your changes.

# INCLUDE — Adding Series to a RATS Data File

**INCLUDE** is one of three instructions that you can use to add data to a RATS format file (*User's Guide*, Section 2.5). It adds a single series to the file using data which you have already brought into RATS. The instruction **EDIT** allows you to edit or create a series using a screen editor, while **STORE** includes many series or the contents of an entire data file.

```
include   fileseries   series   start  end
```

## Wizard

You can also store series on a RATS format files using the mouse. First, open or create a RATS format file by doing *File–Open* or *File–New*. Then, use *Series Window* on the *View* menu to display a list of the series in memory. Finally, drag and drop the desired series from the series window onto the RATS data file window.

## Parameters

*fileseries* — Name you want to assign to the series when it is saved on the data file. There should not be a series with the name *fileseries* already on the file. If you wish to change an existing file series, use **STORE** or **EDIT**, or **DELETE** the series before **INCLUDE**ing.

*series* — The series (in working memory) which supplies the values for *fileseries*. Since you often will be using the same name on the data file, the default for *series* is *fileseries*. However, the names can be different.

*start end* — Starting and ending entries of the range of data you want to save. If you have not set a **SMPL**, this defaults to the defined range of *series*.

## Option

**comments=***STRING or VECT[STRINGS] containing comments* **[no comments]**

Use this option if you want to add comments describing the series. You can add up to two (80 character) comments to a series.

## Description

**INCLUDE** adds to the data file a series with name *fileseries*. RATS takes the data for this from *series* entries *start* to *end*. If you only specify the *fileseries* parameter, **INCLUDE** takes data from that series and saves it on the file under the same name. As with other editing instructions, you *must* **SAVE** the file when you are finished with all your changes to make them permanent.

## Supplying Comments

If you wish to add comments to a series, you can define the STRING or VECT[STRINGS] ahead of time, or create it directly as the argument for the option, as shown in the example below.

## Example

```
cal(q) 1980:1
all 2006:4

* Read the AUSSALES series in from a text file:
open data aussales.dat
data(format=free) / aussales

* Open an existing RATS format file:
dedit austdata.rat

* Add the AUSSALES series to the RATS file, and include
* a comment describing the series:
include(comments="Australian Sales, Total") aussales sales

* Save the changes to the RATS file and close the file
save
quit
```

# RENAME — Renaming Data File Series

Use **RENAME** to rename or change the comments for a series stored on the RATS format data file that you are currently editing. It has no effect on the data in that series. *You must do a* **SAVE** *instruction later in order to make the changes permanent.*

```
rename  old name   new name
```

## Wizard

If you open a RATS format file using *File–Open...*, you can rename a series using the *Rename* toolbar ion: 

## Parameters

*old name* Current name of the series.

*new name* New name for the series.

## Option

**comments=***STRING or VECT[STRINGS] containing comments* **[no comments]**
Use this option if you want to add comments describing the series. You can add up to two (80 character) comments to a series.

## Examples

```
dedit mydata.rat
rename  fzm1  m1
rename(comments="GDP in 1992 dollars")  gdp92  realgdp
  save
```

This renames two series on the file, and attaches the comment "GDP in 1992 Dollars" to the series REALGDP.

```
dedit prices.rat
open data prices.xlsx
store(convert=xlsx,org=cols)
rename 1234 a1234
save
```

converts an XLSX file to RATS format, then fixes an illegal name (RATS variable names can't start with a digit).

# UPDATE — Altering Data on a RATS Data File

**UPDATE** adds data to or replaces data on a RATS data file series. It is a "batch" style alternative to using the full-screen editor invoked by **EDIT**, or the *File–Open* operation. You must open the file for editing using **DEDIT** before using **UPDATE**. Any changes you make only become permanent when you execute a **SAVE**.

**update** *file series* *begin entry*(expanded) *end entry*(expanded)
< data cards >

## Parameters

| | |
|---|---|
| *file series* | Name of the series (on the RATS data file) you want to update. |
| *begin entry* | This is the date or entry number of the first entry you want to change or add. You can use date fields, but note that these are for the calendar setting of the series on the file. By default, it is the current end of the *file series* + 1. |
| *end entry* | The date or entry number of the end of the updated section. If you omit this, **UPDATE** will add or alter one or more values starting with *begin entry*. |

## Description

**UPDATE** reads data for the update from data cards following the instruction. Data can be separated by blanks, tabs or commas. If you give explicit values for both the *begin entry* and *end entry* parameters it will replace or add data for just those entries. If you omit *end entry* or both, **UPDATE** will change or add as many entries as it processes from a *single* line of data. With panel and intra-day data, you can only add entries to the end of the series with **UPDATE**. Use **EDIT**, **SET**, or **COMPUTE** to modify existing values of a panel data series.

You do *not* need to set up a full RATS program (with **CALENDAR** and **ALLOCATE**) to use **UPDATE**, as it takes all its information from the data file. You can update a file simply with **DEDIT**, one or more **UPDATE**s, **SAVE** and **END**.

## Example

```
dedit mydata.rat
update  realgdp
  1604.3 1611.4 1625.2 1644.7
update sales 2006:3
  11234
save
```

adds four entries to the end of REALGDP and replaces the 2006:3 value of SALES.

# 8. RATS Error Messages

This section lists the various error and warning messages that rats can display. In many cases, we've included some comments that may help you resolve the issue. When faced with an error or warning, be sure to take a moment to consider carefully what the message says. While some causes are hard to identify properly, in many cases the error message will tell you *exactly* what you need to know to resolve the problem.

# 8.1 RATS Error Messages and Troubleshooting

This appendix lists all of the error and warning messages that RATS can produce. Many of these are largely self-explanatory. However, for some of the more common errors, especially those that can have fairly obscure causes, we have tried to provide suggestions to help you resolve the problem.

The messages are grouped by type. Each error has a unique error code consisting of 1 to 3 letters followed by a number, indicating the general type of error. For example, basic syntax errors begin with the letters SX, while errors related to Vector Autoregressions begin with the VAR.

Many of the messages will include detailed information about the specific cause of that error. For example, if you try to multiply a 3x3 matrix with a 4x4 matrix, RATS would produce the error:

```
## MAT2. Matrices with Dimensions 3 x 3 and 4 x 4 Involved in * Operation
```

In the listing below, information that varies depending on the context of the error (such as the specific dimensions involved in the MAT2 error show above) are enclosed in << and >>. Where possible, we indicate the type of information that RATS will display in an actual error message. For example, <<*name*>> means that RATS will display the name of a variable, instruction, option, etc., while <<data type>> indicates that it will display a data type, such as SERIES or REAL.

Some of the error messages end with a question, such as "Did you forget to SOURCE?". Note that these only suggest a *possible* cause for the error.

Finally, note that a number of these error messages are obsolete, and thus will never be generated by Version 7 or later of RATS. We include them here only because they are still present in the error message file used by RATS (RATSERRS.MSG).

## General Syntax Errors

**SX1** **Identifier <<*name*>> is Already in use as a(n) <<*data type*>>**
You've tried to re-declare an existing variable as a different type. Use a different name for the variable, or clear the memory and re-start.

**SX2** **Expected << " *or* ' >> to Terminate String**

**SX3** **Expected a Label/Identifier**

**SX4** **Expected Blank or Tab Here**

**SX5** **Expected = Here**

**SX6** **Expected ) Here**

**SX7** **( Must Follow Immediately After Function Name**
Don't put a blank space between a function name and the "(". For example, use %ran(1.0), not %ran (1.0).

**SX8** **<<*data type*>> is not an Aggregate Data Type**

**SX9** **<<*data type*>> is not a Basic Data Type**

**SX10** **] Expected After Type Declaration**

**SX11** **Identifier <<*name*>> is Not Recognizable. Incorrect Option Field or Parameter Order?**

Don't put a space between an instruction name and an option field. For example: use "**print(nodates)**", not "**print (nodates)**". Check for misspelled names.

**SX12 Expected a Variable Name Here**

**SX13 "<<*name*>>" Is Not a Valid RATS Variable Name**

See "Symbolic Names" in Section 1.9 of the *User's Guide* for details on what RATS accepts as valid variable names.

**SX14 Ill-formatted number**

**SX15 Trying to Store into Constant or Expression. Called Parameter by Value?**

If a **PROCEDURE** parameter is called by value, you cannot change the value stored in that variable. If necessary, call the parameter by address instead.

**SX16 Missing Operand or Adjacent Operators**

Be sure to put at least one space *before* the equal sign in a **SET** instruction. For example, use "**set x =t**" and not "**set x=t**".

**SX17 Missing Operator or Adjacent Operands**

The SX17 can occur if you refer to "N(entry)" where N is a series number —use "([series] N)(entry)" instead. In **SET** or **FRML** expressions, you can use lag notation: "N{0}"

**SX18 Dates Must be a:b or a:b:c**

Don't forget to separate the year and period (or the year, month, and day) in a date expression with colons, such as: 1999:3:1.

**SX19 This Character is Illegal Here**

**SX20 Expected <<*character*>> Here**

**SX21 A <<*character*>> Here is Unneeded or Unexpected**

**SX22 Expected Type <<*data type*>>, Got <<*data type*>> Instead**

An expression may be returning a different data type than the declared type of the variable you are setting, or you are using a variable of a type other than that expected by the instruction (such as using a REAL variable as the *series* parameter on **SET**).

**SX23 Expected Variable/Element of Type <<*type*>>, Got Expression Instead**

**SX24 Expected Dimension Field Here**

When using **DIMENSION**, don't put a space between a variable name and the left parentheses. For example, use "**dim xmat(10,10)**", not "**dim xmat (10,10)**".

**SX25 Logical Operators are .EQ./.NE./.LE./.LT./.GE./.GT./.AND./.OR./.NOT.**

**SX26 Matrix Constructed with ||...|| has Too Complicated a Type**

**SX27 Illegal Combination of Data Types for Operation**

**SX28 <<*{ or }*>> For Lags Used In Improper Manner**

**SX29 Need a } to End the Lag Field**

**SX30 Characters After End of Expression**

**SX31 Expression to Left of , Has No Effect (year,period is obsolete notation)**

**SX32 Expected */ To End Block Comment**

**SX33 Expression is Too Complicated. Break into Parts?**

## Errors Related to Instruction Names or Usage

**I1** **Expected Instruction - <<*string*>> Is Not Recognizable As One**

**I2** **Expected Instruction Here**

The most common cause of these two errors are mis-spelled instruction names. If you are doing an **INPUT**, **READ**, or **DATA(UNIT=INPUT)** instruction, you may also have provided more data than the input instruction is expecting, causing RATS to try to interpret a line of data as a new instruction.

**I3** **Instruction Format Should Be <<*instruction format*>>**

**I4** **<<*instruction*>> Requires RATS Pro Version**

**I5** **RATS Linear Doesn't Include Instruction <<*instruction*>>**

**I6** **<<*instruction*>> Required X Windows Version**

## Errors Involving Instruction Parameters

**P1** **/ For Parameter Pairs Should Be Followed by Space**

If using the "/" symbol for the default range, make sure you have a space before and after the "/".

**P2** **Second Half of Parameter Pair Is Absent**

If you supply a value for a *start* parameter, you must also supply a value for the *end* parameter (use "*" for either parameter to use the default value for that parameter).

**P3** **Instruction <<*instruction*>> Requires Parameter <<*parameter*>>**

**P4** **<<*parameter string*>> is Invalid - Choose From <<*list of choices*>>**

**P5** **<<*parameter choice*>> Cannot Be Used with <<*parameter choice*>>**

## Errors Related to Options

**OP1** **Looking For Option Field or White Space. This Character Is Unexpected.**

**OP2** **Expected , to Separate Options or ) to End Option Field**

Use a comma to separate each option in an option field, and close the option field with a right parenthesis.

**OP3** **This Instruction Does Not Have An Option <<*option*>>**

**OP4** **The Option <<*option*>> Is Already Set By This Instruction**

You have probably listed the same option twice.

**OP5** **Form NO<<*option*>> is Not Permitted For Option <<*option*>>**

**OP6** **Expected Option Here**

**OP7** **<<*argument*>> Is Not One of the Choices for Option <<*option*>>**

**OP8** **Option <<*option*>> Needs = Here**

**OP9** **Options <<*option*>> and <<*option*>> May Not Be Used Together**

**OP10** **Instruction <<*instruction*>> Requires Option <<*option*>>**

If you have problems with an option field, carefully check the syntax and the list of options for that instruction in the on-line help or the *Reference Manual*.

## Errors Involving Supplementary Cards

**SC1** **Expected Supplementary Card (# ....) Here**

**SC2** **Require Supplementary Card Field <<*name*>> (Position <<*number*>>)**

**SC3** **Expected <<*number*>> Entries on Supplementary Card, Got Only <<*number*>>**

**SC4** **Can't Use / or * On This Type of Supplementary Card**
**SC5** **Use of * Illegal At This Location**
**SC6** **Need a LIST Instruction to Use CARDS**
**SC7** **LIST Needs At Least <<*number*>> Entries, Had Only <<*number*>>**

## Errors Involving Data Series

**SR1** **ALLOCATE Instruction Needed Before Series or Equations Can Be Used**
**SR2** **FREQUENCY Instruction Needed Before Complex Series Can Be Used**
You must do an **ALLOCATE** instruction before defining or using real-valued series, and you must use do a **FREQUENCY** instruction before defining or using complex-valued series. If you are using procedures that work with series, you may need to do your **ALLOCATE** instruction *before* **SOURCE**ing in the procedure.

**SR3** **Tried to Use Series Number <<*number*>>, Only <<*number*>> Are Available**
**SR4** **Tried To Use Series Number <<*number*>> (-series n1 n2 triples are no longer legal)**
**SR5** **<<*writing/using*>> Range <<*entry*>> to <<*entry*>> of Series <<*series*>> (Incorrect start,end or SMPL)**
**SR6** **Missing a Necessary Parameter. Check Instruction Syntax**
**SR7** **Block with <<*number*>> Elements is Too Big for This Version of RATS**
**SR8** **Badly Formed TO triple**
**SR9** **High TO Low Range: <<*entry*>> TO <<*entry*>> Is Illegal**
**SR10** **Missing Values And/Or SMPL Options Leave No Usable Data Points**
This can occur if you try to apply an operation to a series that contains no data, or try to run an estimation where all observations are dropped because of missing values in one or more series. Check all of the series involved using **TABLE** or **PRINT**. Watch for transformations, such as taking logs or roots of negative numbers, that can eliminate data. Make sure you are not trying to start an estimation too early in the sample range. For non-linear estimation, check for valid initial conditions for all parameters, make sure you aren't taking logs or roots of negative numbers, etc.

**SR11** **Illegal range, start = <<*entry*>> , end = <<*entry*>>. Are your parameters correct?**
**SR13** **Need to Include an Entry Range. No Previous Instruction Has Set the Default.**
If you don't use an **ALLOCATE** (which is now optional, rather than required), you will need to specify an explicit ending period on the first instruction in your program that uses series (usually a **DATA** instruction). Otherwise, you will get this error.

## Regression/Estimation Problems

**REG1** **Cannot Execute Unless Preceded by a Completed Regression**
**REG2** **LINREG(CMOM) First Requires CMOM/MCOV without CORR, CENTERED or MATRIX option**
**REG3** **<<*instruction*>> Fails - No Prior Regression or Regression Was Different Size**
**REG4** **Variable <<*name*>> is Not in the Prior Regression**
**REG5** **Variable <<*name*>> is Not in the Prior CMOMENT**
**REG6** **Matrix <<*name*>> Is Singular. Check for Collinearity in First <<*number*>> <<*rows or columns*>>**

**REG7** **Variance is Zero**

**REG8** **No Instruments Available. Use an INSTRUMENT instruction first**

**REG9** **Model is Underidentified - <<*number*>> Parameters To Estimate and Only <<*number*>> Instruments**

**REG10** **First <<*number*>> Instruments Are Linearly Dependent Over Regression Range**

**REG11** **Regression Position <<*number*>> Should Be Between 1 and <<*number*>>**

**REG12** **SIGMA Is Singular/Not PSD At Row <<*number*>>. Too Many Equations for Data Set Size?**

For SUR estimation, you *must* have more time periods per series than equation.

**REG13** **Singular Regressions - Check for Collinearity among Rows 1 to <<*number*>>.**

**REG14** **Unsatisfactory Initial Estimates. Setting Last MA to 0.**

**REG15** **Series <<*name*>> Has All <<*ZERO or NONZERO*>> Values Over Estimation Range. Are You Using Zero-NonZero Coding?**

**REG16** **Initial regression uses all the degrees of freedom**

**REG17** **Either use series list parameters or SCRATCH option.**

**REG18** **METHOD=CORC/MAXL Cannot Be Used With Gaps/Missing Values Switching to HILU/SEARCH**

This is just a warning that RATS is switching to a different estimation method for the **AR1** instruction due to missing values in the sample range.

**REG19** **Need at least <<*number*>> Autocovariances. Adding 0's at end of COVARIANCES series**

**REG20** **ITERATE/BOXJENK Cannot Be Used with Gaps/Missing Values**

ARIMA estimation methods cannot be applied if there are any missing values in the estimation range. If you didn't expect to have missing values, check the start and end dates used for the regression, and use **PRINT** to check the data. You can use **SAMPLE** to "compress out" the missing data if desired.

**REG21** **Can't Freely Estimate SW Matrix with <<*number*>> Conditions. Have Only <<*number*>> Observations**

## Errors Involving Matrices

**MAT1** **Matrix <<*name*>> Has Not Been Dimensioned**

You must dimension an array before you can set or reference individual elements.

**MAT2** **Matrices with Dimensions <<*rows x columns*>> and <<*rows x columns*>> Involved in <<*operator*>> Operation**

**MAT3** **Matrix with Dimensions <<*rows x columns*>> Involved in <<*operator*>> Operation. Need <<*rows x columns*>>**

**MAT4** **Must Use the Form matrix=<<*function*>>(...) for this Operation.**

The function returns an array as a result, so an array name must be provided on the left side of the equal sign.

**MAT5** **Needed Matrix with Dimensions <<*rows x columns*>>, Got <<*rows x columns*>> Instead**

**MAT6** **Trying to Store <<*rows x columns*>> Matrix Into VECTOR**

**MAT7** **Function <<*function name*>> Requires a <<*matrix type*>> Matrix As Argument**

**MAT8** **Non-Positive Argument for function <<*function name*>>**

**MAT9** **Base Address for OVERLAY is Invalid (Subscripts Out of Range)**

**MAT10 You Can't OVERLAY with a Series, Only with an Array**

**MAT11 You Can't DIMENSION a Series**

**MAT12 Syntax: EWISE array(I,J)=expression or array(I)=expression**

**MAT13 Store into Out-of-Range Matrix or Series Element**
This can occur if, for example, you try to set element 11 of a matrix that has only been dimensioned to have 10 elements. If you are doing a non-linear estimation using recursive **FRML**s, as when estimating GARCH models with **MAXIMIZE**, where series are being used to hold values of sub-formulas—those series must be initialized over the entire estimation range. Set them to a value over the full range.

**MAT14 Non-invertible Matrix. Using Generalized Inverse for SYMMETRIC.**
RATS tried to invert a matrix, but the matrix was non-invertible. If the matrix was of type SYMMETRIC, RATS has applied a generalized inverse routine.

**MAT15 Subscripts Too Large or Non-Positive**
Array elements are numbered from 1 to the dimensioned size. Don't use zero or a negative number, or a number larger than the dimensioned size, in referring to an array element. If this appears as a result of a **MAXIMIZE** command, you most likely have specified the entry range incorrectly, or else one of your formulas is trying to store a result into a series or array element that has not been defined yet—make sure that any series used to hold residuals or variances have been explicitly defined over the entire estimation range.

**MAT16 Column Choice of %ld is Invalid**

**MAT17 Can't Use %s Range of %ld to %ld in %s operation**

## Errors Relating to Equations

**EQ1 Cannot Use Equation Number <<*number*>> (Value should be Positive)**

**EQ2 Equation Includes More Than One Explanatory Variable**

**EQ3 Equation <<*name or number*>> Has Not Been Defined**

**EQ4 Equation <<*name or number*>> Has At Least One Undefined or NA Coefficient**

**EQ5 Equation <<*name or number*>> Does Not Include Variable <<*name*>>**

**EQ6 Variable <<*name*>> Is Not the Dependent Variable of Equation <<*name*>>**

**EQ7 Variable <<*name*>> Has Zero Coefficient in Equation <<*name or number*>>**

**EQ8 Cannot Replace the Dependent Variable with a Lag**

**EQ9 ALLOCATE Equation <<*number*>> is Increased in Size. %COEFF is No Longer Valid**

**EQ10 Equation <<*name* or number>> Needs Residuals for This Operation**

**EQ11 Need To Use MODIFY Instruction to Start Equation Modification**

## Non-Linear Estimation Problems

**NL1 FRML References More Than 20 Deep. Do You Have a Self-referencing FRML?**

**NL2 NONLIN Instruction is Required Before Non-Linear Estimation Can Be Done**

**NL3 FRML <<*name*>> Has Not Been Defined**

**NL4 FRMLs Cannot Have Moving Average Parameters**

**NL5 FRML <<*name*>> was Defined Without a Dependent Variable**

**NL6** **NONLIN Parameter <<*name*>> Has Not Been Initialized. Trying 0**
This is just a warning that you have not provided an initial value for the specified parameter. RATS will try to do the estimation using 0.0 as the initial value. If the estimation fails to converge, use **COMPUTE** to supply better initial values before doing the estimation. Also, note that 0.0 may not be a valid initial value for some parameters (if the formula takes the log of the parameter, for example).

**NL7** **NONLIN Parameter Has Invalid Form (Common errors: using = > or < for == >= or <=)**

**NL9.** **PARMSET is Size <<*size*>>. Doesn't Match Last Estimation of Size <<*name*>>**

**NL10.** **Analytical Derivatives Can't Be Applied to Formula**

## Errors Involving Vector Autoregressions

**VAR1** **A SYSTEM instruction is needed first**

**VAR2** **Need SYSTEM and VARIABLES instructions first**

**VAR3** **<<*name*>> requires either SYSTEM(KALMAN) or KFSET instruction**

**VAR4** **SYSTEM definition is incomplete**

**VAR5** **KFSET Lists <<*number*>> Arrays, TVARYING Lists <<*number*>>. They should be equal**

**VAR6** **<<*name*>> Instruction Lists <<*number*>> Arrays, Should Have <<*number*>>**

**VAR7** **Need ESTIMATE or KALMAN(STARTUP=xxx) First**
You must either **ESTIMATE** the model or do **KALMAN(STARTUP=date)** to initialize the Kalman Filter before using **KALMAN** to do updating.

**VAR8** **SYSTEM Lists <<*number*>> Equations, VARIABLES Lists <<*number*>> Variables. They should be equal**

**VAR9** **You Can't ADD and DROP on the same KALMAN instruction**

**VAR10** **Can't DROP Entry <<*number*>> - Filter Only Includes Entries <<*number*>> to <<*number*>>**

**VAR11** **COHISTORY can only be used with a single equation model**

## Compiled Mode Problems

**CP1** **END(}) is Improperly Placed. Match with BEGIN({),DO,DOFOR,LOOP,PROCEDURE,MENU,FIND**

**CP2** **This Instruction Must Precede All Executable Instructions**

**CP3** **Instruction is Illegal Outside Compiled Section**

**CP4** **ELSE Must Come Immediately After IF block**
If you are using an **IF**–**ELSE** structure and want to execute several statements when the **IF** command is true, you must enclose those statements in { and }.

**CP5** **CHOICE Must Come Immediately After MENU or Another CHOICE block**

**CP6** **Need CHOICE instruction to follow MENU**

**CP7** **No DO, DOFOR, LOOP, WHILE, UNTIL or FIND for BREAK/NEXT**

**CP8** **Label <<*name*>> Is Undefined**

**CP9** **Label <<*name*>> Is Already In Use**

**CP10** **Statement Label is Illegal Here**

**CP11** **Statement Label <<*name*>> is Unreachable From BRANCH**

**CP12** **DO Loop Has Increment of 0**

**CP13** **Switch and Choice Options Cannot be Done By Reference**

**CP14** **Please Do Separate DIMENSION Instruction in Compiled Section**
Inside a procedure or loop, you cannot include dimension fields on a **DECLARE** instruction—use a separate **DIMENSION** instruction in those cases.

**CP15** **This Supplementary/Data/Text/CARDS card is not Used**

**CP16** **Fatal Error - RATS Program Bug in Executing Compile**

**CP17** **PROCEDURE Must be Initial Statement in a Compiled Section**

**CP18** **<<*name*>> is not the Name of a PROCEDURE (Did you forget to SOURCE?)**

**CP19** **<<*name*>> is not a PROCEDURE Parameter**

**CP20** **<<*name*>> may be used only in a PROCEDURE**

**CP21** **SOURCE Instructions/Executes Nested Too Deeply. Recursive Procedure?**

**CP22** **<<*name*>> may not be used in a compiled section**

**CP23** **Option abbrevation <<*abbrev*>> already in use in procedure <<*name*>>**

## Miscellaneous Errors Relating to Specific Instructions

**X1** **Syntax is CALENDAR(options) year month day with this frequency**

**X2** **Don't Use CALENDAR 1 1 1 for Undated Data. See page 1-5 in the Manual**

**X3** **You Probably Mean <<*number*>>:1. Note that year:1 is needed with Annual Data**
Don't use just a year number, such as "`1999`", as a date reference when working with annual data. Use "`1999:1`" instead.

**X4** **<<string>> is illegal picture. Pictures must look like *, ##, *.## or ##.##**
See **DISPLAY** for details on picture codes.

**X5** **ENTER(VARYING) Requires a Single VECTOR[...] type Variable**

**X6** **You cannot ENTER an Array with more than one object on the list**

**X7** **TREND=EXPONENTIAL/SEASONAL=MULTIPLICATIVE Cannot Be Used with Non-Positive Data**

**X8** **Unstable ESMOOTH Parameters. Try Different Values.**

**X9** **You should specify an explicit seasonal with this frequency data**

**X10** **Use the TABLE Instruction to Get Statistics on Several Series**

**X11** **Series Lists Have <<*number*>> and <<*number*>> Variables. They Should be Same Size**

**X12** **Cannot Do COMPRESS with This Combination of Weights**

**X13** **Redundant Restrictions. Using <<*number*>> Degrees, not <<*number*>>**
See Section 6.2 of the *User's Guide*.

**X14** **Sorry, QUERY Can't Handle Arrays**

**X15** **Expected <<*number*>> More Data Values**

**X16** **MEDIT only accepts REAL Valued Arrays**

**X17** **Problem and network do not match**
The number of input and/or output series, hidden nodes, or `DIRECT` option setting do not match those used to generate the *`memory vector`* for this neural network.

**X18** **Input pattern has different range than test data**
The range of values in the input series exceeds the maximum or minimum values specified when the network was first estimated.

**X19** Unable to open X Window for display

**X20** This Item Type Needs an INTEGER for "output value"

**X21** You Need a SERIES, LIST, STRINGS, or REGRESSORS Option to Define the List

**X22** DISTRIB=GED Can't Be Used with Multivariate Model. Switching to Normal

**X23** FILTER(TYPE=<<type>>) Needs <<value>> for WIDTH/SPAN option

## Forecasting/Simulation Errors

**FO1** Equations <<*name or number*>> and <<*name or number*>> both have dependent variable <<*name*>>

Each equation in the system must have a unique dependent variable. See Section 11.2 in the *User's Guide*.

**FO2** Need the "equation" parameter on each supplementary card

**FO3** You need to do THEIL(SETUP) first

**FO4** Identities must come last

**FO5** SHOCKS array has <<*number*>> columns. Should have <<*number*>>

**FO6** SIMULATE(SETUP) is no longer needed. See description of SIMULATE.

**FO7** Gauss-Seidel solution is explosive. Check model. Try DAMP option.

**FO8** Gauss-Seidel did not converge in <<*number*>> Iterations. Criterion = <<*value*>>.

See Section 11.2 in the *User's Guide* for tips on getting your simultaneous equation model to converge.

**FO9** Simultaneous block is singular. Check your equations.

**FO10** Either actual or forecast is NA. Skipping step(s).

**FO11** IMPULSE, ERRORS and HISTORY can't handle model with FRMLs.

## Memory/Internal Errors

**M1** <<*parameter or option*>> = <<*number*>> is Illegal. Value should be between <<*number*>> and <<*number*>>

**M2** <<*parameter or option*>> = <<*number*>> is Illegal. Value should be <<*required value type*>>

**M3** RATS Internal Error!!! This should never happen

**M4** A memory request for an additional <<*number*>> bytes cannot be satisfied

**M5** <<*name*>> has Value <<*number*>> , Must Be <<*number*>>

**M6** User Abort

**M7** Untrapped Floating Point Error: Overflow or NA

**M8** Untrapped Miscellaneous Error

**M9** RATS Internal Error!!! Save Your Work Now Using Save As

**M10** Classroom RATS Limit of <<*number*>> Data Points Exceeded. You Need <<*number*>>

**M11** Integer divide by zero

**M12** Polynomials can't have negative powers

**M13** Table Must be 2xN RECT[INTEGER]

## Dynamic Linear Modeling Errors

**DLM1** Rank of Prediction Error Variance < Number of Observables

**DLM2** No Observations Produce Valid Output. Check Data and Initial Values

**DLM3** The G Option Requires Both A and SW to Solve Out Initial Conditions

**DLM4** With TYPE=CONTROL, B Option is Required

**DLM5** Probable Model Error. Diffuse prior was not reduced to zero rank

## Linear/Quadratic Programming Errors

**LPQ1** Contradictory constraints

**LPQ2** Unbounded problem

**LPQ3** General failure of LPQP

**LPQ4** Simplex method does not converge

**LPQ5** Active set method does not converge

**LPQ6** All elements of "b" vector must be non-negative

## Input/Output Errors

**IO1** Unit "<<*I/O unit name*>> " is already open

**IO2** Unit <<*I/O unit name*>> Is Already In Use For <<*access type*>>

**IO3** Unable to open <<*file name*>>

Check to make sure the file isn't already open in another application.

**IO4** Unit "<<*I/O unit name*>>" does not exist

**IO5** Unit "<<*I/O unit name*>>" is permanent. You cannot <<*OPEN or CLOSE*>> it

**IO6** Units <<*I/O unit name*>>" and "<<*I/O unit name*>>" are already aliases

**IO7** Unit <<*I/O unit name*>> is not open

**IO8** An error occurred while writing to the disk, the disk may be full.

**IO9** Invalid input "<<*string*>>" on line <<*number*>> while processing series <<*name*>> entry <<*number*>>

Usually occurs when a file being read with **DATA** includes text information. This is only a warning—RATS will set the indicated entry to NA and continue reading data. See the end of Section 2.8 in the *User's Guide*.

**IO10** Unexpected end of file while processing line <<*number*>>. (series <<*name*>> entry <<*number*>>)

**IO11** Input/Output error on <<*operation*>> <<*error code*>>

**IO12** A date is missing from series <<*name*>> after entry <<*number*>>

**IO13** "<<*string*>>" is not a valid date on line <<*number*>>

**IO14** Data are not in ascending date sequence

**IO15** Using dates from the data file

**IO16** Input file does not contain dates

**IO17** Header names <<*number*>> series while there are <<*number*>> series on file

The file should have one series name per column or row of data. This indicates that there are too few or too many series names.

**IO18** Unexpected end of line while processing line <<*number*>> (series <<*name*>> entry <<*number*>>)

**IO19** The file does not have a valid <<*format type*>> format

**IO20** The file contains no valid data

**IO21** **RATS only process the first worksheet in a spreadsheet file**

**IO22** **Format has nothing to read or write**

**IO23** **Format <<*character*>> and element <<*type*>> are of incompatible types**

**IO24** **FORTRAN Hollerith Field Length of <<*number*>> is Illegal/Clearly Wrong**

**IO25** **Illegal FORTRAN format "<<*format string*>>"**

**IO26** **Unbalanced parenthesis in FORTRAN format**

See Section 2.9 in the *User's Guide* on using FORTRAN formatting codes.

**IO27** **The file is not in RATS format.**

**IO28** **File appears to be damaged**

**IO29** **No file is currently being edited**

**IO30** **There is no series <<*name*>> on the file**

Check the list of series on the **DATA** instruction versus the list on the file. Or, omit the list of series and add the VERBOSE option on **DATA**—this will read in *all* series on the file and report their names.

**IO31** **Series <<*number*>> already exists on the file**

**IO32** **At most two comments can be attached to a series**

**IO33** **The file being edited is not version 4**

**IO34** **Date Type is Not Supported by RATS Version 3 Format**

**IO35** **Unable to rename file <<*name*>> to <<*name*>>**

**IO36** **Unable to delete file <<*name*>>**

**IO37** **Cannot use UNIT=INPUT with this format**

**IO38** **Cannot convert between these Frequencies**

**IO39** **Unexpected End of File. Only <<*number*>> Values Were Processed**

**IO40** **Expected <<*type*>>. Got "<<*string*>>".**

**IO41** **"<<*name*>>" is not the Name of Series in Working Memory**

**IO42** **Binary file contents do not match READ requests**

**IO43** **Format <<*format*>> is illegal or not supported**

**IO44** **Date scheme <<*string*>> is not currently supported on this file type**

**IO45** **Data can not be appended to unit <<*I/O unit name*>>**

**IO46** **Separate year and period or year month day on UPDATE**

**IO47** **<<*format*>> cannot process a <<*string*>>**

**IO48.** **File with <<*number*>> columns might not be readable by other programs**

## FAME Format Data Errors

**FM1** **FAME HLI error <<*format*>>**

## X11 Seasonal Adjust Errors (Professional Version Only)

**X1101** **X11 can take monthly or quarterly series only.**

**X1102** **Series with missing values cannot be used in X11.**

**X1103** **Series has non-positive values. Switching to additive adjustment.**

# 9. Manual Errata

This section lists errors and omissions in the original printed copies of the version 9.0 editions of the RATS documentation. In many cases, these issues have been corrected in the revised PDF versions of the manuals provided with current versions of the software.

At this point, none.

# Bibliography


Engle, R. (1974). "Band Spectrum Regression." *International Economic Review*, Vol. 15, pp. 1-11.

Hayashi, F. (2000). *Econometrics*. Princeton, New Jersey: Princeton University Press.

Hoerl, A.E. and R.W. Kennard (1970). "Ridge Regression: Biased Estimation for Non-Orthogonal Problems." *Technometrics*, Vol. 12, pp. 55-82.

Judge, G.G., W.E. Griffiths, R.C. Hill and T-C. Lee (1980). *The Theory and Practice of Econometrics*. New York: Wiley.

Shiller, R.J. (1973). "A Distributed Lag Estimator Derived from Smoothness Priors." *Econometrica*, Vol. 41, pp. 775-788.

Sims, C.A. (1974). "Seasonality in Regression." *Journal of the American Statistical Association*, Vol. 69, pp. 618-626.

Theil, H. (1971). *Principles of Econometrics*. New York: Wiley.

# Index

## U



## V



## W



## X