Preface

Section 1 of the Reference Manual documents all of the instructions available in RATS, in alphabetical order. In a change from previous editions, the instructions used in frequency domain analysis have now been included with the other instructions in Section 1, rather than being listed separately.

Section 2 documents the built-in functions. This is followed by several appendices with additional documentation, including details on the RATS syntax, reserved variables and error messages.

The point of the Reference Manual is to describe the syntax, algorithms and output. Most instructions will include at least one or two short examples of their use. We have included few full running examples here, preferring to leave those to the User’s Guide. As might be expected from the name, we expect that this manual will be used mainly for quick checks, while the User’s Guide will be used to learn techniques.

A list of the changes and new features in Version 7 is included at the beginning of the User’s Guide.
# Table of Contents

Preface ........................................................................................................................ iii

**Section 1: The RATS Instructions** 1

ACCUMULATE — Computing Partial Sums ................................................................. 2
ALLOCATE — Preparing for Work with Data Series ................................................... 4
AR1 — Regression with Autocorrelation Correction .................................................. 6
ASSOCIATE: Assigning Coefficients to Equations ...................................................... 12
BOOT — Randomization for Bootstrapping ............................................................... 15
BOXJENK — ARIMA, Transfer Function, Intervention Models .............................. 17
BRANCH or GOTO — Branching to a Specific Instruction ......................................... 24
BREAK — Breaking Control Out of Loops ................................................................. 25
CACCUMULATE: Partial Sums of Complex Series ................................................. 26
CALENDAR — Setting the Date Information ............................................................. 27
CATALOG — Listing the Contents of a RATS Data File ........................................... 35
CDF — Computing Marginal Significance Levels ....................................................... 36
CHANGE — Switching I/O Units .............................................................................. 37
CLABELS: Labeling Complex Series ......................................................................... 39
CLEAR — Clearing a Data Series .......................................................................... 40
CLN, CEXP and Related Instructions ...................................................................... 41
CLOSE — Closing an Open I/O Unit ...................................................................... 42
CMASK: Creating Seasonal Masks ......................................................................... 43
CMOMENT — Cross-Moment and Correlation Matrices .......................................... 44
CMULTIPLY and related instructions ...................................................................... 48
CNTRL — Tools for Controlling Interactive Procedures ........................................ 49
COMPUTE — Evaluating Scalar and Matrix Expressions ....................................... 50
COPY — Flexible Data Output ............................................................................... 54
CORRELATE — Autocorrelation and Related Functions ......................................... 58
CPRINT: Printing Complex Series ......................................................................... 64
CROSS — Cross-Correlations .............................................................................. 66
CSAMPLE: Reducing Frequencies ......................................................................... 70
CSET: General Transformations .......................................................................... 71
CTOR: Data From Complex to Real ...................................................................... 73
CVMODEL — Covariance Matrix Modelling ......................................................... 75
CXTRENUM: Extreme Values for Complex Series ................................................... 79
DATA — Reading Data Series ............................................................................... 80
DBOX — Creating User-Defined Dialog Boxes ....................................................... 83
DDV — Discrete Dependent Variables .................................................................. 92
DEBUG — Debugging Tools .................................................................................. 98
DECLARE — Setting Data Types ......................................................................... 99
DEDIT — Initiating RATS Data File Editing ......................................................... 101
Introduction

DEFAULT — Changing Default Options ................................................................. 103
DELETE — Removing Series from a RATS Data File ........................................... 104
DENSITY — Estimate Density Function ................................................................. 105
DETERMINISTIC — Listing the “Other” Variables in a VAR .................................. 108
DIFFERENCE — Time Series Differencing .............................................................. 109
DIMENSION — Setting Array Sizes ...................................................................... 112
DISPLAY — Displays Computed Values ................................................................. 113
DLM — Dynamic Linear Models ............................................................................ 116
DO — Simple Loops Over an Index ...................................................................... 124
DOFOR — Looping Over a List .............................................................................. 126
ECT — Error Correction Terms ............................................................................ 128
EDIT — Screen Data Editing ................................................................................ 130
ENCODE — Restricted Regressions ....................................................................... 135
END — Ending a Program .................................................................................... 137
ENTER — Defining Your Own Supplementary Cards ............................................ 138
ENVIRONMENT — Error Modes and Other Options .............................................. 140
EQUATION — Defining Linear Equations ............................................................... 142
EQV — Attaching Names to Numbered Series .................................................... 146
ERRORS — Decomposition of Forecast Variance .................................................. 147
ESMOOTH — Exponential Smoothing ................................................................... 153
ESTIMATE — Estimating a VAR System ................................................................. 158
EWISE — Elementwise Operations on Matrices ................................................... 162
EXCLUDE — Testing Exclusion Restrictions ....................................................... 164
EXECUTE — Executing a PROCEDURE ............................................................... 166
EXP — Taking the Anti-Log of a Series ................................................................. 169
EXTREMUM — Extreme Values ............................................................................ 170
FFT, IFT: Fourier Transforms ................................................................................ 171
FILTER — General Linear Filtering ....................................................................... 173
FIND — General Optimization .............................................................................. 179
FIXED — Setting Values of Arrays in PROCS and FUNCTIONS ............................ 183
FMATRIX — Matrix from a Filter ......................................................................... 185
FOLD: Folding a Spectrum .................................................................................. 187
FORECAST — Dynamic and Static Forecasts ....................................................... 188
FREQUENCY: Initializing Frequency Domain Analysis .......................................... 194
FRML — Creating Formulas ................................................................................ 196
FUNCTION — User-Defined Functions ................................................................. 203
GARCH — ARCH and GARCH Models ................................................................. 206
GCONTOUR — Contour Plots .............................................................................. 212
GRAPH — High-Resolution Time Series Graphs .................................................. 214
GROUP — Building General Simultaneous Models .............................................. 224
GRPARM — Graphics Parameters ....................................................................... 225
GRTTEXT — Adding Text to Graphs .................................................................... 228
GSET — Setting Series of Arrays ......................................................................... 231
HALT — Stopping Execution From Within Compiled Sections ............................. 232
Introduction

HISTORY — Historical Decompositions ................................................................. 233
IF and ELSE — Conditional Execution ................................................................. 237
IMPULSE — Impulse Response Functions ............................................................. 240
INCLUDE — Adding Series to a RATS Data File .................................................. 246
INFOBOX — Informational Dialog Boxes and Progress Bars ............................. 248
INITIAL — Solving Yule-Walker Equations ......................................................... 250
INPUT — General Information Input ................................................................. 251
INQUIRE — Obtaining Information about Series ............................................... 254
INSTRUMENTS — Setting the Instrument List .................................................... 257
KALMAN — Kalman Filtering ............................................................................. 259
KFSET — Kalman Filter Setup .......................................................................... 263
LABELS — Setting Output Labels for Series ..................................................... 266
LAGS — Listing the Lags for a VAR ................................................................. 267
LDV — Limited Dependent Variable Estimation ............................................... 268
LINREG — Linear Regressions .......................................................................... 272
LIST,CARDS — Shorthand for Sets of Supplementary Cards ............................. 278
LOCAL — Declaring Local Variables ................................................................. 280
LOG — Taking the Log of a Series ................................................................ 281
LOOP — Loop Forever ..................................................................................... 282
LQPROG — Linear and Quadratic Programming ............................................... 283
MAKE — Creating an Array from Data Series .................................................... 287
MAXIMIZE — General Maximization ................................................................. 290
MCOV — Consistent Covariance Matrices ......................................................... 295
MEDIT — Spreadsheet Array Editor/Viewer ..................................................... 300
MENU,CHOICE — Pop Up Menu Selections ..................................................... 303
MESSAGEBOX — Simple Dialog Boxes ............................................................ 304
MODIFY — Equation Maintenance ................................................................. 306
MRESTRICT — Testing General Restrictions ..................................................... 307
MVSTATS — Moving Statistics and Fractiles ..................................................... 309
NEXT — Loop Control ..................................................................................... 312
NLLS — Non-Linear Least Squares .................................................................... 313
NLPAR — Controlling Non-linear Estimation .................................................... 318
NLSYSTEM — Non-linear Systems Estimation .................................................. 320
NNLEARN — Neural Net Training ...................................................................... 328
NNTEST — Neural Net Testing and Predictions ............................................... 333
NONLIN — Setting Free Parameters for Non-Linear Estimation ........................ 334
NPREG — Non-Parametric Regressions ............................................................. 336
OPEN — Opening I/O Units ............................................................................. 339
OPTION — Defining Options for PROCEDURES ............................................... 341
ORDER — Sorting Data and Ranking Data ....................................................... 344
PANEL — Panel Data Transformations .............................................................. 346
PFORM — Forming Panel Data Series ............................................................... 348
POLAR: Polar Decomposition ........................................................................ 351
PREGRESS — Panel Data Regressions .............................................................. 353
PRINT — Printing Data .................................................................................. 356

RATS Reference Manual
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRJ</td>
<td>Fitted Values/Normal Distribution Statistics</td>
<td>358</td>
</tr>
<tr>
<td>PROCEDE</td>
<td>User-Defined Procedures</td>
<td>362</td>
</tr>
<tr>
<td>PRD</td>
<td>Printing Data File Series</td>
<td>364</td>
</tr>
<tr>
<td>PST</td>
<td>Analysis of Variance for Panel Data</td>
<td>366</td>
</tr>
<tr>
<td>QUERY</td>
<td>Requesting Input from the User</td>
<td>368</td>
</tr>
<tr>
<td>QUIT</td>
<td>Aborting Data Editing</td>
<td>370</td>
</tr>
<tr>
<td>QZ</td>
<td>QZ Decomposition</td>
<td>371</td>
</tr>
<tr>
<td>RATIO</td>
<td>Tests with Multiple Equation Systems</td>
<td>372</td>
</tr>
<tr>
<td>READ</td>
<td>General Information Input</td>
<td>374</td>
</tr>
<tr>
<td>RELEASE</td>
<td>Releasing Segments of Memory</td>
<td>377</td>
</tr>
<tr>
<td>RENAME</td>
<td>Renaming Data File Series</td>
<td>378</td>
</tr>
<tr>
<td>REPORT</td>
<td>Report Generation</td>
<td>379</td>
</tr>
<tr>
<td>RESTRICT</td>
<td>Testing or Imposing General Linear Restrictions</td>
<td>385</td>
</tr>
<tr>
<td>RETURN</td>
<td>Returning from a PROCEDURE or FUNCTION</td>
<td>390</td>
</tr>
<tr>
<td>REWIND</td>
<td>Rewinding a RATS I/O Unit</td>
<td>391</td>
</tr>
<tr>
<td>RLS</td>
<td>Recursive Least Squares</td>
<td>392</td>
</tr>
<tr>
<td>RREG</td>
<td>Robust Regression</td>
<td>396</td>
</tr>
<tr>
<td>RTOC</td>
<td>Transferring from Real to Complex</td>
<td>400</td>
</tr>
<tr>
<td>SAMPLE</td>
<td>Extracting Data and Changing Frequencies</td>
<td>402</td>
</tr>
<tr>
<td>SAVE</td>
<td>Saving an Edited RATS Format File</td>
<td>404</td>
</tr>
<tr>
<td>SCATTER</td>
<td>High-Resolution X-Y Scatter Plots</td>
<td>405</td>
</tr>
<tr>
<td>SEASONAL</td>
<td>Creating Seasonal Dummies</td>
<td>413</td>
</tr>
<tr>
<td>SEED</td>
<td>Initializing the Random Number Generator</td>
<td>416</td>
</tr>
<tr>
<td>SELECT</td>
<td>User Selection from a List</td>
<td>418</td>
</tr>
<tr>
<td>SET</td>
<td>General Data Transformations</td>
<td>420</td>
</tr>
<tr>
<td>SHOW</td>
<td>Run-Time Information</td>
<td>422</td>
</tr>
<tr>
<td>SIMULATE</td>
<td>Random Simulations of a Model</td>
<td>423</td>
</tr>
<tr>
<td>SMPL</td>
<td>Setting the Default Entry Range</td>
<td>427</td>
</tr>
<tr>
<td>SOURCE</td>
<td>Secondary Input Files</td>
<td>429</td>
</tr>
<tr>
<td>SPECIFY</td>
<td>Bayesian Priors for Vector Autoregressions</td>
<td>430</td>
</tr>
<tr>
<td>SGRAPH</td>
<td>Special Graphs</td>
<td>434</td>
</tr>
<tr>
<td>SStats</td>
<td>Series Statistics</td>
<td>437</td>
</tr>
<tr>
<td>STATISTICS</td>
<td>Sample Statistics</td>
<td>439</td>
</tr>
<tr>
<td>STEPS</td>
<td>Static Forecasts</td>
<td>443</td>
</tr>
<tr>
<td>STORE</td>
<td>Saving Data on a RATS Data File</td>
<td>446</td>
</tr>
<tr>
<td>STWISE</td>
<td>Stepwise Regressions</td>
<td>449</td>
</tr>
<tr>
<td>SUMMARIZE</td>
<td>Sums of Coefficients</td>
<td>453</td>
</tr>
<tr>
<td>SUR</td>
<td>Linear Systems Estimation</td>
<td>457</td>
</tr>
<tr>
<td>SWEEP</td>
<td>Sweep Regressions</td>
<td>464</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>Setting Up a VAR System</td>
<td>466</td>
</tr>
<tr>
<td>TABLE</td>
<td>Basic Statistics on Series</td>
<td>468</td>
</tr>
<tr>
<td>TAPER</td>
<td>Tapering Data</td>
<td>471</td>
</tr>
<tr>
<td>TEST</td>
<td>Testing Specific Coefficient Values</td>
<td>473</td>
</tr>
<tr>
<td>THEIL</td>
<td>Computing Forecast Performance Statistics</td>
<td>476</td>
</tr>
<tr>
<td>TRFUNCTION</td>
<td>Transfer Functions</td>
<td>482</td>
</tr>
</tbody>
</table>
Introduction

TVARYING — Time-Varying Coefficients ................................................................. 484
TYPE — Setting Procedure Parameter Types .......................................................... 485
UFORECAST — Univariate Forecasting ................................................................. 486
UNTIL — Conditional Loops .................................................................................. 489
UPDATE — Altering Data on a RATS Data File ...................................................... 490
USERMENU — Defining Pull-Down Menus ............................................................... 491
VADD — Adding a Variable to An Equation ............................................................ 494
VARIABLES — Specifying the Variables in a VAR .................................................. 495
VCV — Computing a Residual Covariance Matrix .................................................. 496
VREPLACE — Substituting Variables Within an Equation ........................................ 499
WHILE — Conditional Looping ............................................................................. 501
WINDOW: Smoothing Spectral Estimates ............................................................... 502
WRITE — Printing Matrices .................................................................................... 505
X11 — X11 Seasonal Adjustment .......................................................................... 507

Section 2: The RATS Functions 511

Alphabetical Listing of RATS Functions ................................................................. 520

Appendix A: Expression Syntax and Operators 569

Appendix B: List of Reserved Variable Names 573

Appendix C: RATS Error Messages and Troubleshooting 577

Appendix D: Instructions Listed by Task 589

Bibliography 597

Index 605
Section 1: The RATS Instructions

Section 1 of the Reference Manual provides a detailed description of every RATS instruction, except the frequency domain instructions which are covered in Section 2. The instructions are organized alphabetically, with two exceptions: ELSE is included with its companion instruction IF; and CHOICE is described with MENU.

The description of each instruction begins with the basic syntax of the instruction, including its full name, the names we use for the parameters (if any), and the syntax for any supplementary cards, text cards, or data cards.

Several instructions have more than one possible syntax depending on the options used or the context of the instruction. For example, the description of CALENDAR shows the ten possible forms of the instruction.

Throughout this manual, we use the following conventions:

Instructions are always in bold Courier font. For example BOXJENK, DATA.

Parameters are always in italicized Courier font. For example, start, end, series.

Options are always in Courier font (such as DATES and ORGANIZATION).

For instructions which have one or more options, we include an “Options” section which describes them in detail. Here, each option appears initially in bold type for clarity. We use the following conventions for describing options:

Switch Options are listed in a form such as [dates]/nodates. The brackets indicate the default setting of the option.

Choice Options each of the possible choices is listed after the option name, with brackets around the default choice. For example: organization=[rows]/columns.

Value Options are presented like missing=missing value code. We put a description of the value in italics, similar to instruction parameters. Further discussions of this value will use the same phrase, again in Courier italics.

If a value option has a default value, it appears in bold type, enclosed in brackets. For example,

cvcrit=convergence criterion for r  [.0001]
**Accumulate**

**ACCUMULATE — Computing Partial Sums**

ACCUMULATE computes the partial sums of series and stores the results in newseries. Explicitly, entry $N$ of newseries is equal to the sum of the first $N$ entries of series.

```
accumulate(option)  series  start  end  newseries  newstart
```

**Wizard**

You can use the Transformations wizard on the Data menu to compute partial sums. Choose the “Partial Sums–Accumulation” operation in the dialog box.

**Parameters**

- **series** Series to sum.
- **start**  **end** Range of entries over which to compute partial sums. If you haven’t set a SMPL, this defaults to the defined range of series.
- **newseries** Series for the partial sums. By default, newseries=series.
- **newstart** New starting entry for the resulting series. By default, newstart=start.

**Option**

- **standardize/ [nostandardize]**
  
  When you use the option STANDARDIZE, ACCUMULATE divides the entries of newseries by the final sum. newseries thus ends with the value 1.0. This option was called SCALE in previous versions.

**Missing Values**

RATS excludes missing values in series from the calculation of the partial sums and sets the corresponding entries in newseries to missing. For example:

```
set trend 1 10 = t
set trend 5 6 = %na
accumulate trend / trendsum
print
```

produces the following output
### Examples

```
acc sales / totalsales
set pchange = sales/totalsales{1}
linreg pchange
\# constant totalsales{1}
```

takes the series `SALES`, computes the total sales through each entry, then computes a linearized logistic trend regression (Pindyck and Rubinfeld (1998), p. 472).

```
impulse(model=varmodel,result=impulses,noprint,steps=nsteps)
dec rect[ser] accumimp(%nvar,%nvar)
do i=1,%nvar
   do j=1,%nvar
      accumulate impulses(i,j) 1 nsteps accumimp(i,j)
   end do i
end do j
```

This stores accumulated impulse responses into an array of series called `ACCUMIMP`.

### Notes

`SSTATS` can compute sums or averages for a single range of data. For instance:

```
sstats(mean) 1963:1 1963:12 deuip>>avg1963
```

computes (into `AVG1963`) the average value of `DEUIP` over 1963:1 to 1963:12

If you need to use a depreciation or appreciation factor applied to the previous total, use `SET` with the `FIRST` option rather than `ACCUMULATE`. The following computes a capital stock measure from an investment series, assessing 7% depreciation:

```
set(first=13.7) capital = 0.93*capital{1} + invest
```

Finally, use `MVSTATS` if you want to compute averages across a set of entries which shifts as you move through the data set—that is, over a moving “window” of entries. `ACCUMULATE` just adds new entries on at the end without dropping the early ones.
ALLOCATE — Preparing for Work with Data Series

The **ALLOCATE** instruction sets the default data series length. This information is used by instructions, such as **DATA** and **SET**, that work with series.

Prior to Version 6 of RATS, the **ALLOCATE** command was required in any program that used data series. It is now optional. If you omit the **ALLOCATE** instruction, you can set the default range by supplying explicit **start** and **end** dates (or entry numbers) on your first **DATA**, **SET**, or **SMPL** instruction.

If you use **ALLOCATE**, you will usually use the first of the two forms shown below. The second form is generally seen only in programs written for much older versions.

```
allocate  length
allocate  series  length
```

**Wizard**

The **Data** wizards, located on the **Data** menu, automatically determine the appropriate **ALLOCATE** setting when the data are read from a single data file.

**Parameters**

- **length**: This sets the default series length. This is not a binding constraint—you can define series (such as out-of-sample forecasts) which exceed it. We would recommend that you set it to the length of the data which you read with **DATA** since **length** sets the default value for the **end** parameter on **DATA**.

  The **length** parameter can be:

  - a simple entry number.
  - a date value, if you have used **CALENDAR** to define a dating scheme. Note that you **must** use a “:” when specifying a date, even for annual data (for instance, 1991:1)
  - **individual//observation** for panel data. **observation** can be an entry number or a date.

- **series**: If using the second form, you specify a non-zero value for this parameter, which tells RATS to create a set of data series numbered 1 to **series**.

**Examples**

```
allocate 1200
calendar(d) 1986:1:8
allocate 2002:12:31
```

1200 observation data set

*Daily, Jan. 8, 1986 through Dec. 31, 2002*
Pre-Allocating Series

If you use the second form, RATS will create series numbers 1 to series. Any series created later by other instructions will be numbered series+1 and above. These numbered series can be helpful when working with large, well-structured data sets. However, many of these cases are more easily handled using VECTORS of SERIES (see Section 16.3 in the User’s Guide).

Variables Defined by ALLOCATE

%X

If you use the series parameter to define a block of numbered series, ALLOCATE defines the RECTANGULAR array %X(n,m) with dimensions length x series. Entry (n,m) of this array references entry n of series m.

You can manipulate %X and its elements just like any other RECTANGULAR array.

See Also . . .

CALENDAR Sets the date of the first entry in the data set. In time series or panel data sets this comes before ALLOCATE.

SCRATCH Creates new numbered data series.

Section 16.3 Numbered data series.
AR1 — Regression with Autocorrelation Correction

AR1 estimates a regression, correcting for first order serially correlated errors. With the INSTRUMENTS option, it does two-stage least squares. See Section 5.5 in the User’s Guide for more information on using the AR1 instruction.

```
ar1( options )  depvar  start  end  residuals
#< supp. card >  explanatory variables in regression format
```

Wizard

AR1 estimation is available via the Regressions wizard on the Statistics menu. Use the “Method” list to select either “AR(1)—Regression” or “AR(1)—Instrumental Variables”.

Parameters

- **depvar**  
  Dependent variable for the regression.
- **start end**  
  Range to use in estimation. If you haven’t set a SMPL, this defaults to the largest range for which all variables involved are defined. If you choose a method which does not retain the initial observation (Cochrane–Orcutt or Hildreth–Lu), RATS will actually run the regressions beginning at \( start+1 \), using entry \( start \) to provide the lag for \( start+1 \).
- **residuals**  
  (Optional) Residuals are automatically saved in \( \%\)RESIDS. You can use this parameter to save them in a different series.

A fifth coefficient used in previous versions for saving the coefficients has been deprecated. Coefficients are now saved automatically in the vectors \( \%\)BETA and \( \%\)BETASYS, as described under “Variables Defined” later in this section.

Options

```
method=corc/[hilu]/maxl/search/pw
```

This chooses the method for estimation. See the following page for details:

- **CORC**  
  is Cochrane–Orcutt, or, for instrumental variables, Fair’s (1970) procedure.
- **HILU**  
  (the default) is Hildreth–Lu, a grid search procedure.
- **MAXL**  
  is Beach and MacKinnon’s (1978) maximum likelihood procedure.
- **SEARCH**  
  is a maximum likelihood grid search procedure.
- **PW**  
  is Prais-Winsten, which is similar to SEARCH, but doesn’t use the log variance terms from the likelihood.

RATS may not be able to honor your choice. The methods which retain the initial observation (MAXL, SEARCH and PW) cannot be used for instrumental variables and the iterated methods (MAXL and CORC) cannot be used if there are missing observations. AR1 will pick the closest permitted choice when it must switch.
rho= input value of rho [not used]
Use this if you want to input the value of $\rho$ rather than having it estimated.

cvcrit= convergence criterion for $\rho$ [.0001]
The goal in each of the methods described above is to reach a point where the estimate of $\rho$ changes by less than the convergence criterion.

[print]/noprint
vcv/[novcv]
smpl= SMPL Series or formula (User’s Guide, Section 5.2)
dfc= Degrees of Freedom Correction (Section 5.15)
unravel/[nounravel] (Section 5.11)
equation= equation to estimate
entries= number of supplementary card entries to process [all]
title= title to identify estimation method
These are the same as for LINREG.

instruments/[noinstruments]
wmatrix= weighting matrix
Use the INSTRUMENTS option to do two-stage least squares. You must set your instruments list first using the instruction INSTRUMENTS. WMATRIX is described in Section 7.8 of the User’s Guide.

define= equation to define
frml= formula to define
These define an equation and formula, respectively, for forecasting purposes. The equation (or formula) created incorporates the serial correlation within it, so that an estimated model of

$$y_t = 20.0 + 2.5x_t + u_t, \quad u_t = .8u_{t-1} + \varepsilon_t$$

produces the equivalent equation

$$y_t = .8y_{t-1} + 4.0 + 2.5x_t - 2.0x_{t-1} + \varepsilon_t$$

heterogenous/[noheterogenous]
prhos= series of rho values [not used]
These only appeal to panel data sets (User’s Guide, Section 14.5). With HETEROGENOUS, RATS estimates a separate $\rho$ for each cross-sectional unit. This is a simple two-step estimation procedure, with no iteration. None of the METHOD choices apply. Note: this option was called DIFFERING in previous versions. DIFFERING is still recognized, but not recommended.

When using HETEROGENOUS, you can use PRHOS to save the series of estimated rho values for each individual.
You can use this to provide a description of the estimation method, or the model that you are estimating.

Technical Details

For the following model with first order serially correlated errors:

\( y_i = X_i \beta + u_i \), \( u_i = \rho u_{i-1} + \epsilon_i \)

the (log) likelihood function, assuming Normality, is

\[
\frac{-T}{2} \log(2\pi) - \frac{T}{2} \log(\sigma^2) + \frac{1}{2} \log(1 - \rho^2) - \frac{1}{2\sigma^2} \left( (y_i - X_i \beta)^2 \right)
\]

\[
\frac{1}{2\sigma^2} \left\{ \sum_{t=2}^{T} \left( y_i - \rho y_{i-1} - (X_i - \rho X_{i-1}) \beta \right)^2 \right\}
\]

- METHOD=MAXL and METHOD=SEARCH maximize this function. MAXL does this by an iterative procedure while SEARCH uses an efficient grid search.
- METHOD=CORC and METHOD=HILU minimize the part in braces, with CORC using an iterative procedure and HILU doing a grid search.
- METHOD=PW is a GLS procedure which includes terms for the first observation. It minimizes the part in braces plus \((1 - \rho^2) (y_i - X_i \beta)^2\)

The goal in every method is to reach a point where the estimate of \( \rho \) changes by less than convergence criterion (set by the CVCRIT option). This usually takes more trials with the search procedures. However, the search procedures guarantee that you have found the global optimum.

The objective function for two-stage least squares is

\[
\sum (u_i - \rho u_{i-1}) Z_i (Z'Z)^{-1} Z_i' (u_i - \rho u_{i-1})
\]

where \( Z \) is the vector of instruments. Given \( \rho \), \( \beta \) is estimated by two-stage least squares of \( y_i - \rho y_{i-1} \) on \( X_i - \rho X_{i-1} \). If you choose METHOD=HILU, RATS uses a search procedure to minimize (3) over \( \rho \). If you use METHOD=CORC, given \( u = y - X \beta \), \( \rho \) is estimated by

\[
\sum \frac{u_i u_{i-1}}{\sum u_{i-1}^2}
\]
Missing Values

If there are any missing values within the data range, the simple iterative process described above for the Cochrane–Orcutt (CORC) and Beach–MacKinnon (MAXL) estimators can’t be used, since there will be terms missing in (4). If you have requested one of these, the most similar search procedure will be used instead.

If there is a gap of $s$ periods in the data before period $t$, the likelihood function will include the extra term

$$
\frac{1}{2} \log \left( \frac{1 - \rho^2}{1 - \rho^{2s}} \right)
$$

and the term for $t$ in the sum in (2) is replaced by

$$
(y_t - \rho^s y_{t-s} - (X_t - \rho^s X_{t-s})\beta) \left( \frac{1 - \rho^2}{1 - \rho^{2s}} \right)
$$

Both of these are to adjust for the fact that

$$
u_t|u_{t-s} \sim N(\rho^s u_{t-s}, \sigma^2(1 + \rho^2 + \rho^4 + \ldots + \rho^{2(t-s-1)}))$$

Examples

The first estimates an investment equation using instrumental variables, producing the results shown on the next page.

```
ar1(inst,frml=investeq) invest
# constant ydiff gnp{1} rate{4}
```

This estimates first by Hildreth-Lu (the default method) and then by maximum likelihood:

```
ar1 loggpop
# constant logpg logypop logpnc logpuc
ar1(method=maxl) loggpop
# constant logpg logypop logpnc logpuc
```

Variables Defined by AR1

AR1 defines the standard regression variables (see the section on LINREG for details). The variable %RHO holds the estimated $\rho$ coefficient, rather than the autocorrelation coefficient of the residuals. The standard %BETA and %XX variables do not include the $\rho$ term. The variables %BETASYS and %XXSYS hold the full coefficient vector and covariance matrix, respectively, for the entire estimation, including the $\rho$ term.
Hypothesis Tests
You can use any of the hypothesis testing instructions after AR1, but you can’t test RHO using EXCLUDE or SUMMARIZE.

Fitted Values and Forecasting
You can use the PRJ instruction to get fitted values after AR1. PRJ can also compute forecasts of AR1 models. Or, if you use DEFINE to save the estimated equation, you can use UFORECAST or FORECAST to get forecasts.

Sample Output

Regression with AR1 - Estimation by Instrumental Variables
Dependent Variable INVEST
Quarterly Data From 1950:02 To 1985:04
Usable Observations 143 Degrees of Freedom 138
Centered R**2 0.985487 R Bar **2 0.985066
Uncentered R**2 0.998511 T x R**2 142.787
Mean of Dependent Variable 385.10769231
Std Error of Dependent Variable 130.65960726
Standard Error of Estimate 15.96727320
Sum of Squared Residuals 35183.626277
Durbin-Watson Statistic 1.878401
Q(35-1) 41.615770
Significance Level of Q 0.17310844

<table>
<thead>
<tr>
<th>Variable</th>
<th>Coeff</th>
<th>Std Error</th>
<th>T-Stat</th>
<th>Signif</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Constant</td>
<td>-63.75464036</td>
<td>22.29827500</td>
<td>-2.85917</td>
<td>0.00490723</td>
</tr>
<tr>
<td>2. YDIFF(1)</td>
<td>0.10192898</td>
<td>0.04765235</td>
<td>2.13901</td>
<td>0.03419497</td>
</tr>
<tr>
<td>3. GNP</td>
<td>0.20599768</td>
<td>0.01029049</td>
<td>20.01825</td>
<td>0.00000000</td>
</tr>
<tr>
<td>4. RATE(4)</td>
<td>-6.35287532</td>
<td>1.55915896</td>
<td>-4.07455</td>
<td>0.00000774</td>
</tr>
<tr>
<td>5. RHO</td>
<td>0.78992212</td>
<td>0.05313052</td>
<td>14.86758</td>
<td>0.00000000</td>
</tr>
</tbody>
</table>

The $R^2$, other summary statistics and the residuals are based upon the complete model—so they use the $\varepsilon$’s, not the $u$’s. The estimate of $\%RHO$ is separated from the other regressors in the regression output. RATS computes the standard errors and covariance matrix from a linearization of the objective function.

If you use the HETEROGENOUS option with panel data, RATS omits the output for $\%RHO$, and computes the standard errors from the second stage regression on the quasi-differenced data.
Higher-order Autocorrelation Corrections

You can use the `NLLS` instruction to estimate a model with corrections for higher-order serial correlation. For instance, the following includes both first- and second-order autocorrelation correction (see `FRML` for more details on how this code works):

```
nonlin rho1 rho2
linreg y
# constant x1 x2
frml(lastreg,names="B",addparms) regfrml
frml auto1 = rho1*y{1} + rho2*y{2} + regfrml(t) - $
                 rho1*regfrml(t-1) - rho2*regfrml(t-2)
compute rho1=rho2=0.1
nlls(frml=auto1) y
```
ASSOCIATE: Assigning Coefficients to Equations

ASSOCIATE allows you to set the coefficients for an equation. This can be particularly useful for equations that you want to have in a specific form for Monte Carlo work, or for equations which have been estimated outside of RATS.

\[\text{associate( options ) equation coeff VECTOR}\]

# coefficients (under some circumstances)

Details

When you estimate an equation within RATS, the estimating instruction automatically sets the coefficients and residual variance. So, you only need ASSOCIATE if you wish to alter what is stored, or to assign coefficients to equations (such as identity equations) that have not been estimated within your RATS program. ASSOCIATE provides three ways to set the coefficients of an equation:

1. It can read the coefficients from supplementary cards (the default method).
2. It can read them from a file (using the options BINARY or FREE)
3. It can copy them from a VECTOR by using the coeff VECTOR parameter.

Parameters

equation ASSOCIATE sets the coefficients of this equation.
coeff VECTOR (Optional) This is a VECTOR which holds the coefficients.

Options

\text{variance= residual variance}
Residual variance for this equation. You only need to supply this if you are going to use SIMULATE, IMPULSE or ERRORS. It is usually set when the equation is estimated.

\text{residuals= series of residuals}
Series holding the residuals for this equation. You only need this information if you want to use FORECAST, STEPS, SIMULATE or THEIL with an ARMA equation, or the HISTORY instruction with any equation. It is usually set by using the residuals parameter on the LINREG or other estimation instruction.

\text{frml=FRML to associate with the equation}
Use the FRML option to associate a FRML with the equation. Whenever the equation is estimated, the FRML will be updated as well. This has the same effect as using a FRML option on an EQUATION instruction.
[coeffs]/nocoeffs
Use NOCOEFFS when you want to use RESIDUALS, VARIANCE or FRML, but you do not want to change the coefficients of *equation*. Omit the supplementary card if you use NOCOEFFS.

perm/[noperm]
You can only use the PERM option in conjunction with the *coeff VECTOR* parameter. PERM makes the association with the specified VECTOR permanent—if you change the values of the VECTOR, you change the coefficients.

free/[nofree]
binary/[nobinary]
unit=[data]/input/other unit
These options let you read the coefficients from a file rather than from a supplementary card. With FREE, the coefficients are read free-format (tab, blank, or comma-delimited text file) and with BINARY, they are read as binary data (see Section 2.13 of the *User’s Guide*). In either case, omit the supplementary card.

Examples

equation  yeq  y
# constant  y{1 2}  
associate(variance=.01)  yeq
# 5.0  1.3  -.4  

sets up as equation YEQ the two lag autoregression

\[ y_t = 5.0 + 1.3 y_{t-1} - 0.4 y_{t-2} + u_t; \quad \text{Var}(u_t) = .01 \]

This could also have been done by adding options to the **EQUATION** instruction:

equation(coeffs=||5.0,1.3,-.4||,variance=.01) yeq
# constant y{1 2}  

equation 1 shortrate
# constant shortrate{1 to 12} longrate{1 to 12}
equation 2 longrate
# constant shortrate{1 to 12} longrate{1 to 12}
open coeffile coeff.dat
associate(free,unit=coeffile) 1  Coefficients for SHORTRATE equation
associate(free,unit=coeffile) 2  Coefficients for LONGRATE equation

reads the coefficients for two equations from the text file COEFF. DAT. You might find this useful if you are doing a good deal of analysis with one estimated model and don’t want to reestimate the coefficients in each RATS program that uses the model.

There are a few ways to save estimated coefficients to a file. The example on the following page demonstrates one way.
linreg shortrate
  # constant shortrate{1 to 12} longrate{1 to 12}
  set coeff1 1 %nreg = %beta(t)

linreg longrate
  # constant shortrate{1 to 12} longrate{1 to 12}
  set coeff2 1 %nreg = %beta(t)

open copy coeff.dat
copy / coeff1 coeff2
close copy

Notes
Before Version 5, ASSOCIATE was used to reset the coefficients of a Vector Autoregression when doing Monte Carlo integration and similar types of simulations. The new functions %MODELGETCOEFFS and %MODELSETCOEFFS provide a much simpler way to do that now.

Similarly, the functions %EQNSETCOEFFS and %EQNSETVARIANCE provide alternative methods of setting the coefficient values and variance of an individual equation. They may be more convenient to use than ASSOCIATE in many cases.

See Also . . .
- EQUATION     Defines equations
- FRML         Defines formulas (FRMLS)


**BOOT — Randomization for Bootstrapping**

`BOOT` creates a SERIES of INTEGERS and fills all or part of the array with random integers. The most common use for this instruction is drawing observations at random from a series or set of series as part of a bootstrapping or randomization operation. See Section 13.10 of the *User's Guide* for detailed information.

```
boot( option )             BOOT series  start  end  lower  upper
```

**Parameters**

- **BOOT series**
  The SERIES of INTEGERS created by `BOOT`. You do not need to `DECLARE` or `DIMENSION` it.

- **start  end**
  The range of entries in `BOOT series` set by `BOOT`. If you haven’t set a `SMPL`, this defaults to the `ALLOCATE` range.

- **lower  upper**
  The lower and upper bounds on the value range of the random integers. These default to `start` and `end`. If you specify these as a range of entries, `BOOT` fills the series with random entry numbers from that range.

**Option**

- **[replace]/noreplace**
  Determines whether or not the sampling will be done with replacement or without replacement. With `NOREPLACE`, once a value is drawn, it won’t be drawn again for a different element of `BOOT series`. With `REPLACE`, numbers may be drawn more than once. Drawing with replacement is the normal procedure in bootstrapping operations, as the sample is treated as if it were the population from which the data are drawn. Drawing without replacement is typically part of approximate randomization analyses, and is usually done to shuffle the entire entry range.

- **block=block size [not used]**

- **method=[overlap]/nooverlap/stationary**
  Used for block bootstrapping. `METHOD=OVERLAP` allows for overlapping blocks, so every full block within `[lower,upper]` can be selected. `METHOD=NOOVERLAP` partitions `[lower,upper]` into separate blocks of size “`block size`” and randomizes among them. For `METHOD=STATIONARY`, `block size` can be real-valued. When randomizing an entry, a new start point within `[lower,upper]` is chosen with probability `1/block size`; otherwise, the previous value is incremented by one. (A new value will also be chosen if incrementing would take the value above `upper`). `block size` is thus (except for the truncation effect at `upper`) the expected size of the block.
Examples

calendar(q) 1980:1
allocate 1999:4
boot entries / 1990:1 1999:4

The dates of lower and upper correspond to the 41st and the 80th entry numbers, so BOOT fills the ENTRIES series with random integers ranging from 41 to 80.

@hurst(header="R/S Analysis of Equally-Weighted Returns") ew
boot(block=40,method=nooverlap) shuffle
set reshuffle = ew(shuffle(t))
@hurst(header="R/S Analysis of Block Shuffled Returns") reshuffle

runs the HURST procedure on the series EW, then does a non-overlapping block reshuffling of its data and re-executes the procedure.

boot(noreplace) entry 1 50
set shuffle 1 50 = ressqr(entry(t))

creates SHUFFLE as a random reordering of the fifty elements of the series RESSQR.

Notes

BOOT draws with replacement by scaling and translating uniform random numbers. For instance, the following is equivalent to the combination of the BOOT and SET instructions in the first example in this section:

set rndent = fix(%uniform(1990:1,1999:4+1))

In general, use FIX(%UNIFORM(lower,upper+1)). The +1 is needed because the upper bound is included as a possible value.

Draws without replacement are done by randomly shuffling the numbers between LOWER and UPPER.

See Also . . .

SEED Controls the seeding of the random number generator
%RANINTEGER (Function) Draws a random integer from a set range
%RANPERMUTE (Function) Returns a random permutation of \{1,...,n\}
%RANCOMBO (Function) Returns a random combination (sample without replacement) from \{1,...,n\}
%RAN (x) (Function) Fills an array with draws from a Normal distribution with a standard deviation of x.
BOXJENK — ARIMA, Transfer Function, Intervention Models

**BOXJENK** estimates ARIMA, seasonal ARIMA, transfer function and intervention models. See Chapter 9 of the User’s Guide for more information.

```
boxjenk(options) depvar start end residuals
# series numlags denlags delay (one per INPUT)
# explanatory variables in regression format (if using REGRESS option)
```

**Wizard**

The *Box Jenkins (ARIMA) Models* wizard on the *Statistics* menu provides dialog-driven access to most of the features of the **BOXJENK** instruction.

**Parameters**

- **depvar** Dependent variable.
- **start end** Estimation range. If you use these to set the range (rather than using the default range), and are not using the maximum likelihood method, you must set `start` to allow for:
  - the autoregressive and seasonal autoregressive lags in the dependent variable.
  - the required lags for input variables. The total number of lags of actual data required is the sum of the highest AR lag, the highest seasonal AR lag and the highest lag in the input numerator.

You do not have to allow for lags in the moving average part or any denominator lags in the inputs.

If you haven’t set a **SMPL**, the range defaults to the maximum range over which all of these lags are defined.

- **residuals** (Optional) The residuals are automatically saved in the series `%RESIDS`. You can supply a series name for this parameter if you also want to store the residuals in a different series.

**Options**

- **constant/[noconstant]**
  - **constant** includes an intercept (constant) term in the model as an estimated parameter (by default, there is none). **DEMEAN** offers an alternative for handling series with non-zero means. It removes the mean from the dependent variable (after differencing, if required), prior to estimating the model. The mean removal is done using an internal copy—the actual series itself is not affected.
**BoxJenk**

- **span**=seasonal span for seasonal ARMA [CALENDAR seasonal—see below]
- **diffs**=number of regular differencings [0]
- **sdiffs**=number of seasonal differencings [0]
- **ar**=list of autoregressive lags [0]
- **ma**=list of moving average lags [0]
- **sar**=list of seasonal autoregressive lags [0]
- **sma**=list of seasonal moving average lags [0]

These jointly specify the ARMA part of the model. See “Technical Information” for details on model parameterization. For frequencies defined in terms of the number of periods per year, **SPAN** defaults to the CALENDAR seasonal (for example, **SPAN**=12 for monthly data). For frequencies like weekly and daily, where there is no clear definition of a span, **SPAN** defaults to 1.

RATS supports any combination of lags for the AR, MA, SAR, and SMA options:

- For N consecutive lags (all lags from 1 through N) for a given parameter, use the format AR=N, MA=N, etc.
- For non-consecutive lags, use ||list of lags||. If you are listing more than one lag, separate them by commas. For example, use AR=||3|| for an AR parameter at lag 3 only, while AR=||1,3|| gives parameters on lags 1 and 3. You can also use a VECTOR of INTEGERS.

- **[print]/noprint**
- **vcv/ [novcv]**
- **dfc=**Degrees of Freedom Correction  (*User’s Guide*, Section 5.15)
  These are the same as for **LINREG**. Note that there are no **WEIGHT** or **SMPL** options.

- **smpl=**SMPL series or formula (*User’s Guide*, Section 5.2)
  You can supply a series or a formula that can be evaluated across entry numbers. Entries for which the series or formula is zero or “false” will be skipped, while entries that are non-zero or “true” will be included in the operation. You must use the MAXL option with this.

- **maxl/[nomaxl]**
  Estimates the model using maximum likelihood estimation rather than conditional least squares. **MAXL** has the advantage of being able to handle missing values within the estimation range.

- **method=[gauss]/bfgs/simplex/genetic/initial/evaluate**
- **pmethod=gauss/bfgs/simplex/genetic/initial/evaluate**
- **piters=**number of PMETHOD iterations to perform [none]

**METHOD** sets the estimation method to be used, with Gauss-Newton being the default choice. **INITIAL** is the initial guess algorithm—the same one used by the RATS instruction **INITIAL**. **EVALUATE** simply evaluates the model given the initial parameter values (which you can input using the **INITIAL** option).
Use PMETHOD and PITERS if you want to use a preliminary estimation method to refine your initial parameter values before switching to one of the other estimation methods—RATS will automatically switch to the METHOD choice after completing the preliminary iterations requested via PMETHOD and PITERS.

*iterations* = iteration limit  [100]
*subiterations* = subiteration limit  [30]
*cvcrit* = convergence limit  [.00001]
*trace/[notrace]*

**BOXJENK** uses non-linear optimization. ITERATIONS sets the maximum number of iterations, SUBITERS sets the maximum number of subiterations, CVCRIT the convergence criterion. TRACE prints the intermediate results. See Chapter 7 in the User’s Guide for more details.

*initial* = VECTOR of initial guesses

The initial guess values used by RATS are usually adequate for well-specified models. However, if you are having trouble getting convergence, you can input your own guess values. The model coefficients are in the following order:

1. Constant.
2. Autoregressive.
3. Seasonal autoregressive.
4. Moving average.
5. Seasonal moving average.
6. Inputs in order, numerator first.

The option **INITIAL=%BETA** will start iteration from the point at which the previous **BOXJENK** left off, if you decide you simply want to let the process run for a few more iterations.

*define* = equation to define from result

If you intend to use the model for forecasting, you must use **DEFINE** to save the estimated equation. RATS obtains the equation from the model by multiplying out the autoregressive and input denominator polynomials. For instance, RATS converts the model:

\[
y_i = \frac{10}{1-.3L} x_i + \frac{1+.4L}{1-.5L} u_i
\]

into the equation

\[
(1-.3L)(1-.5L)y_i = 10(1-.5L)x_i + (1-.3L)(1+.4L)u_i, \text{ or }
\]

\[
y_i = .8y_{i-1} - 1.5y_{i-2} + 10x_i - 5x_{i-1} + u_i + .1u_{i-1} - .12u_{i-2}
\]

If you transformed your dependent variable or any input series with differencing operators prior to doing **BOXJENK**, or if they were residuals themselves from **BOXJENK** (prewhitened), then you must use **MODIFY** and **VREPLACE** on the equation to put it into a form directly usable by the forecasting instructions.
**inputs**=number of inputs [0]

**applydifferences/[noapplydifferences]**
Inputs are the number of transfer function inputs or intervention model dummy variable series. For each input, you include a supplementary card describing the polynomial associated with it. See the description of the supplementary cards below. If you use APPLYDIFFERENCES, RATS applies the differencing operators (DIFFS and SDIFFS options) to all the inputs. Although APPLY is not the default, you will usually use it with your inputs.

**regressors/[noregressors]**
This allows you to add extra (non-ARIMA) variables using standard regression format, rather than intervention style inputs via the INPUTS option. The REGRESSORS and INPUTS are mutually exclusive—you cannot use both at the same time. The APPLYDIFFERENCES option can be used with REGRESSORS.

**title="title for output" ["Box-Jenkins"]**
This option allows you to supply your own title to label the resulting output.

**derives=VECTOR[SERIES] for partial derivatives**
This saves the series of partial derivatives of the residuals. The first series in the VECTOR will be the partials with respect to the first parameter displayed in the BOXJENK output, the second series will be the partials with respect to the second parameter, and so on.

**Supplementary Cards**
If using the REGRESSORS option, supply any additional regressors in regression format. If using the INPUTS option, supply one supplementary card for each input. The lag polynomial form of an input is:

\[
\frac{(\omega_0 + \omega_1 L + \ldots + \omega_n L^n)}{(1 - \delta_1 L - \ldots - \delta_m L^m)} X_{t-d}
\]

Please note the sign convention (+ in numerator, – in denominator).

**series**
The input series. For an intervention model, this will be some type of dummy variable.

**numlags**
The number of numerator lags: \(n\) in the formula above. You can also use ||list of lags|| or a VECTOR of INTEGERS for non–consecutive lags. Note, however, that there will always be an \(\omega_0\) parameter.

**denlags**
The number of denominator lags: \(m\) in the formula above. Again, you can use ||list of lags||.

**delay**
The delay period for the series: \(d\) in the formula above. You can omit this if it is 0.
Missing Values

**BoxJenk** requires use of the MAXL option if there are any missing values within the data range.

Technical Information

The parameterization used for the ARIMA model in RATS is:

\[(1-L)^d(1-L^s)^e y_t = \alpha + \frac{(1+\theta_1 L + \ldots + \theta_q L^q)(1+\Theta_1 L^s + \ldots + \Theta_q L^{se})}{(1-\phi_1 L - \ldots - \phi_p L^p)(1-\Phi_1 L^s - \ldots - \Phi_p L^{sp})} u_t\]

where:
- \(y_t\) is the dependent variable
- \(u_t\) is the series of residuals
- \(p\) is the number of autoregressive coefficients
- \(\phi_n\) is the AR coefficient at lag \(n\)
- \(q\) is the number of moving average components
- \(\theta_n\) is the MA coefficient at lag \(n\)
- \(\alpha\) is the optional constant
- \(L^n\) is the lag (or backshift) operator at lag length \(n\)
- \(d\) is the number of differences
- \(s\) is the seasonal span
- \(e\) is the number of seasonal differences
- \(l\) is the number of seasonal MA components
- \(\Theta_n\) is the seasonal MA coefficients at lag \(n\)
- \(m\) is the number of seasonal AR components
- \(\Phi_n\) is the seasonal AR coefficient at lag \(n\)

Note the sign convention on the coefficients. Also, note that the parameterization of the constant term is different from that used in some other econometrics software.

If you are using the **INPUTS** option, polynomial terms of the form shown under “Supplementary Card” on the previous page are added to the right hand side of the above expression (one polynomial per input).

Algorithm

By default, **BoxJenk** uses the Gauss-Newton algorithm with numerical derivatives. (See the User’s Guide, Section 7.2). The simplex and genetic methods are also available via the **METHOD** option and **PMETHOD** options. These can be helpful in improving initial parameter guesses for models that provide difficult to fit. Maximum likelihood estimation is done using a state-space representation. Ansley’s (1979) conversion of the full-sample likelihood into a least-squares problem is employed if you use Gauss-Newton. **METHOD=BFGS**, though, often provides better estimation performance.
For transfer function models, RATS generates

\[ Z_t = \frac{X_t}{\partial(L)} \]

when denominator lags are present) by solving \( \partial(L)Z_t = X_t \), where presample values of \( X \) are set equal to the mean of the first twenty observations.

Hypothesis Tests

You can use \texttt{TEST} and \texttt{RESTRICT} to test hypotheses on the coefficients. The coefficient order is provided in the description of the \texttt{INITIAL} option.

Examples

\texttt{boxjenk(diff=1,sdiff=1,ar=1,ma=||1,12,13||,maxl,define=reseq) $ rescons}

estimates by maximum likelihood an ARIMA (1,1,3)x(0,1,0) with the MA parameters on lags 1, 12 and 13. This also defines the forecasting equation \texttt{RESEQ} from the result.

\texttt{boxjenk(diffs=1,ar=1,inputs=1,apply,define=saleseq) sales}

# ads 0 1 0

\texttt{boxjenk(diffs=1,ar=||2,4||,define=adseq) ads}

estimates a transfer function model from \texttt{ADS} to \texttt{SALES}. The transfer term is \( (0,1,0) \) and the noise term is \texttt{ARIMA}(1,1,0). The second \texttt{BOXJENK} fits an \texttt{ARIMA}(2,1,0) to \texttt{ADS} with AR lags on 2 and 4. Use the two equations together to forecast \texttt{SALES} out-of-sample.

\texttt{set change = t>=1974:3}

\texttt{boxjenk(diffs=1,sdiffs=1,sma=1,inputs=1,apply) assist}

# change 0 0 0

This is an intervention model. The time series model for \texttt{ASSIST} is \texttt{ARIMA} (0,1,0)x(0,1,1). The intervention is a change in the level of the series from 1974:3 on.

\texttt{dec vect[series] days(7)}
\texttt{do i=1,7}
\texttt{\hspace{1cm} set days(i) = %tradeday(t,j)}
\texttt{end do}
\texttt{boxjenk(regressors,sdiffs=1,ma=1,applydiff) logsales}
\texttt{# days}

The code above estimates a \texttt{REGARIMA} model with trading day counts as the exogenous variables.
Variables Defined by BOXJENK

In addition to the standard regression variables (see LINREG), BOXJENK defines:

- **%NARMA**: number of ARMA parameters (useful for degrees of freedom correction when computing a $Q$ statistic) (INTEGER)
- **%FUNCVAL**: final value of the estimated function (REAL)
- **%ITERS**: iterations completed (INTEGER)
- **%CONVERGED**: = 1 or 0. Set to 1 if the process converged, 0 if not.
- **%CVCRIT**: final convergence criterion. This will be equal to zero if the subiterations limit was reached on the last iteration (REAL).

Output

The output is the standard regression output, except for the labels on the coefficients, and the inclusion of the Ljung–Box $Q$ statistic. RATS labels the AR, MA, and seasonal parameters as AR, AR_SEAS, MA, and MA_SEAS. The numerator and denominator coefficients for input series are N_{nnnnn} and D_{nnnnn}. The $R^2$ is computed based upon the original dependent variable before differencing, if any is used. This instruction produces the output shown below:

```
boxjenk(ma=1,maxl,constant,inputs=1,define=saleseq) sales
# adv 1 1 0
```

```
Box-Jenkins - Estimation by Gauss-Newton
Convergence in  9 Iterations. Final criterion was  0.0000061 <=  0.0000100
Dependent Variable SALES
Usable Observations    35     Degrees of Freedom    30
Centered R**2   0.770528     R Bar **2   0.739932
Uncentered R**2 0.987865     T x R**2     34.575
Mean of Dependent Variable 24.602857143
Std Error of Dependent Variable  5.898478653
Standard Error of Estimate 3.008039840
Sum of Squared Residuals 271.44911030
Log Likelihood                 -85.87982
Durbin-Watson Statistic        1.978944
Q(8-1)                         2.720342
Significance Level of Q       0.90961236

<table>
<thead>
<tr>
<th>Variable</th>
<th>Coeff</th>
<th>Std Error</th>
<th>T-Stat</th>
<th>Signif</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. CONSTANT</td>
<td>14.510838586</td>
<td>2.109283706</td>
<td>6.87951</td>
<td>0.00000012</td>
</tr>
<tr>
<td>2. MA(1)</td>
<td>0.722780338</td>
<td>0.145590260</td>
<td>4.96448</td>
<td>0.00002575</td>
</tr>
<tr>
<td>3. N_ADV(0)</td>
<td>0.117441236</td>
<td>0.029083762</td>
<td>4.03803</td>
<td>0.00034391</td>
</tr>
<tr>
<td>4. N_ADV(1)</td>
<td>0.129543509</td>
<td>0.035887205</td>
<td>3.60974</td>
<td>0.00110205</td>
</tr>
<tr>
<td>5. D_ADV(1)</td>
<td>0.284528188</td>
<td>0.159772292</td>
<td>1.78084</td>
<td>0.08506872</td>
</tr>
</tbody>
</table>
```
Branch

BRANCH or GOTO — Branching to a Specific Instruction

You can accomplish most program control with instructions like IF, ELSE, WHILE, UNTIL, BREAK and NEXT. However, there are situations when it is simpler to jump directly to a specific instruction. This is the purpose of BRANCH. You can also use GOTO as a synonym for BRANCH.

```
branch  branch label
```

Parameters

branch label The label indicating the point in the program where you want to continue program execution. RATS permits forward and backward branching. This must be sixteen characters or less in length.

Placing Branch Labels in a Program

The BRANCH instruction and the branch label must be part of the same compiled section, although the BRANCH instruction can be “nested” deeper than the branch label, as in the example below. Labels must be unique within a compiled section, but you can use the same label in a different compiled section.

When labeling the location for the branch, you must prefix the branch label with a colon (:). Put at least one blank between the end of the label and the next instruction.

You cannot BRANCH into a block of statements ({}), IF-ELSE or any loop) from outside of the block. For instance, the following is illegal:

```
branch 100
do for xmat = amat bmat cmat
   :100 matrix xmat=inv(xmat)
end do for
```

Example

In the example below, there is no good alternative to BRANCH. We wish to break out of a pair of nested loops when some condition in the inner loop becomes true. A BREAK instruction would only break the J loop.

```
{
   do i=1,nvar
      do j=1,nlag
         @compval i j value
         if value>100.0
            branch done
         end do j
      end do i
   :done
      display "Result Achieved At Variable" i "Lag" j
}
```
BREAK — Breaking Control Out of Loops

**BREAK** forces an exit from the innermost loop (**DO**, **DOFOR**, **WHILE**, **UNTIL** or **LOOP**). Execution continues with the first instruction after the loop. **BREAK** is useful with **WHILE** and **UNTIL** if there is an alternate condition for exiting.

Because **BREAK** only exits the innermost loop, you must use **BRANCH**, rather than **BREAK**, to exit a more complex loop structure.

```
break (no parameters)
```

**Example**
This loops under control of an **UNTIL**. However, the program exits the loop and prints a warning message when the iteration limit is exceeded.

```RATS
compute rhoold=1.0, iter = 0
until abs(rhoold-rho)<.0001
 {
   compute iter=iter+1
   if iter>maxiters {
      display "Iteration limit exceeded"
      break
   }
   compute rhoold=rho
   @comprho rho
}
display "End of loop"
```

*Execution continues here after BREAK*
CACCUMULATE: Partial Sums of Complex Series

Accumulates the complex series \textit{cseries}: entry \textit{n} of the accumulated series \textit{(newcseries)} is equal to the sum of the first \textit{n} entries of \textit{cseries}. The most common use of \textbf{CACCUM} is the Durbin cumulated periodogram test (see Durbin (1969) and \textit{User’s Guide}, Section 6.5).

\begin{verbatim}
caccumulate( option ) cseries start end newcseries newstart
\end{verbatim}

**Parameters**

- \textit{cseries} \hspace{1cm} The complex series to transform.
- \textit{start} \textit{end} \hspace{1cm} Range of entries to transform. By default, range of \textit{cseries}.
- \textit{newcseries} \hspace{1cm} Complex series for the result. By default, same as \textit{cseries}.
- \textit{newstart} \hspace{1cm} Starting entry of the result series. By default, same as \textit{start}.

**Options**

\textbf{standardize/[nostandardize]}

If you use \textbf{STANDARDIZE}, \textbf{CACCUMULATE} normalizes the resulting series so the final entry has the value one. This option was called \textbf{SCALE} in earlier versions.

**Example**

This is the business end of a Durbin test. The \textbf{CACCUMULATE} instruction cumulates the periodogram (squared Fourier transform) over frequencies 0 to \(\pi\) and scales it to have an end value of 1.0. If the series examined is white noise, this cumulated periodogram should differ only slightly from the theoretical spectral distribution function for white noise: a straight line. The \textbf{CXTREMUM} instruction computes the maximum gap between the two distribution functions.

\begin{verbatim}
frequency 3 nords
compute half=(nords+1)/2
rtoc
# resids
# 1
clabels 1 2 3
# "Cumprdgm" "Whitenoise" "Gap"
fft    1
cmult  1 1
\textit{cacc(scale)} 1 1 half
\textit{cunits} 2
\textit{cacc(scale)} 2 1 half
\textit{cssubtract} 2 1 1 half 3
\textit{cxt(part=absval)} 3 1 half
\end{verbatim}

\textit{Periodogram of resids}

\textit{Spectral density of white noise (constant)}

\textit{Computes extreme values of series 3}
CALENDAR — Setting the Date Information

CALENDAR sets the periodicity and starting date for the current work space. Version 7 introduces a simplified format for CALENDAR. The older variations are still supported, and are documented at the end of this section. Do not use a CALENDAR when working with cross section data, or data with no regular date pattern.

Common Frequencies

- calendar(a) year:1
- calendar(q) year:quarter
- calendar(m) year:month
- calendar(b) year:month:day
- calendar(w) year:month:day
- calendar(d) year:month:day
- calendar(7) year:month:day

For Annual (A), Quarterly (Q), Monthly (M), Bi-Weekly (B), Weekly (W), and Daily data (D=five day per week, 7=seven days per week). The parameter specifies the date of the first entry, in standard RATS date notation.

Other Date Schemes

- calendar(ypp=years per period) year:1
  For data with multiple years per period.

- calendar(ppy=periods per year) year:period
  For data with a fixed number of periods per year.

- calendar(ppw=periods per week) year:month:day
- calendar(ppd=periods per day, other options) year:month:day
  Use PPW for data with a set number of periods per week (other than 5 or 7). Use PPD when you have multiple observations per day.

- calendar(dpp=days per period) year:month:day
  For data with a fixed number of days per period.

- calendar(panelobs=numperiods, other options) startdate
  For panel data. If desired, you can use any of the other options shown above except PPD to specify the periodicity of the panel data.

- calendar(irregular)
  For erratic time series—indicates that you have time series data, but without a specific date scheme.

- calendar(recall=saved calendar)
  to recall an earlier calendar setting
Wizards

The Calendar wizard on the Data menu allows you to set the CALENDAR using a dialog box. Both of the Data wizards use the same dialog box to set CALENDAR as part of the process of reading in data.

Description

CALENDAR is one of the most important instructions in RATS:

• With time series data, use CALENDAR to tell RATS the starting date and periodicity of the data.
• With panel data, use CALENDAR to set the number of time periods per individual, and, optionally, the starting date and periodicity of the data.

After setting a CALENDAR, you can refer to entry ranges using an easy-to-understand date notation, and your output will be labeled similarly. Note that you should never use a CALENDAR instruction with cross-sectional data. It will only confuse RATS.

Date Parameter

The date parameter sets the starting date for the CALENDAR. This will be the date associated with the first entry of your data series. Use the standard RATS date format notation, described on the next page.

Frequencies

The form of the CALENDAR instruction will vary depending on the desired frequency, as described in the following paragraphs. See the examples later in this section.

Annual, Quarterly, Monthly

Use the A, Q, or M options, respectively, to choose one of these frequencies.

Weekly and Biweekly Data

Use the W or B options for weekly or biweekly data.

The date parameter should be the last day of the first period. For example, if the week ends Friday, give the date of the first Friday, not the first Monday. This affects how RATS transforms data between frequencies. If, for instance, you read daily data into a weekly work space with weeks ending on a Friday, the value for a given week will depend on the daily value for that Friday and (up to) six preceding days.

Daily Data (5, 7, or other number of periods per week)

Use the D option for a daily calendar with five days per week (Saturdays and Sundays are omitted), or the 7 option for a daily calendar with entries for all seven days.

For a less standard periods per week, use the PPW option. For example, if you have six day per week data starting on June 5, 2000, use: cal(ppw=6) 2000:6:5
**Intraday Data**

If you have a data set with a *fixed* number of time periods *per day*, use the `PPD` option to set the periods per day, combined with one of the other options (usually `D` or `7`) to describe the day-to-day arrangement of the data set.

You can use any number of periods with `PPD`—you do not have to cover a full 24 hour period. For instance, data at five minute intervals from 9:05am to 3:00pm could be handled using `PPD=72`.

**Other Periodic Schemes**

Use the `PPY` (period per year) option for data with a set number of periods per year (other than monthly or quarterly, which you would handle with the `M` or `Q` options). Use `DPP` (days per period) for frequencies specified as a specific number of days.

**Data Recorded Less Than Once a Year**

For data which is equally spaced, but more than one year apart, use the `YPP` (years per period) option to set the number of years per period. The date parameter sets the starting year of the first observation.

**Other Time Series Data**

Use `CALENDAR(IRREGULAR)` for any time series data which does not fit into any of the above. Note, in particular, the requirement in all of the above for *fixed* numbers of periods or period lengths. The only point of `CAL(IRREGULAR)` is to inform RATS that the data set *does* have time series properties. In all other ways, there is no difference between `CALENDAR(IRREGULAR)` and omitting `CALENDAR` entirely.

**Panel Data**

Include the option `PANELOBS=periods per individual` together with other options to describe the date scheme in the time direction. `periods per individual` is the number of time periods for each individual in a data set. The setting of the time series scheme is optional and has no effect on your ability to use the special panel data features of RATS. Note that you *cannot* use `PANELOBS` with the `PPD` option. See Chapter 14 of the *User's Guide* for more on panel data.

**RATS Date Notation**

You refer to observations by date in RATS using one of the following formats:

- `year:period` annual, monthly, quarterly, other periods per year, or years per period. For example, for monthly data, `2006:12` is December, 2006. For annual data, `2003:1` is the year 2003.

- `year:month:day` weekly, daily, etc. (`2005:5:15` for May 15, 2005).

- `year:month:day//period` intraday data. (`2000:12:1//3` for the third observation on December 1, 2000).
individual//date dated panel data (2/2005:4 for the April, 2005 observation of the second individual and 1/2001:2:1 for the February 1, 2001 observation of the first individual).

individual//period dated or undated panel data.

You can also use “year:period” for weekly or daily data; it will give you the indicated week or day (business day for business day data) within the year. With seven-day data, 2005:35 would be the February 4th observation (35th period for that year).

Also, with annual data, you must use “year:1”, and not just “year”, for a date, as RATS won’t recognize it as a date except in that form.

Century and “Year 2000” Issues

RATS allows you to use two digits for year references in a date. However, for backwards compatibility with older RATS code, the program assumes that these refer to 20th century dates (19xx). To avoid confusion and mistakes in reading your data, we strongly recommend you use four-digit year references in all of your programs and data sets.

If you still have existing data sets (such as RATS format files) that have been created using two-digit date formats, define a CALENDAR using a four-digit year, read the data into RATS, and then write the data back out—this will update the file so that it contains four-digit year information.

Examples

The first three examples all use ALLOCATE to set the default ending period:

```
calendar(m) 1950:1
allocate 2002:12

cal 1791:1
allocate 1856:1
    Annual data, beginning in 1791, ending in 1856. Note the :1 on the date references is required when specifying dates for annual data.

cal(w) 1985:5:8
allocate 1989:3:1
    Weekly data, with the first week ending on May 8, 1985. Data end March 1, 1989.
```
Daily data with seven days per week, beginning on January 1, 1980, and set the
default ending period (December 31, 1999) via a **DATA** instruction.

Eight time periods per business day, starting July 1, 2000, ending on August 31,
2003 (with a full eight periods on the last day).

Panel data at a weekly frequency, with 52 time periods per individual, first week
ending Friday, January 5, 1990. There are 30 individuals in the data set.

**Holidays and Related Matters**

RATS expects business day data to have five periods per week (except, perhaps, for
the first and last weeks). What do you do if you have a data set which has no obser-
vations for holidays? There are three ways to deal with this:

1. You can treat it as **IRREGULAR** data. Use this if you want a continuous
   stream of data with no gaps (for example, for recursive models such as ARCH
   and GARCH which can’t handle missing values).

2. Use a business day **CALENDAR**, and either insert missing values in the data
   file for the missing observations or, if your file includes dates, let the **DATA**
   instruction take care of “padding” the data set by inserting missing values for
   any days that aren’t included in the source file. This is the best choice when
   using regression techniques where missing values can simply be dropped
   from the sample. However, if the insertion of missing values poses a problem
   for the types of analysis you wish to do, use one of the other two approaches.

3. The third option is to read in the data with gaps as in (2) above, but use
   **SAMPLE**(SMPL=series) instructions to produce “compressed” versions of the
   series, with the gaps removed. You can use the full series where possible, and
   use the compressed versions as needed for the kinds of analysis discussed in
   (1) above. See **SAMPLE** for details.
Converting Frequencies

RATS will not allow you to work with two frequencies of data simultaneously, meaning that only one CALENDAR can be in effect at a time. However, the DATA instruction can translate data automatically from other frequencies to the current CALENDAR frequency (see User’s Guide, Section 2.3 for details). Thus, if you have a mixture of monthly and quarterly data, you can work with it all either at quarterly or at monthly frequencies.

DATA itself does not use any sophisticated techniques for producing a higher frequency version of a series. For instance, in translating quarterly data from a file into monthly series, it merely copies the quarterly value to each of the corresponding months. We have provided procedures (on the files INTERPOL.SRC, DISTRIB.SRC, and CHOWLIN.SRC) to do more complex interpolations and distributions.

Saving/Recalling Calendars

It’s sometimes necessary to switch to a different frequency temporarily, or to switch out of time-dated data to simple sequence data. You can save the current calendar setting, then recall it later on using the function %CALENDAR() and the RECALL option on the CALENDAR instruction. The following saves the initial quarterly calendar setting in SAVECAL, switches to IRREGULAR, then switches back later.

```
calendar(q) 1954:1
...
compute savecal=%calendar()
calendar(irregular)
...
calendar(recall=savecal)
```

Functions . . .

The following functions can be applied after you’ve done a CALENDAR. Most take as argument an entry number and return information about the date for that entry within the CALENDAR scheme that you’ve set. One uses different arguments, and some take no arguments at all. See Section 2 of the Reference Manual for details on these.

- %MONTH() 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() returns the output date label, such as “2001:1:31”

%DAY() returns the day of entry n

%DAYCOUNT() returns the number of days in entry n. For example, %daycount(2005:2) returns 28.

%DOW() 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 nth 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() returns the individual's number for an entry in a panel data set.

%JULIAN() 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.

%PERIOD() 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() 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() returns the day of the week (1 = Monday, 2=Tuesday, ...)

%YEAR() returns the year of entry n
# Calendar

## Deprecated CALENDAR Formats

The following older formats for specifying the `CALENDAR` instruction are still supported in Version 7, but we recommend using the new formats for any new tasks. Note that most of these use multiple parameters (separated by at least one blank space), rather than date format notation, to supply the starting date.

- **calendar** `year  period  peryear`
  for annual, semi-annual, monthly or quarterly data or any other frequency with a fixed number of entries per year

- **calendar**(daily or sevenday) `year  month  day`
  for daily data with five days per week or seven days per week

- **calendar**(entriesperweek=perweek) `year  month  day`
  for data with the indicated number of entries per week

- **calendar**(weekly or biweekly) `year  month  day`
  for weekly or biweekly data

- **calendar**(days=period) `year  month  day`
  for data every period days

- **calendar**(yearsperentry=years) `year`
  for data spaced the indicated number of years apart

- **calendar**(perday=numper, other options) `year  month  day`
  for intraday data
CATALOG — Listing the Contents of a RATS Data File

CATALOG lists the contents of the open RATS format data file. It can display series names only, or the full information for each series: names, dates and comments.

catalog( options )   list of data file series   (optional)

Wizard

If you do the File-Open RATSData operation, you’ll get the same information, but in a scrolling list. From this list, you can view/edit data, export data, and more.

Parameters

list of series   If you provide a list of series names, CATALOG will display information only for those series. Otherwise, RATS will list all the series on the file.

Options

full/[nofull]

FULL requests full information (names, dates, comments) on each series in the list. Otherwise CATALOG lists names only. FULL is the default only if you ask for information on specific series by using the list parameter.

unit=[output]/copy/Other unit

Choose where you want output to go.

like=template string for series to list

LIKE allows you to list information only for series that match a template you supply. You can use the standard wildcard characters “*” and “?”. For example, LIKE="X*" will list all series whose name begins with X, while LIKE="X?" lists only those series whose names are two letters long, with the first letter being X.

Examples

catalog

Lists the names of all the series on the file.

catalog(full,unit=copy)

Lists full information on all series to the current COPY unit.

catalog   real_gnp   nom_gnp

Lists full information on the series REAL_GNP and NOM_GNP

See Also . . .

DEDIT   Opens a RATS data file for editing.
PRTDATA   Prints series from a RATS data file.
CDF — Computing Marginal Significance Levels

**CDF** computes and displays the marginal significance level of a statistic from one of four distributions: the $F$, the $t$, the $\chi^2$ and the standard normal. The functions %FTEST($x$, $m$, $n$), %TTEST($x$, $n$), %CHISQR($x$, $d$) and %CDF($x$) compute the same significance levels as **CDF**, but **CDF** can display the results as well.

```
cdf( option ) distribution statistic degree1 degree2
```

**Parameters**

- **distribution**: The distribution selected. Choose (at least three letters) of
  - FTEST (for $F$)
  - TTEST (for two-tailed $t$)
  - CHISQUARED
  - NORMAL (for two-tailed standard normal).

  With **TTEST** and **NORMAL**, divide the significance level by two if you want just a one-tailed test.

- **statistic**: The value of the test statistic. This can be an expression.

- **degree1, 2**
  - For $t$ and $\chi^2$, **degree1** is the degrees of freedom.
  - For $F$, **degree1** and **degree2** are the numerator and denominator degrees of freedom, respectively.

**Option**

```
[print]/noprint
```

RATS displays the statistic and significance level unless you use **NOPRINT**.

```
title="string for output title"
```

You can use the **TITLE** option to include information in the output to identify what is being tested.

**Example**

This does a simple specification test by regressing the residuals on a larger set of exogenous variables. See Section 6.9 in the *User's Guide* for more.

```
linreg lwage / resids
# constant exper expersq educ
linreg resids
# constant exper expersq educ age kidslt6 kidsge6
cdf(title="Specification Test") chisqr %trsq 3
```

**Variables Defined by CDF**

- **%CDSTAT**: the computed test statistic (real)
- **%SIGNIF**: the marginal significance level (real).
CHANGE — Switching I/O Units

CHANGE makes one I/O unit take over the role previously assigned to another unit. It can be used with the standard RATS I/O units, and with user-defined I/O units created with OPEN instructions. For reasons discussed below, CHANGE is only useful in a few limited situations.

```
change oldunit newunit
```

Parameters

- **oldunit**: Name of the I/O unit that you want to set. It makes little sense for this to be anything other than INPUT, OUTPUT or PLOT.

- **newunit**: The existing I/O unit that you want to take the place of oldunit.

Description

RATS makes newunit take over the role currently assigned to oldunit. For instance,

```
change input keyboard
```

will switch the source of RATS instructions from whatever it is currently (probably a file) to the keyboard.

```
change output screen
```

switches output to the screen or edit window.

Standard RATS I/O Units

The standard RATS I/O units are INPUT, OUTPUT, DATA, COPY and PLOT. Predefined “hardware” units are KEYBOARD, SCREEN and PRINTER (PRINTER may not always be available). You can also use your own names (see OPEN for details). It makes little sense for newunit to be DATA or COPY as any instruction which might use either of those units has a UNIT option which allows you to set the unit directly.

Also, note that the SOURCE instruction (which tells RATS to read and execute commands stored on an external file) is generally superior to using CHANGE INPUT.

Examples

Suppose you are running a long batch job, and you want to direct some output (such as regression results) to one file, and other output (such as hypothesis test results) to another file. If you only need to change output files once, you could simply use two OPEN OUTPUT commands:
open output regress.out
linreg ...
open output hypotest.out
restrict ...

However, if you need to switch back and forth between the files, as in a **DO** loop, you would have to use the **CHANGE** instruction, as **OPEN OUTPUT** erases the previous contents of the file that is opened. For example:

```plaintext
open regfile regress.out
open testfile hypotest.out
do i=1,10
   change output regfile
   linreg ...
   change output testfile
   restrict ...
end do
```

Note that you use the *unit* names defined by the **OPEN** command on the **CHANGE** command, not the *file* names.

A similar example is where you have an important piece of output which you would like to place in its own window, so that it can be found easily. This can be done by something like

```plaintext
open(window) regwindow "Key Regressions"
change output regwindow
linreg ...
....
change output screen
```

This opens a new window (which will be titled on the display as “Key Regressions”) and puts all output created by the **LINREG** and any other instructions down to the next **CHANGE OUTPUT** into that window. That second **CHANGE OUTPUT** switches the remaining output back to the main output window.
CLABELS: Labeling Complex Series

CLABELS attaches an output label to a numbered complex series. RATS will use these labels to identify series on any output. By default, labels are No Label(n) where n is the series number.

Any number of series may share a label. Labels aren’t subject to the restrictions put on symbolic names, so you can use any combination of up to sixteen characters.

```
clabels list of complex series
# labels within "..." separated by blanks
```

Parameters

- list of series  List of complex series you want to label.

Supplementary Cards

The labels can be any collection of characters (up to sixteen) enclosed within quotes ("...") or '...'). You can also use string expressions, LABEL variables, or elements of an array of LABELS.

Example

```
freq 5 512
clabels 4 5
# "Crossspec" "Spectrum"
```

labels series 4 and 5 as CROSSPEC and SPECTRUM, respectively.
CLEAR — Clearing a Data Series

CLEAR resets or creates one or more series, setting all of their entries to the missing value code (%NA). This prevents old, unwanted data from being included in statistical analyses later on.

```
clear( options )  list of series
```

Parameters

- `list of series` The list of the series you want to clear. Use the `ALL` option to clear all existing series.

Option

- `all/[noall]` If you use `ALL`, RATS clears (sets to missing values) all existing series.

Notes

CLEAR can be very helpful when you want to set a new series in an unconventional way, such as with `COMPUTE` instructions. You cannot refer to particular entries of a series in an expression until you’ve created the series and defined it over some range. CLEAR provides a convenient way of doing this.

Example

```
dec vector snames
open data sicnames.dat
read(varying) snames
open data sic.rat
do i=1,%rows(snames)
   clear series
   labels series
   # snames(i)
   data(format=rats) / series
   ....
end do i
```

This reads in a list of series names from the text file SICNAMES.DAT. The named series are then processed sequentially. CLEAR cleans up SERIES each time through, so only the new data will be used in the (unspecified) analysis.

This is a convenient way to deal with very large sets of series, provided you only need them one at a time. See Section 16.3 in the User’s Guide for some other suggestions.
CLN, CEXP and Related Instructions

These instructions do the following operations: complex log, complex exp, complex square root, conjugation, entry copying and scaling by constant, respectively. Note that while CLN and CSQRT are complex-valued, you will probably apply them only to series with zero imaginary parts, such as spectral densities.

```
cln     cseries   start   end   newcseries   newstart
cexp    cseries   start   end   newcseries   newstart
csqrt   cseries   start   end   newcseries   newstart
conjugate cseries   start   end   newcseries   newstart
cmve    cseries   start   end   newcseries   newstart
cscale(option) cseries   start   end   newcseries   newstart
```

Parameters

- **cseries** The complex series to transform.
- **start** **end** Range of entries to transform. By default, the defined range of `cseries`.
- **newcseries** Complex series for the result. By default, same as `cseries`.
- **newstart** Starting entry of the result series. By default, same as `start`.

Option (for CSCALE only)

```
scale=real expression
```

CSCALE multiples each entry of `cseries` by the value you specify with this option.

Example

```
cln 3
ift 3
cset 3 = %z(t,3)*(t<=ordinate/2)
fft 3
cexp 3
ift 3
cset(scratch) 3 = %z(t,3)/%z(1,3)
```

is a section of the procedure SPECFORE, which computes forecasts using spectral methods. It takes the log of series 3 (a spectral density), applies an inverse transform to the series, sets its negative “lags” to zero, transforms the series back, then exponentiates the series and inverse transforms it again. The CSET instruction at the end normalizes series 3 to a value of 1.0 in entry 1.
CLOSE — Closing an Open I/O Unit

CLOSE closes an open I/O (input/output) unit. Because RATS automatically closes any open files at the end of a session, you will rarely need to close a file yourself. CLOSE is necessary only if you want to read data from a file to which you have written earlier, or want to read data into another program (like a spreadsheet) before you exit RATS. You must CLOSE the file before re-OPENing it for input.

```
close  RATS I/O unit
```

Parameters

**RATS I/O unit**  This can be COPY, PLOT, DATA or any other unit which you have defined.

Examples

This is a (section of) a two–part program. The first runs 100,000 replications of some testing procedure, producing values for the scalar variables CDSTAT1 and CDSTAT2. The DISPLAY instruction writes each pair of values to file SIMUL.DAT. When the replications are finished, the first program is terminated, and a second program reads this information back in as data series for analysis.

```
open copy simul.dat
do i=1,100000
...  
display(unit=copy) cdstat1 cdstat2
end do i
*
end(reset)
*
close copy
open  data simul.dat
all 100000
data(org=obs) / cdstat1 cdstat2
order cdstat1
order cdstat2
...
```

See Also . . .

**OPEN**  Opens an I/O unit (includes general discussion of RATS I/O units).

**REWIND**  Rewinds an I/O unit.
CMASK: Creating Seasonal Masks

**CMASK** creates a *seasonal mask*: a complex series with value one everywhere except in bands about the seasonal frequencies.

```
cmask cseries start end seasonalwidth bandwidth bandcenter
```

### Parameters

- **cseries**: Series to set.
- **start end**: Range of entries to set. By default, 1 and `FREQUENCY` length.
- **seasonalwidth**: Number of entries between seasonals. By default, RATS will use the number of frequencies divided by the `CALENDAR` seasonal.
- **bandwidth**: The width of the zero band around each seasonal frequency. RATS rounds even values up to the next odd value. By default: `seasonalwidth/6`.
- **bandcenter**: The center (entry number) of the first seasonal band. By default, entry one (zero frequency). If this is `seasonalwidth+1`, **CMASK** will leave the low frequencies unaffected.

### Description

**CMASK** creates a mask for seasonal frequencies. It sets entries `start` to `end` of `cseries` to the value 1.0. Then it sets to zero a band of width `bandwidth` about every `seasonalwidth` entry, beginning with `bandcenter`. A band will wrap around to the other end of the series if necessary. For instance:

```
cmask 3 1 128 32 5 33
```

results in zeros in entries 31 to 35, 63 to 67 and 95 to 99. These are the bands around $\pi/2$, $\pi$, and $3\pi/2$.

### Notes

You need to take some special steps when smoothing a spectrum which you intend to mask: use the option `MASK=masking series` on `WINDOW`, then multiply the smoothed series by the mask.

```
fft 2
cmult(scale=1./(2*%pi*%scaletap)) 2 2
cmask 3 1 128 32 5 1
window(mask=3,width=9) 2 1 128 4
cset 4 = %z(t,4)*%z(t,3)
```
CMoment

CMOMENT — Cross-Moment and Correlation Matrices

CMOMENT computes the cross-moment (X’X) matrix of a set of series. With the CORR option, it computes their correlation matrix.

cmoment(options) start end
# list of series in regression format (omit with EQU, LASTREG, MODEL, or INST options)
# second list of variables (only if using ZMATRIX option)

Description

Since cross-moment and correlation matrices are frequently important in statistical work, CMOMENT has significant value by itself. However, CMOMENT is most useful when combined with a LINREG instruction using the CMOM option. When you use LINREG with CMOM, LINREG does not compute the X’X and X’y arrays. Instead, it gets that information from the matrix you create with the CMOMENT. This has several important uses:

• You can create many types of ridge or Bayesian estimators by altering the cross-moment matrix before going to LINREG. See User’s Guide, Section 5.12.
• You can cut down on the computation time required for doing large repetitive regressions. The single CMOMENT instruction can compute all the cross-products necessary for later regressions. See, for example, the information criteria searches in Example 5.5 in the User’s Guide.

The arrays produced by CMOMENT are also used by the several functions described in Section 13.4 of the User’s Guide for making random draws from posteriors.

Parameters

start end Range of entries to use in computing the cross-moment matrix. If you haven’t set a SMPL, this defaults to the largest interval over which all the variables are defined.

When you do a LINREG(CMOM), RATS uses the start to end range used in the CMOMENT instruction. Any SMPL and WEIGHT options, if any, are also carried over. You cannot alter these when you do the regressions.

Supplementary Card

The first supplementary card lists the variables to include in the cross-moment matrix. You must include in this list the dependent variable(s) of any regressions that you base on this CMOMENT. Omit this if using the EQUATION, LASTREG, MODEL, or INSTRUMENTS option.

The second card is used only when using ZMATRIX.
Options

print/[noprint]
Use PRINT to print the cross-moment matrix. RATS prints it in a table like that used for the covariance matrix of regression coefficients (see the User’s Guide, Section 5.1).

smpl=SMPL series or formula (User’s Guide, Section 5.2)
You can supply a series or a formula that can be evaluated across entry numbers. Entries between start and end for which the series or formula is zero or “false” will be omitted from the computation.

spread=series of residual variances (Section 5.4)
This lets you provide a series with residual variances for weighted least squares.

corr/[nocorr]
center/[nocenter]
With CORR, CMOMENT computes the correlation matrix instead of the cross-moment matrix. With CENTER, it computes a centered (means subtracted) cross-moment matrix. You cannot use a LINREG(CMOMENT) instruction after CMOMENT with either of these.

equation=equation supplying variables
lastreg/[nolastreg]
model=model supplying variables
instruments/[noinstruments]
[depvar]/nodepvar
These provide shortcuts for using standard sets of variables. MODEL takes the explanatory variables from a model with linear equations; EQUATION takes these from a single equation, and LASTREG uses the explanatory variables from the last regression (or similar instruction). INSTRUMENTS uses the current set of instruments as the list. If you include any of these options, omit the supplementary card.

RATS will form the cross-moment matrix using the explanatory variables, and will include the dependent variable of the equation, regression, or model unless you use the option NODEPVAR.

Note: With LASTREG, the sample range is determined by the CMOM instruction (that is, by the default range, the start and end parameters, or the SMPL option), which may not be the same as the range used in the preceding regression. To use the same range as the regression, use the %regstart() and %regend() functions on the start and end parameters of the CMOM instruction.

zxmatrix=RECTANGULAR Z’X matrix [unused]
Takes two sets of supplementary cards, or some combination of LASTREG/EQUATION and INSTRUMENT options and possibly a supplementary card. It computes
CMoment

\[ Z'X \] where \( Z \) is the first list of variables and \( X \) the second. If you use LASTREG or EQUATION, that will always be the first list; INSTRUMENT will be first if you use it but not LASTREG or EQUATION. If you only use one of these options, use a single supplementary card to supply the other list of variables. You cannot use a LINREG(CMOMENT) instruction after CMOMENT with the ZMATRIX option.

\texttt{matrix=}SYMMETRIC array for computed matrix [%CMOM]

This saves the computed cross-moment matrix in the SYMMETRIC array. By default, the result goes into the array \%CMOM, which you can access. You cannot use a LINREG(CMOMENT) instruction after a CMOM using the MATRIX option.

\texttt{setup/ [nosetup]}

SETUP creates and dimensions the \%CMOM array and prepares for future LINREG(CMOM) instructions, but does not actually compute the matrix. Instead, you can fill the array yourself with subsequent instructions, such as COMPUTE.

Notes

Correlations computed by CMOM(CORR) may be different from pairwise correlations computed with the function \%CORR or with CROSS. CMOMENT does the computations over a single set of entries appropriate for all the variables involved. This may not be the same as the entries which could be used for a single pair of the variables.

Missing Values

Any observation for which any of the series in the cross-moment matrix is missing is dropped from the sample.

Examples

\texttt{cmom}

\begin{verbatim}
# constant m1{0 to 12} gnp
do lag=1,12
   linreg(cmoment) gnp
   # constant m1{0 to lag}
end do lag
\end{verbatim}

computes a set of distributed lag regressions. This is similar to what you would get with \texttt{LINREG}s alone except the regressions are computed over a uniform sample period, because the CMOM is computed using the full 12-period lag length for M1.

\texttt{cmom(corr,print)}

\begin{verbatim}
# rate30 rate60 rate90 rate120 rate1yr rate2yr
\end{verbatim}

computes and prints the correlation of six interest rate series.

\texttt{cmom 1974:2 *}

\begin{verbatim}
# uscpi{1} exrat{1} itcpi{1} uscpi exrat italcpi
compute s=%sweeptop(%cmom,3)
\end{verbatim}

“sweeps out” the lag values. This, in effect, runs a one lag VAR.
cmom

# constant shortrate{0 to 24} longrate
linreg(cmom) longrate
# constant shortrate{0 to 24}
compute beta=%beta

* * Draw residual precision conditional on previous beta *
* 
compute rssplus=nu*s2+%rsscmom(%cmom,beta)
compute hu =%ranchisqr(nu+%nobs)/rssplus
* 
* * Draw betas given hu *
* 
compute beta=%ranmvpostcmom(%cmom,hu,priorh,priorb)

uses the information in %CMOM to make random draws for the regression variance (done with the help of %RSSCMOM) and coefficients (%RANMVPOSTCMOM).

Variables Defined by CMOMENT

%CMOM Cross-moment matrix (SYMMETRIC)
%MEANV Vector of means of the variables (VECTOR)
%NCMOM Number of Variables (integer)
%NOBS Number of Observations (integer)

See Also . . .

%CORR Means-subtracted correlation between two vectors or series.
%COV Means-subtracted covariance between two vectors or series.
CROSS Computes cross-correlations or covariances of two series
CORRELATE Computes autocorrelations of a series
CMultiply

CMULTIPLY and related instructions

CADD, CSUBTRACT and CDIVIDE are fairly straightforward. They perform addition, subtraction (of cseries2 from cseries1) and division (of cseries1 by cseries2) of complex series. CMULTIPLY is a critical instruction: it multiplies cseries1 by the complex conjugate of cseries2, entry by entry. This operation comes up constantly in spectral analysis.

<table>
<thead>
<tr>
<th>cmul(option)</th>
<th>cseries1</th>
<th>cseries2</th>
<th>start</th>
<th>end</th>
<th>newcseries</th>
<th>newstart</th>
</tr>
</thead>
<tbody>
<tr>
<td>cadd</td>
<td>cseries1</td>
<td>cseries2</td>
<td>start</td>
<td>end</td>
<td>newcseries</td>
<td>newstart</td>
</tr>
<tr>
<td>csubtract</td>
<td>cseries1</td>
<td>cseries2</td>
<td>start</td>
<td>end</td>
<td>newcseries</td>
<td>newstart</td>
</tr>
<tr>
<td>cdivide</td>
<td>cseries1</td>
<td>cseries2</td>
<td>start</td>
<td>end</td>
<td>newcseries</td>
<td>newstart</td>
</tr>
</tbody>
</table>

Parameters

cseries1, 2 The pair of complex series you want to transform.
start end Range of entries to transform. By default, the common defined range of cseries1 and cseries2.
newcseries Complex series for the result. By default, same as cseries1.
newstart Start of the result series. By default, same as start.

Option (only for CMULTIPLY)

scale=real expression to scale result
Use this option to multiply all entries of newcseries by a value.

Comments

CMULTIPLY is the instruction generally used to convert a Fourier transform into a spectrum and cross-spectrum. The proper scaling factor for spectral estimates is SCALE=1./(2*%PI*N), where N is the number of actual data points and not the padded length.

If you use a taper (see TAPER), you should use the %SCALETAP variable defined by TAPER instead of N.

Examples

cmult(scale=1./(2*%pi*nobs)) 1 1
cmult 1 2 / 3

The first replaces series 1 by its squared absolute value (series 1 times its complex conjugate), divided by 2\pi NOBS. The second sets 3 equal to the product of 1 and the conjugate of 2.
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.

\[ \text{cntrl(options)} \quad \text{pause message} \]

Parameter

\text{pause message} \quad \text{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]

\[ \text{CNTRL(PAGE)} \] 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 \text{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.
Compute

COMPUTE — Evaluating Scalar and Matrix Expressions

COMPUTE is one of the most important and frequently used instructions in RATS. It evaluates an expression and stores the result in a variable. COMPUTE can operate on most of the variable types supported by RATS.

\[
\text{compute} \quad \text{variable} = \text{expression}
\]

Description

Among its uses in typical RATS programs:

- You can do calculations using values such as the Residual Sum of Squares (%RSS variable) produced by RATS instructions like LINREG and AR1.
- You can set integer values for starting and ending observations, then use these variables as parameters on RATS instructions. You can then easily alter the program to use a different range of entries by changing just a few COMPUTE instructions.
- You can do matrix calculations such as multiplication, addition, inversion and determinants.

With COMPUTE, you can do just about any numerical computation you could do with a high level programming language such as FORTRAN, C, Basic, or Pascal, and often much more efficiently, since RATS can work with entire matrices.

Parameters

- variable. This can be almost any type of variable supported by RATS, except FRML, which must be set using the instruction FRML. It can be an element of a series, but not an entire series itself—use SET to set multiple elements of a series with a single instruction.

- expression The expression can include any legal combination of arithmetic and matrix operators, functions, numeric values, character literals, and previously defined variables and arrays. If the variable... is of a known type, the expression must evaluate to a type which can be converted to the variable's type.
Variables, Data Types, and Type Modifiers

Every variable in RATS will be of a certain data type (real, integer, series, etc.), as described in Section 4.3 of the User’s Guide.

- If you introduce a new variable with a COMPUTE instruction, you can specify the data type of the variable using a type modifier. A type modifier is simply the name (or three-letter abbreviation) of one of the RATS data types, listed inside brackets ([ and ]) before the variable name.
- You can also declare the variable ahead of time using a DECLARE statement.
- If you haven’t declared the variable ahead of time, and you don’t use a type modifier, RATS will determine the data type of the new variable based on the result of the expression. For example, the instruction `COMPUTE A = 5.95` would define the variable `A` as a REAL because 5.95 is a real value. You have to be careful here not to be misunderstood. `COMPUTE SUM=0` will make `SUM` an INTEGER, while `COMPUTE SUM=0.0` will make it a REAL.
- When setting an array, you will not, in most cases, need to declare or dimension the array ahead of time, as RATS can usually determine the form of the new variable from the context of the expression. In some cases, however, you may need or want to define the array type using either a DECLARE instruction ahead of time, or a type modifier on the COMPUTE instruction (for example, to declare an array as type SYMMETRIC, rather than letting RATS define it as the more general RECTANGULAR type). Also, a few matrix functions do need a dimensioned target array, as they aren’t functions of other matrices. Examples of this are %RAN and %UNIFORM.

Scalar Calculations

`COMPUTE` can do almost any kind of scalar calculation. Some examples:

```
compute ndraws=500
compute fstat = ( (rssr-rssu)/q ) / (rssu/%ndf)
```

This first sets the (INTEGER) NDRAWS to 500. The second computes an F-statistic using previously set values, and saves the result in the variable FSTAT.

Matrix Calculations

`COMPUTE` is the primary instruction for matrix calculations. Some examples:

```
compute c = a*b
```

sets the array C equal to the product of the arrays A and B. You do not have to declare or dimension C ahead of time.

```
compute [symmetric] cmominv = inv(%cmom)
```

computes the inverse of the cross-moment matrix %CMOM and stores the resulting array in the (new) SYMMETRIC array CMOMINV.
Multiple Expressions

You can evaluate multiple expressions with a single `COMPUTE` instruction. Just separate the expressions with commas. For example:

```r
compute b0=%beta(1) , b1=%beta(2) , b2=%beta(3)
```

You can also combine different types of expressions in a single `COMPUTE`:

```r
compute adotb = %dot(a,b), ainv = inv(a), binv = inv(b)
```

Here, `ADOTB` is a real variable, while `AINV` and `BINV` are arrays.

Literal Matrices

You can use `COMPUTE` to construct arrays directly in an expression. For example:

```r
compute [vector] r = ||1.0, 2.0, 3.0||
```

creates a 3-element `VECTOR` called `R`, with elements 1.0, 2.0, and 3.0, while

```r
compute [vector[label]] names = ||"US","Canada","Japan","Mexico"||
```

creates and sets a 4-element `VECTOR` of `LABELS`.

Strings and Labels

To construct complicated strings, it is probably easiest to use `DISPLAY` with the `STORE` option. You can, however, set `LABEL` and `STRING` variables using either literals or the `+` operator (which concatenates strings).

```r
compute header = "Cross-correlations"
```

```r
declare vector[labels] vlabel(10)
do i=1,10
    compute vlabel(i) = "series"+i
end do
```

The second creates a vector containing the labels “series1”, “series2”, and so on.

Setting Series and Array Elements with COMPUTE

You can use `COMPUTE` to fill individual entries of series and arrays.

```r
compute sales(1990:1) = %na
```

sets entry 1990:1 of series `SALES` to the missing value code.

```r
do row=1,%rows(%cmom)
    compute %cmom(row,row) = %cmom(row,row)+k
end do row
```

adds the constant `K` to each entry on the diagonal of the `%CMOM` array.
declare vector b(n)
compute fill=0 , b = %const(0.0)
do i=1,n
   if a(i)>0
      compute fill=fill+1 , b(fill) = a(i)
end do i

copies the positive entries of the vector A to the vector B. At the end of the loop, FILL
is equal to the number of entries copied.

You should prefer SET (for data series) or EWISE (for matrices) to looping over a
COMPUTE wherever reasonable. Those instructions are faster and shorter. However,
there are circumstances in which COMPUTE offers the cleanest coding.

If you are setting a series, please note that RATS will give you an “unrecognizable
name” error if the series name first appears in the COMPUTE instruction itself. To
avoid this, introduce the series first using a CLEAR instruction:

clear posval
do i=1,435
   if y(i)>0.0
      compute posval(i)=1.0
   else
      compute y(i)=posval(i)=0.0
end do i

Note, by the way, that this example could be done (more efficiently) by

set y 1 435 = %max(y,0.0)
set posval 1 435 = y>0.0

Models, Equations and ParmSets

COMPUTE can also be used to construct new MODEL, EQUATION or PARMSET objects
from existing MODELS or EQUATIONS. For example:

compute bigmodel = model1 + model2
compute foremodel = foremodel + salesident
compute neweqn = equation1 + 3.0*(equation2 + equation3)
compute fullset = base + constraints

The first line constructs a new model called BIGMODEL from existing models MODEL1
and MODEL2. The second line adds the equation SALESIDENT to the existing model
FOREMODEL. The third line constructs NEWEQN as an equation whose right hand side
is a linear combination of three existing equations. The fourth line creates FULLSET
as a combination of the BASE and CONSTRAINTS parmsets.

See Also . . .

UG, Section 4.1  Scalar calculations
UG, Section 4.6  Matrix calculations
COPY — Flexible Data Output

COPY is used to write data series out to a file in any of a variety of formats, and is generally preceded by an OPEN COPY command supplying the name of the file you want to create. COPY can also be used to display data in the output window, but the PRINT command is usually more convenient for this.

```
copy( options )    start   end   list of series
< text cards >    used only with the option HEADER
```

Wizard

You can also copy series to a file using Data—Show Series Window, which displays a list of all the series in memory. From this window, select the series you wish to export, then do File–Export, choose the file format using the “Save As Type” field, enter a file name, and click on “OK” to export the data.

Parameters

- **start end** Range of entries to be printed or saved to disk. If you have not set a SMPL, COPY uses the smallest range required to show all defined data for the series. Any undefined data are treated as missing values.

- **list of series** List of series to be printed or saved. If you omit the list, all current series are printed.

Options

- **organization=columns/rows**
  The option ORGANIZATION (or ORG) tells RATS how the data should be organized on the file. Use COLUMNS if you want the data arranged in columns (that is, series run down the page in columns, with each row containing a single observation for all series). Use ROWS if you want the series to run across the page in rows.

  Note that RATS still supports the choices OBSERVATION (equivalent to COLUMNS) and VARIABLE (equivalent to ROWS) used in older versions of RATS. The ORG option is ignored for RATS and PORTABLE format files, which have a set format.

- **unit=[copy]/output/other unit**
  This option sets the destination of the COPY operation. This can be an external file or the current OUTPUT unit. The default choice is the COPY unit, which you open with the instruction OPEN COPY. If you don’t open the unit in advance, RATS will prompt you for a file name.
Copy

format=[free]/wks/prn/xls/rats/dbf/dif/portable/"(FORTRAN format)"
cdf/tds//binary/html/cdf

The format to be used for the output. See Chapter 2 of the User’s Guide for
details on the various file formats supported. The FREE and FORTRAN formats are
also discussed on the following pages. You can use any of these if you are putting
the output to the COPY or other unit. However, UNIT=OUTPUT will only accept
FREE, PRN, PORTABLE and CDF.

dates/[nodates]
With the DATES option, observations are labeled with dates or entry numbers (if
no CALENDAR is in effect). With NODATES, observations are not labeled at all.

header=number of header lines
Use this option if you want to put a header at the top of the data. Follow the
COPY instruction with the indicated number of lines of text. These lines are
printed above the data or added to the top of the data file. Note: HEADER does not
work with the BINARY or spreadsheet formats.

picture="picture-style formatting string"
This provides a simple means of formatting the numbers. A picture clause takes
a form like "*.###" or "###.##". The first of these means three digits right of
the decimal, and as many digits left as needed. The second means three digits
left and two right (numbers will be rounded to the specified number of decimals).
This applies to formats FREE, PRN, DIF, HTML and CDF. Formats which produce a
spreadsheet or database, such as XLS or WKS, will always write the full double
precision value. See DISPLAY for details on picture clauses.

Notes
For most file formats, the COPY command creates a new file, replacing any existing
file with the same name. For this reason, you cannot use multiple COPY commands to
append data to a single file. If you have a repetitive process that is generating data
that needs to be saved in a single file, you first need to collect all the data in a set of
series, and then write the series to disk using a single COPY command. VECTORS of
SERIES can often be very useful in these situations.

The only exceptions to this are when you are writing text files using the FREE or
FORTRAN formats. For these formats, you can use multiple COPY commands to
append data to a single file, as long as you don’t close the file between COPY com-
mands. After doing OPEN COPY, the first COPY will create the new file (replacing any
previously existing file with that name). Subsequent COPY commands will continue
to write to the file until you quit RATS, issue a CLOSE COPY command, or use OPEN
COPY to supply a different file name. If you do want to append data to a file that has
been closed, use the APPEND option on the OPEN COPY command to reopen the file.
Using FORMAT=FREE

COPY with FORMAT=FREE is not quite analogous to DATA with FORMAT=FREE: if you use the DATES option, COPY will produce a file which DATA cannot process.

Use FORMAT=FREE in one of two situations:

- You are porting data to a program which cannot accept any of the “labeled” formats. You will get a file which consists of numbers only.
- You want to print data to the OUTPUT unit (usually the screen when working in interactive mode).

While the spreadsheet formats will use only one (possibly extremely long) line to represent all entries of a series (with \texttt{ORG=ROWS} or observation (with \texttt{ORG=COLS}), FORMAT=FREE uses multiple lines if necessary to keep the width of the output within 80 columns. For instance, this is a segment of the output for a single quarterly data series (this was done with options \texttt{FORMAT=FREE,ORG=ROWS,DATES}):

\begin{verbatim}
1947:01  224.9000000  229.1000000  233.3000000  243.6000000
1948:01  249.6000000  257.1000000  264.0000000  265.5000000
1949:01  260.1000000  256.6000000  258.6000000  256.5000000
1950:01  267.4000000  276.9000000  294.5000000  305.9000000
1951:01  319.9000000  327.7000000  334.4000000  338.5000000
\end{verbatim}

FORTRAN Formats

FORTRAN formats (\textit{User’s Guide}, Section 2.9) are very similar to free-format with a picture clause. It does, however, give you a bit more control over the appearance.

If you use a FORTRAN format with the option DATES, your format must allow for the date string at the start of each line. This is an \texttt{A\_w} format, where \texttt{w} is at least 7 for quarterly and other “periods per year” data, 10 for daily, weekly, etc. and 15 for intraday or panel data. In addition, if you use \texttt{ORG=ROWS} with DATES, there is a special option \texttt{ACROSS} which you must set correctly:

\texttt{ACROSS=number of entries per line [4]}

\texttt{ACROSS} indicates the number of data values the format will print per line.

For example, we could display the data above formatted eight across by using COPY with the options \texttt{FORMAT="(A8,8F8.1)",ACROSS=8,DATES}.

\begin{verbatim}
1947:01  224.9  229.1  233.3  243.6  249.6  257.1  264.0  265.5
1949:01  260.1  256.6  258.6  267.4  276.9  294.5  305.9
1951:01  319.9  327.7  334.4  341.1  341.3  347.0  359.2
1953:01  365.4  368.8  367.8  362.6  362.0  361.8  366.2  375.0
\end{verbatim}
Examples

open copy fooddata.xls
  copy(dates,format=xls,org=columns)

writes all series in memory onto an Excel file, labeled with dates. Please note that you have to create an entire spreadsheet or database file with a single instruction—RATS cannot “append” data to an existing spreadsheet.

open copy fooddata.lst
  copy(dates,format="(a8,5f8.3)",org=cols)

writes the same series to a text file using FORTRAN format.

open copy geprice.dat
  copy(across=5,org=rows,header=1,dates,format="(a10,5f13.3)") $ 2000:1:3 2003:12:31 ge
  Daily stock prices for General Electric

creates the file GEPRICE.DAT, with daily data listed five across and a header line at the top of the file.

open copy geprice.dat
  copy(header=1,format=portable) 2000:1:3 2003:12:31 ge
  Daily stock prices for General Electric

is similar to the previous example, but uses PORTABLE format. Notice that you do not need to use the DATES option (or ACROSS) with PORTABLE.

cal(q) 1960:1
all 1998:4
open data oecdg7.rat
data(format=rats) / canrgdps frargdps deurgdps gbrrgdps usargdps
  pform g7rgdp
# canrgdps to usargdps
cal(q,panelobs=%nobs) 1960:1
open copy g7gdp.rat
  copy(format=rats,header=1) / g7rgdp
  G7 Real GDP Data

constructs a panel data set and writes it out as a RATS format file.

See Also . . .

OPEN         Opens disk files.
PRINT        Displays data series to the OUTPUT unit.
DISPLAY      Displays scalars and expressions.
WRITE        Displays the contents of arrays.
DATA         Reads data from a file into RATS.
CORRELATE — Autocorrelation and Related Functions

**CORRELATE** computes autocorrelations and related functions for a time series. This is usually used at two points in fitting an ARIMA model:

- In the *identification* stage, it helps you determine the type of differencing you need to apply to a series to produce stationarity. It also helps you select tentative ARMA models for the differenced series.
- In the *estimation* stage, you can apply **CORRELATE** to the residuals for model diagnostics.

```
correlate( option )    series    start    end
```

**Wizard**
You can use *Univariate Statistics* on the *Statistics* menu to compute correlations.

**Parameters**
```
series          The series for which to compute statistics.
start  end   Range of entries to use. If you have not set a *SMPL*, this defaults to the defined range of *series*.
```

There is also a fourth parameter which has the same function as the *RESULTS* option.

**Options**
```
number=number of correlations to compute $[\min\left(\frac{T}{4} , 2\sqrt{T}\right)]$

This sets the number of autocorrelation lags to compute. The default is a function of the number of observations.
```

```
method=yule/[burg]

This selects the method used to compute the correlations. The default is the Burg algorithm. The option *YULE* selects the Yule–Walker method (the method used in versions of RATS before 5.0, and the one provided in most other software). Technical information is provided later in this section.
```

```
covariances/[nocovariances]

By default **CORRELATE** computes autocorrelations. If you use the *COVARIANCES* option, **CORRELATE** computes autocovariances rather than autocorrelations.
```

```
results=series for correlations or covariances [none]
partial=series for partial correlations or covariances [none]
inverse=series for inverse correlations or covariances [none]

Use these options to compute and save the autocorrelations, partial autocorrelations and inverse autocorrelations or, with the *COVARIANCES* option, the autocovariances, partial autocovariances and inverse autocovariances, respec-
```
Correlate

tively. These are all saved in entries 1 through number+1, with the 0 lag value in entry one. The RESULTS option was called CORRELATIONS in previous versions. The old name still works in Version 7.

\texttt{stderrs=series for standard errors [none]}

Use STDERRS to compute and save the (asymptotic) standard errors for the autocorrelations.

\texttt{[center]/nocenter}

Use NOCENTER if you want the correlations to be computed without subtracting means from the data.

\texttt{qstats/[noqstats]}
\texttt{span=width of test intervals [all]}
\texttt{dfc=degrees for freedom correction for test [0]}

With QSTATS, CORRELATE computes Ljung–Box (1978) $Q$ tests for absence of autocorrelation. If you use SPAN, it will do a series of $Q$ tests, beginning with lags 1 to width, and increasing the upper bound by width each time. Without SPAN, it does a single test using all the computed lags.

Use DFC to correct the degrees of freedom of the $Q$ statistic. If you are applying CORRELATE to the residuals from a ARIMA model, the degrees of freedom correction should be the number of ARMA parameters that you estimated.

\texttt{[print]/noprint}
\texttt{picture=picture clause for output}
\texttt{title="Title for output"}

Unless you use NOPRINT, RATS displays the computed statistics. PICTURE allows you to control the formatting of the numbers in the output table (see DISPLAY for more on picture codes). Use the TITLE option if you want to provide your own title for the output. By default, this will be “Autocorrelations (covariances) of Series series”

\texttt{organization=rows/columns}

The ORGANIZATION option determines whether the output is oriented in rows or in columns. The default is ORG=ROWS except when using the WINDOW option, where it is ORG=COLUMNS.

\texttt{window="Title of window"}

If you use WINDOW, the output goes to a (read-only) spreadsheet window with the given title, rather than being inserted into the output window or file as text.

\textbf{Partial Autocorrelations}

The partial autocorrelation for lag $k$ is (in effect) the coefficient on lag $k$ in a $k$ lag autoregression. If all partial autocorrelations greater than $p$ are “small,” it indicates that a model no bigger than AR($p$) is adequate for modelling the series.
Inverse Autocorrelations

If a series $y$ has the ARMA representation

\[ \Phi(L)y_t = \Theta(L)u_t, \]

then the inverse autocorrelations are the autocorrelation of the process (call it $z$)

\[ \Theta(L)z_t = \Phi(L)u_t, \]

The inverse autocorrelations have two primary uses:

- They can assist in the identification stage of a Box-Jenkins model, since they reverse the AR and MA polynomials.
- You can use them to compute a maximum entropy estimate of a spectrum (MESA).

Note that the inverse autocorrelations will usually look nothing like the autocorrelations themselves.

Yule–Walker vs Burg Methods

The Yule–Walker equations relate the autocorrelation coefficients to the partial autocorrelation coefficients by

\[
\begin{bmatrix}
\rho(0) & \rho(1) & \rho(2) & \cdots & \rho(k-1) \\
\rho(1) & \rho(0) & \rho(1) & \cdots & \rho(k-2) \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
\rho(k-2) & \cdots & \rho(1) & \rho(0) & \rho(1) \\
\rho(k-1) & \cdots & \rho(2) & \rho(1) & \rho(0)
\end{bmatrix}
\begin{bmatrix}
\pi(k,1) \\
\pi(k,2) \\
\vdots \\
\pi(k,k-1) \\
\pi(k,k)
\end{bmatrix}
= 
\begin{bmatrix}
\rho(1) \\
\rho(2) \\
\vdots \\
\rho(k-1) \\
\rho(k)
\end{bmatrix}
\]

where $\rho(j)$ is the autocorrelation coefficient at lag $j$. $\pi(k,j)$ is an estimate of the lag $j$ coefficient on a $k$ lag autoregression. The partial autocorrelation at lag $k$ is given by $\pi(k,k)$. METHOD=YULE computes the partial autocorrelations by first computing estimates of the $\rho(j)$, then solving versions of the above system for each $k$. The autocorrelations are estimated by

\[
\hat{\rho}(k) = \sum_{t=1}^{T} \frac{(x_t - \bar{x})(x_{t-k} - \bar{x})}{\sum_{t=1}^{T} (x_t - \bar{x})^2} \quad \text{(if NOCENTER, the $\bar{x}$ terms are dropped)}
\]

METHOD=BURG, on the other hand, computes the partial autocorrelations directly from the data. See Burg (1967). This algorithm is also used in Press, et. al. (1988, pp. 449-450). RATS then backs out corresponding estimates of the autocorrelations.
The Yule–Walker equations become ill-conditioned if the series is non-stationary or close to it, producing imprecise estimates of the partial autocorrelations. Burg’s method is less sensitive to roots near the unit circle, and is, therefore, preferred.

**Standard Errors**

In large samples, the variance of the autocorrelation estimators is approximately

\[
\text{Var}(\hat{\rho}(k)) \approx \frac{1}{T} \left( 1 + 2 \sum_{j<k} \hat{\rho}^2(j) \right)
\]

where \( T \) is the number of observations. This is an increasing function of \( k \). The variance of partial autocorrelations is approximately \( 1/T \), independent of the values.

**Q Tests**

The Ljung–Box \( Q \) statistic for \( M \) lags is

\[
Q = T(T + 2) \sum_{j=M}^{M} \frac{\hat{\rho}^2(_j)}{T - j}
\]

Under appropriate circumstances, for a null hypothesis of no serial correlation \( Q \) is asymptotically distributed as a \( \chi^2 \). RATS uses \( (M - \text{DFC} \text{ option}) \) for the degrees of freedom. If the series are residuals from an ARIMA model, the \( \text{DFC} \) option should be set to the number of estimated ARMA parameters: \textbf{BOXJENK} saves this in the variable \%NARMA. For other types of residuals, there is no known asymptotic distribution, so you shouldn’t rely upon this as a formal test.

**Examples**

It is usually easiest to examine correlations by graphing them. The quickest way to do that is to use the \textbf{BJIDENT} procedure included with RATS (see Section 9.4.1 in the \textit{User’s Guide}). However, it is useful to know how to do these yourself so that you can choose exactly how you want to present the information. We recommend that you always use the options \texttt{MAX=1.0, MIN=-1.0} and \texttt{NUMBER=0} on your \texttt{GRAPH}. The \texttt{NUMBER=0} option causes the “time” axis to be labeled with 0 to \texttt{number}. Below is a simple example. See the code in \texttt{BJIDENT.SRC} and other procedures for more complex examples.

```plaintext
log gdpq
difference gdpq / dgdp
correlate(number=40,results=corrl) gdpq
correlate(number=40,results=corrd) dgdp
graph(style=bargraph,key=upright,number=0,max=1.0,min=-1.0) 2
  # corrl
  # corrd
```
Correlate

Output

The code below computes and prints the autocorrelations, partial autocorrelations, and a range of Q-statistics for the series DGDP. The output is displayed in columns.

```
corrld, partial=partd, qstats, span=4, org=column) dgdp
```

Correlations of Series DGDP
Quarterly Data From 1970:02 To 1991:04

<table>
<thead>
<tr>
<th>Lag</th>
<th>ACF</th>
<th>PACF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.31865</td>
<td>0.31865</td>
</tr>
<tr>
<td>2</td>
<td>0.18556</td>
<td>0.09351</td>
</tr>
<tr>
<td>3</td>
<td>0.05177</td>
<td>-0.03551</td>
</tr>
<tr>
<td>4</td>
<td>0.05434</td>
<td>0.03516</td>
</tr>
<tr>
<td>5</td>
<td>-0.00557</td>
<td>-0.03310</td>
</tr>
<tr>
<td>6</td>
<td>-0.01494</td>
<td>-0.01605</td>
</tr>
<tr>
<td>7</td>
<td>-0.07667</td>
<td>-0.06921</td>
</tr>
<tr>
<td>8</td>
<td>-0.30970</td>
<td>-0.30127</td>
</tr>
<tr>
<td>9</td>
<td>-0.06799</td>
<td>0.14451</td>
</tr>
<tr>
<td>10</td>
<td>0.01366</td>
<td>0.09705</td>
</tr>
<tr>
<td>11</td>
<td>0.04005</td>
<td>-0.00226</td>
</tr>
<tr>
<td>12</td>
<td>-0.28780</td>
<td>-0.37764</td>
</tr>
<tr>
<td>13</td>
<td>-0.11757</td>
<td>0.05332</td>
</tr>
<tr>
<td>14</td>
<td>-0.22051</td>
<td>-0.12528</td>
</tr>
<tr>
<td>15</td>
<td>-0.11932</td>
<td>-0.03146</td>
</tr>
<tr>
<td>16</td>
<td>-0.05633</td>
<td>-0.08443</td>
</tr>
<tr>
<td>17</td>
<td>-0.03647</td>
<td>0.04175</td>
</tr>
<tr>
<td>18</td>
<td>0.03144</td>
<td>0.16170</td>
</tr>
</tbody>
</table>

Ljung-Box Q-Statistics

<table>
<thead>
<tr>
<th>Lags</th>
<th>Statistic</th>
<th>Signif Lvl</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>12.801</td>
<td>0.01229</td>
</tr>
<tr>
<td>8</td>
<td>22.795</td>
<td>0.003638</td>
</tr>
<tr>
<td>12</td>
<td>31.987</td>
<td>0.001390</td>
</tr>
<tr>
<td>16</td>
<td>40.468</td>
<td>0.000665</td>
</tr>
</tbody>
</table>

Variables Defined

- **%NOBS** Number of observations (integer).
- **%QSTAT** The test statistic for the (final) $Q$ test.
- **%QSIGNIF** The significance level of the (final) $Q$ test.

Missing Values

In computing the correlations, RATS treats any missing values as being equal to the mean value of the series. The series itself is not modified in any way.
Other Instructions

There are other instructions you can use, directly or indirectly, to compute autocorrelations. In general, these will produce somewhat different results than \texttt{CORRELATE}, which uses methods specifically designed for autocorrelations. For instance \texttt{CMOM(CORR)} applied to a set of lags of a series will give different results for two reasons: each lag of the series will have a separate mean calculated and removed; and \texttt{CMOM} does all calculations over the range available for all the lags. For instance, if you use four lags, it will skip four initial points.
CPRINT: Printing Complex Series

This is the complex analog to the PRINT instruction. It displays the contents of complex data series.

\[ \text{cprint( options ) start end cseries list} \]

Parameters

- **start**  **end** Range of entries to print. By default, the range required to display all defined entries of the series printed.
- **cseries list** List of complex series to print. If you omit the list, RATS prints all of the complex series stored in memory.

Description

RATS prints each entry as a pair of real numbers, the real and imaginary parts respectively. It prints the series in blocks, with two or three to a block, until the list is exhausted. See the sample output below.

Options

- **interval=sampling interval [1]** You can use the INTERVAL option to reduce the number of frequencies printed. CPRINT prints just one entry out of each sampling interval, beginning with start. Because spectral estimates are usually quite smooth, you will lose very little detail while reducing rather considerably the size of the output.
- **lc=[entries]/periods/frequencies** LC (Left Column) indicates how you want to label the entries. RATS assumes that entry one is frequency zero when it computes the PERIODS and FREQUENCIES.
- **length=base number of ordinates (for LC=PERIODS or FREQUENCIES)** tells RATS the number of ordinates upon which the frequencies are based. RATS needs this information to label entries correctly. The base number is:
  - the Fourier transform length if you haven’t sampled with CSAMPLE
  - the transform length divided by the sampling rate if you have.
- **number=labeling number for first entry (for LC=ENTRIES)** If you use NUMBER, RATS labels the first entry it prints with the indicated number, and then numbers the other entries successively.
picture=picture clause [best fit with 11 digits]

You can use the PICTURE option to format the output, usually to reduce the number of decimal places shown and make the numbers easier to read. The picture clause you supply will be applied to both the real and imaginary parts of the number. For instance, PICTURE="##.####" will print (at most) two digits left of the decimal point and four digits right for each part. With values of those magnitudes, you would ordinarily get 8 digits to the right. See DISPLAY for general information on picture codes.

window="Title of window"

If you use the WINDOW option, the output is displayed in a (read-only) spreadsheet window. The series will be in columns, with labels across the top row. You can export the contents of this window to various file formats using File–Export....

Sample Output

cprint(length=nords,lc=freq) 1 (nords+1)/2 9 10

<table>
<thead>
<tr>
<th>ENTRY</th>
<th>Coherence</th>
<th>Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000000</td>
<td>( 0.35994636, 0.00000000)</td>
<td>( 0.00000000, 0.00000000)</td>
</tr>
<tr>
<td>0.024544</td>
<td>( 0.43946027, 0.00000000)</td>
<td>(-0.04697897, 0.00000000)</td>
</tr>
<tr>
<td>0.049087</td>
<td>( 0.41950483, 0.00000000)</td>
<td>(-0.04881498, 0.00000000)</td>
</tr>
<tr>
<td>0.073631</td>
<td>( 0.64649333, 0.00000000)</td>
<td>(-0.52006011, 0.00000000)</td>
</tr>
<tr>
<td>0.098175</td>
<td>( 0.63584895, 0.00000000)</td>
<td>(-0.60448278, 0.00000000)</td>
</tr>
<tr>
<td>0.122718</td>
<td>( 0.68622671, 0.00000000)</td>
<td>(-0.63831181, 0.00000000)</td>
</tr>
<tr>
<td>0.147262</td>
<td>( 0.67584188, 0.00000000)</td>
<td>(-0.67282855, 0.00000000)</td>
</tr>
</tbody>
</table>

etc.

This prints series 9 and 10 labeling the entries by their frequencies. The LENGTH option is used because the number of frequencies printed is different from the number that should be used in calculating the frequencies.

cprint(len=nords,lc=periods,picture="##.####") 1 (nords+1)/2 9 10

<table>
<thead>
<tr>
<th>ENTRY</th>
<th>Coherence</th>
<th>Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infinite</td>
<td>( 0.3599, 0.0000)</td>
<td>( 0.0000, 0.0000)</td>
</tr>
<tr>
<td>256.0000</td>
<td>( 0.4395, 0.0000)</td>
<td>(-0.0470, 0.0000)</td>
</tr>
<tr>
<td>128.0000</td>
<td>( 0.4195, 0.0000)</td>
<td>(-0.0488, 0.0000)</td>
</tr>
<tr>
<td>85.3333</td>
<td>( 0.6465, 0.0000)</td>
<td>(-0.5201, 0.0000)</td>
</tr>
<tr>
<td>64.0000</td>
<td>( 0.6358, 0.0000)</td>
<td>(-0.6045, 0.0000)</td>
</tr>
<tr>
<td>51.2000</td>
<td>( 0.6862, 0.0000)</td>
<td>(-0.6383, 0.0000)</td>
</tr>
<tr>
<td>42.6667</td>
<td>( 0.6758, 0.0000)</td>
<td>(-0.6728, 0.0000)</td>
</tr>
</tbody>
</table>

Same as the previous, but with entries labeled by period, and with a picture clause limiting the display to four digits to the right of the decimal.
Cross

CROSS — Cross-Correlations

CROSS computes the cross-correlations or cross-covariances for a pair of series.

cross( option ) series1  series2  start  end

Wizard

You can use the Cross Correlations wizard on the Statistics menu to do these computations.

Parameters

series1  series2  CROSS computes the cross correlations of these two series.

start  end  The range of entries to use. If you have not set a SMPL, this
defaults to the maximum range over which both series1 and
series2 are defined.

Two additional parameters used in versions of RATS before 7.0 for specifying the
range of lags have been replaced by the FROM and TO options. The old parameters are
still recognized, but we recommend that you use the options.

Description

CROSS computes the cross-correlations of series1 and series2. Note that the “lag”
refers to the number of periods by which series2 lags series1, that is, the correla-
tion at lag k is the correlation between series1(t) and series2(t-k).

Options

from=lowest lag number
to=highest lag number

These specifies the range of lags for which CROSS will compute the cross correla-
tions. Represent leads as negative lags. FROM defaults to –n and TO defaults to n,
where n is \( \min\left(\frac{T}{4}, 2\sqrt{T}\right) \).

covariances/[nocovariances]

If you use the option COVARIANCES, CROSS computes the cross-covariances
rather than the cross-correlations of the series.

results=series for computed cross correlations/covariances [none]

This lets you save the computed statistics in a data series. RATS saves the results
in consecutive entries, with the lowest numbered lag (or highest lead) put into
entry 1. For instance, with first=-10 and last=10, the lead 10 value goes into
entry 1, lag 0 into entry 11 and lag 10 into entry 21. The RESULTS option was
called CORRS in previous versions. The old name still works in Version 7.
With QSTATS, CROSS computes $Q$ tests for absence of correlation. If you compute both lags and leads, it will include tests for lags only, leads only and lags and leads combined. If you use SPAN, it will do a series of $Q$ tests, beginning with the tests for intervals 1 to width, $-\text{width}$ to $-1$ and $-\text{width}$ to $\text{width}$ (subject to your choices of first and last), and increasing the test span by width with each set. Use DFC to correct the degrees of freedom of the $Q$ statistic.

Use NOCENTER if you want the correlations to be computed without subtracting means from the data.

Unless you use NOPRINT, RATS displays the computed statistics. PICTURE allows you to control the formatting of the numbers in the output table. Use the TITLE option if you want to provide your own title for the output. By default, this will be “Cross-correlations (covariances) of Series series1 and series2”

The ORGANIZATION option determines whether the output is oriented in rows or in columns. The default is ORG=ROWS except when using the WINDOW option, where it is ORG=COLUMNS.

If you use the WINDOW option, the output goes to a (read-only) spreadsheet window with the given title rather than being inserted into the output window or file as text. Each lag is shown on a separate line.

### Missing Values

CROSS treats missing values as being equal to the mean of the valid observations.

### Variables Defined by CROSS

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%NOBS</td>
<td>Number of observations (integer)</td>
</tr>
<tr>
<td>%QSTAT</td>
<td>The test statistic for the (final) $Q$ test.</td>
</tr>
<tr>
<td>%QSIGNIF</td>
<td>The significance level of the (final) $Q$ test.</td>
</tr>
</tbody>
</table>
Examples

```
set dsales = sales-sales{1}
set dlead = lead-lead{1}
cross(qstats,org=column,from=-8,to=8) dlead dsales
```

computes cross-correlations of the first differences of SALES and LEAD. It also performs $Q$ tests on the correlations.

Output

```
Cross Correlations of Series DLEAD and DSALES

<table>
<thead>
<tr>
<th>Lag</th>
<th>Cross ACF</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td>0.04854</td>
</tr>
<tr>
<td>-7</td>
<td>0.14119</td>
</tr>
<tr>
<td>-6</td>
<td>0.04364</td>
</tr>
<tr>
<td>-5</td>
<td>0.10842</td>
</tr>
<tr>
<td>-4</td>
<td>0.10449</td>
</tr>
<tr>
<td>-3</td>
<td>0.72007</td>
</tr>
<tr>
<td>-2</td>
<td>-0.38029</td>
</tr>
<tr>
<td>-1</td>
<td>0.07092</td>
</tr>
<tr>
<td>0</td>
<td>-0.00317</td>
</tr>
<tr>
<td>1</td>
<td>0.09698</td>
</tr>
<tr>
<td>2</td>
<td>-0.05844</td>
</tr>
<tr>
<td>3</td>
<td>0.05464</td>
</tr>
<tr>
<td>4</td>
<td>-0.02955</td>
</tr>
<tr>
<td>5</td>
<td>0.06766</td>
</tr>
<tr>
<td>6</td>
<td>-0.10622</td>
</tr>
<tr>
<td>7</td>
<td>0.00208</td>
</tr>
<tr>
<td>8</td>
<td>0.09510</td>
</tr>
</tbody>
</table>

Ljung-Box Q-Statistics

<table>
<thead>
<tr>
<th>Lag Range</th>
<th>Statistic</th>
<th>Signif Lvl</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 to 8</td>
<td>6.482</td>
<td>0.593389</td>
</tr>
<tr>
<td>-8 to -1</td>
<td>110.167</td>
<td>0.000000</td>
</tr>
<tr>
<td>-8 to 8</td>
<td>116.651</td>
<td>0.000000</td>
</tr>
</tbody>
</table>
```

Technical Information

**CROSS** uses the following to estimate the cross-correlation function of $x$ and $y$:

\[
\rho_{xy}(k) = \frac{\sum (x_i - \bar{x})(y_{i-k} - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}} \quad \text{(if NOCENTER, the } \bar{x} \text{ and } \bar{y} \text{ are dropped)}
\]

Notice that there is no correction for “degrees of freedom” even though the numerator will have $k$ fewer terms than the sums in the denominator.
The correlations computed with **CROSS** will differ from those computed with **CMOM** for two reasons:

- **CMOM** computes the mean for each lag separately, while **CROSS** just uses the overall sample mean for \( x \) and \( y \).
- **CMOM** restricts the range used for all calculations based upon the lags and leads required. **CROSS** adjusts each calculation separately.

The Ljung–Box \( Q \) statistic for lags M1 to M2 is

\[
Q = T(T+2) \sum_{M_{1} \leq j \leq M_{2}} \frac{\hat{\rho}_{xy}^{2}(j)}{T-|j|}
\]

For a null hypothesis of no correlation at any lead or lag, \( Q \) is asymptotically distributed as a \( \chi^2 \) with \( M_{2} - M_{1} + 1 \) degrees of freedom.

Also, under a null hypothesis of no correlation at any lead or lag, the asymptotic variance of each of the correlation estimates is \( 1/T \).

**See Also . . .**

The **CROSSCORR** procedure, on the file **CROSSCORR.SRC**, uses the **CROSS** instruction to compute and graph cross-correlations.

- **CORRELATE**
  - Computes autocorrelations for a single series.
- **CMOM**
  - Computes correlations of a set of regressors.
- %CORR (A, B)
  - Computes the correlation of a pair of series or vectors.
- %COV (A, B)
  - Computes the covariance between a pair of series or vectors.
CSAMPLE: Reducing Frequencies

CSAMPLE reduces the number of frequency ordinates in cseries. It creates newcseries by selecting only one entry in each sampling interval. The CPRINT and CTOR have built-in options which accomplish the same task.

```
csample( option ) cseries start end newcseries newstart
```

Parameters

- **cseries**: Complex series to transform.
- **start** and **end**: Range of entries to process. By default, the defined range of cseries.
- **newcseries**: Series for the result.
- **newstart**: Starting entry in newcseries for the result. By default, same as start.

Options

- **interval=sampling interval [1]**
  The INTERVAL option specifies the width of the sampling interval.

Example

```
freq 5 384
...
csample(interval=3) 1 1 384 2
```

series 2 is set equal to entries 1,4,7,... from series 1.
CSET: General Transformations

CSET is the frequency domain equivalent of SET. It doesn’t get as much work as SET because you will usually use special purpose instructions like CMULTIPLY and CACCUMULATE. CSET sets the values of a complex series using a function.

\[
\text{cset( options ) } \quad \text{cseries } \text{start } \text{end } = \text{function}(T)
\]

Parameters

- **cseries**: The complex series to create or set with this instruction.
- **start end**: The range of entries to set. *This defaults to the FREQUENCY range, regardless of the series involved in the function.*
- **function(T)**: The function of the entry number \( T \), which gives the value for entry \( T \) of cseries. *There should be at least one blank on either side of the = which comes between the other parameters and the function.*

Description

This sets the values of entries \( \text{start} \) to \( \text{end} \) of cseries by evaluating the function at each of the entries, substituting for \( T \) the number of the entry being set.

To get entry \( T \) of complex series \( n \), use \%Z\( (T, n) \). There are several specialized functions which are handy for computing functions of frequency ordinates:

- \%UNIT\( (x) \) returns \( \exp(ix) \) for the real number \( x \)
- \%UNIT2\( (t1, t2) \) returns \( \exp(-i2\pi(t1-1)/t2) \) for \( t1 \) and \( t2 \) integers. \( t1 \) is usually \( T \). With \( t2 \) equal to the number of frequencies, this maps the unit circle as \( T \) runs from 1 to \( t2 \).
- \%ZLAG\( (t, x) \) returns \( \exp(-i2\pi(t-1)x/N) \) where \( N \) is the number of frequencies that you have specified on \text{FREQUENCY}. As \( t \) runs from 1 to \( N \), this gives the appropriate Fourier transform for the lag operator \( L^t \).
Options

**first=expression for first entry set**

You need to use the **FIRST** option if the first entry you are setting requires a different formula than the remaining entries.

**scratch/[noscratch]**

Use the **SCRATCH** option when you are redefining a series (that is, the *cseries* is also part of the function) and the transformation uses data from more than just the current entry $T$. This is rare in frequency domain work.

Examples

\[
cset 4 = \frac{\%z(t,3)}{\%csqrt(\%z(t,1)\times\%z(t,2))}
\]

Sets series $4_i = 3_i/\sqrt{2_i \times 1_i}$

\[
cset 3 = \%z(t,3)\times\%z(t,4)
cset 3 = \%z(t,3)\times\text{conjg}(\%z(t,4))
\]

The first makes series 3 equal to series 3 multiplied by the conjugate of series 4, while the second makes 3 equal to 3 multiplied by 4 (without conjugation).

\[
cset 3 = \%unit2(t,400)
cset 3 = \%cexp(\%cmplx(0.0,2\times\pi\times(t-1)/400.))
cset 3 = \%cmplx(\cos(2\times\pi\times(t-1)/400.),\sin(2\times\pi\times(t-1)/400.))
\]

Any of these makes entry $t$ of series 3 $= \exp\left(\frac{2\pi i (t-1)}{400}\right)$

\[
cset 5 = c=\%cmplx(1.0,0.0),\%do(i,1,4,c=c+b(i)\times\%zlag(t,i)),c
\]

Makes series 5 equal to the transfer function for $1 + b(1)L + b(2)L^2 + b(3)L^3 + b(4)L^4$. Writing it this way allows the number of lags in the polynomial to be changed quickly by changing the size of the vector $b$.

\[
cset(\text{scratch}) 3 = \%z(t,3)\times\%z(1,3)
\]

Replaces series 3 by itself normalized so that entry 1 is 1.0. The **SCRATCH** option is needed because entry 1 will be overwritten before it gets used to compute entries from 2 on.
CTOR: Data From Complex to Real

CTOR copies data from complex series into real-valued series. You must do this if, for example, you want to graph any series created in the frequency domain. It is also necessary if you have real-valued series which you have created in the frequency domain by an operation such as frequency domain filtering.

```
ctor( option ) start   end   newstart
# list of complex series
# list of real series
```

Parameters

- `start`  `end` Range of entries to transfer. This defaults to the defined range of each complex series.
- `newstart` Entry in the real series to which the `start` entry is copied. By default, same as `start`. The default starting entry may change from series to series if you use the default `start` to `end` range.

Description

This transfers data from the complex series listed on the first supplementary card to the corresponding real series listed on the second supplementary card.

Options

- `part=[real]/imaginary/absvalue` PART specifies which part of the complex numbers you want to send to the time domain. (ABSVALUE = Absolute Value)

- `interval=sampling interval [1]` You can use the INTERVAL option to reduce the number of frequencies transferred. CTOR copies just one entry out of each sampling interval, beginning with `start`. Because spectral estimates are usually quite smooth, you will lose very little detail while reducing rather considerably the time required to draw a graph.
Examples

```plaintext
ctor(interval=4) 1 720
#  1
#  spectrum
```

copies the real part of every fourth entry of complex series 1 to entries 1 through 180 of the real series SPECTRUM.

This next example sends two series to the frequency domain, filters them by using a seasonal mask on the Fourier transforms, then sends the results back. Note that, while this produces “values” for series 1 and 2 which go beyond the actual data length, it is only the original range which is actually useful.

```plaintext
compute nords=2*%freqsize(2007:6)
freq 3 nords
rtoc 1960:1 2007:6
# x y
# 1 2
fft 1
fft 2
cmask 3 1 nords
cmult 1 3
cmult 2 3
ift 1
ift 2
ctor 1960:1 2007:6
# 1 2
# xadj yadj
```

See Also...

**RTOC** The inverse operation of **CTOR**—this copies real data into complex series.
CVMODEL — Covariance Matrix Modelling

CVMODEL estimates a parametric model for a covariance matrix. Its main use is in structural VAR modelling. See the User’s Guide, Section 10.5, for most of the technical details.

The instruction estimates the parameters in the A and B matrices in

(1) \( \text{Au}_i = \text{Bv}_i ; \ E(\text{v}_i \text{v}'_i) = \text{D} ; \ \text{D diagonal} \)

In most applications, only one of A and B is actually used. A and B are specified using FRML[RECT]'s, that is, formulas which evaluate to a rectangular matrix.

```
cvmodel( options ) sigma  A Formula  B Formula
```

Parameters

We recommend that you use the A and B options rather than the A Formula and B Formula parameters.

\( \text{sigma} \) The SYMMETRIC covariance matrix of the residuals u.

\( \text{A Formula} \) A FRML[RECT] which gives the A matrix in (1).

\( \text{B Formula} \) A FRML[RECT] which gives the B matrix in (1).

Options

\( \text{a=} \) A formula producing RECTANGULAR array

\( \text{b=} \) B formula producing RECTANGULAR array

Use these to provide formulas for the A and B matrices in (1).

\( \text{v=} \) formula producing SYMMETRIC array to model covariance matrix

Use this to supply a formula that models the covariance matrix itself. Omit the A and B formulas when using this option.

\( \text{factor=} \) resulting factoring matrix for sigma

If the model is just identified, this should give a matrix F such that \( FF' = \Sigma \). If the model is overidentified, it won’t be an exact factorization.

\( \text{parmset=} \) PARMSET to use [default internal]

This option selects the parameter set to be estimated (see the User’s Guide, Section 7.6 for more information). RATS maintains a single unnamed parameter set which is the one used for estimation if you don’t provide a named set.
dmatrix=[concentrated]/identity/marginalized

dfc=degrees of freedom correction [0]

pdf=prior degrees of freedom [0]

These select the specific form for the maximand. The DMATRIX option determines the treatment of the “D” matrix, whether it’s concentrated out, set to the identity or integrated out. The DFC option is used when you’re maximizing the posterior density. It gives the value of $c$ in the formulas in Section 10.5 of the User’s Guide. The PDF option sets $\delta$, which is used for the posterior density with $D$ integrated out.

method=[bfgs]/simplex/genetic

pmethod=bfgs/[simplex]/genetic

piters=number of PMETHOD iterations to perform [none]

BFGS is Broyden, Fletcher, Goldfarb and Shanno; SIMPLEX is the simplex algorithm, GENETIC is a genetic algorithm. See Chapter 7 of the User’s Guide for a fuller description of these methods. Note that BFGS is the only method which computes standard errors. However, it is very sensitive to initial guess values.

Use PMETHOD and PITERS if you want to use a preliminary estimation method to refine your initial parameter values before switching to one of the other estimation methods. For example, to do 20 simplex iterations before switching to BFGS, you use the options PMETHOD=SIMPLEX, PITERS=20, and METHOD=BFGS.

hessian=initial guess for inverse Hessian (METHOD=BFGS only)

You can use this with METHOD=BFGS. Without it, CVMODEL will start with the identity matrix. See Section 7.2 in the User’s Guide.

observations=Number of observations used to compute sigma

This usually isn’t needed to estimate the parameters, but is needed in order to compute standard errors and do a test for overidentifying restrictions. The default value is the number of observations in the last regression you have run.

[print]/noprint

vcv/[novcv]

title="title for output" ["Covariance Model"]

These are the same as for other estimation instructions (see LINREG).

reject=expression giving a “rejection” zone for the parameters

If the expression evaluates to a non-zero (“true”) value, the function is immediately assigned the missing value.
**Examples**

This estimates a model for a six-variable covariance matrix. With nine free parameters, this is overidentified by six. (There are $21 = 6 \times (6 + 1)/2$ distinct values in a $6 \times 6$ covariance matrix, with six parameters used for the variances of the $v$'s, leaving fifteen to be estimated as part of the model). It is estimated using the BFGS method.

The factor matrix is put into `AFACtor`. Note that `AFRML` must be declared as a `FRML[RECT]` before it can be defined using the `FRML` instruction.

```r
nonlin rx ur cu cr pc pr mp mc mr
dec frml[rect] afrml
frml afrml = |1.0,0.0,ur,0.0,0.0,0.0|$
cu,1.0,cr,0.0,0.0,0.0|$
0.0,0.0,1.0,rx,0.0,0.0|$
0.0,0.0,0.0,1.0,0.0,0.0|$
0.0,mc,rmr,0.0,1.0,mp|$
0.0,pc,pr,0.0,0.0,1.0|$
compute ur=cu=cr=rx=mc=mr=mp=pc=pr=0.0
cvmodel(method=bfgs,factor=afactor,a=afrml) %sigma
```

If you use `METHOD=BFGS`, the output will look something like that shown below. With `METHOD=SIMPLEX` or `METHOD=GENETIC`, the only columns that will show in the bottom table will be Variable and Coeff, since neither of those methods can compute standard errors.

If the model is overidentified, it will show both the Log Likelihood of the estimated model, and the unrestricted Log Likelihood, along with the test of the restrictions. For a just identified model, it will show both the likelihoods, but no test. The two likelihoods should match—if they don’t, you hit a local but not global maximum.

The output is shown on the following page.
Covariance Model - Estimation by BFGS
Convergence in 19 Iterations. Final criterion was 0.0000011 < 0.0000100
Observations 102
Log Likelihood 2567.62144857
Log Likelihood Unrestricted 2573.70076146
Chi-Squared(6) 12.15862579
Significance Level 0.05852218

<table>
<thead>
<tr>
<th>Variable</th>
<th>Coeff</th>
<th>Std Error</th>
<th>T-Stat</th>
<th>Signif</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. RX</td>
<td>25.06964162</td>
<td>6.38993684</td>
<td>3.92330</td>
<td>0.00008734</td>
</tr>
<tr>
<td>2. UR</td>
<td>-0.00248033</td>
<td>0.00054575</td>
<td>-4.54484</td>
<td>0.00000550</td>
</tr>
<tr>
<td>3. CU</td>
<td>-0.26283113</td>
<td>0.09190076</td>
<td>-2.85995</td>
<td>0.00423714</td>
</tr>
<tr>
<td>4. CR</td>
<td>-0.00057282</td>
<td>0.00054212</td>
<td>-1.05662</td>
<td>0.29068605</td>
</tr>
<tr>
<td>5. PC</td>
<td>0.12961118</td>
<td>0.06825089</td>
<td>1.89904</td>
<td>0.05755919</td>
</tr>
<tr>
<td>6. PR</td>
<td>-0.00018272</td>
<td>0.00035541</td>
<td>-0.51412</td>
<td>0.60717028</td>
</tr>
<tr>
<td>7. MP</td>
<td>-1.22317380</td>
<td>0.42668633</td>
<td>-2.86668</td>
<td>0.00414800</td>
</tr>
<tr>
<td>8. MC</td>
<td>-0.06782941</td>
<td>0.29526292</td>
<td>-0.22973</td>
<td>0.81830512</td>
</tr>
<tr>
<td>9. MR</td>
<td>0.00119296</td>
<td>0.00153545</td>
<td>0.77694</td>
<td>0.43719267</td>
</tr>
</tbody>
</table>

The code below uses the v option to estimate a one factor model for the covariance matrix RR.

dec vect lambda(n) d(n)
compute lambda=%const(0.1)
compute d=%xdiag(rr)
dec frml[symm] vf
frml vf = %outerxx(lambda)+%diag(d)
cvmodel(method=bfgs,v=v,obs=tdim) rr

Notes
These models can be quite difficult to estimate successfully (see Section 10.5.2 of the User’s Guide). There are often multiple local maxima, and there can even be multiple global maxima in very different regions of the parameter space. Before you put a lot of effort into writing up results, experiment with different initial guess values and METHOD=GENETIC to make sure you didn’t end up at the wrong spot on the first try.

Variables Defined
Standard estimation: %NREG, %LOGL, %XX, %BETA, %STDERRS, %TSTATS
Non-linear estimation: %CVCRIT, %ITERS, %CONVERGED, %LAGRANGE

%CDSTAT  The test statistic for overidentifying restrictions
%SIGNIF   The significance level of that test
%FUNCVAL  The log likelihood at the final estimates
%NVAR     The number of variables (integer)
CXTREMUM: Extreme Values for Complex Series

CXTREMUM locates the smallest and largest values of a complex series \( cseries \) using the mapping to the reals specified by the PART option. It is the complex analog of EXTREMUM.

\[
\text{cxtremum}( \text{options} ) \quad cseries \quad \text{start} \quad \text{end}
\]

Parameters

- \( cseries \): Complex series whose extreme values you want to locate.
- \( \text{start} \quad \text{end} \): Range of entries to search.

Options

- \( \text{part}=[\text{real}]/\text{imaginary}/\text{absvalue} \)
  
  The part of the complex numbers (real, imaginary, or absolute value) whose extreme values you want.

- \( \text{[print]}/\text{noprint} \)
  
  NOPRINT suppresses the output. Use NOPRINT if you want to use the variables described below, but you don’t need the printed output.

Variables Defined by CXTREMUM

- \( \%\text{MAXIMUM} \): maximum value (real)
- \( \%\text{MINIMUM} \): minimum value (real)
- \( \%\text{MAXENT} \): entry at which maximum was found (integer)
- \( \%\text{MINENT} \): entry at which minimum was found (integer)

Output

\[
\text{cxt(part=absval)} \quad 3
\]

Extreme Values of Absolute Value of Series Y
Minimum Value is 0.0000000000000 at Entry 128
Maximum Value is 0.3094760486005 at Entry 56
DATA — Reading Data Series

**DATA** reads data series from an external file into working memory, and is usually preceded by an **OPEN DATA** command, although this is not required. We’ve devoted Chapter 2 of the *User’s Guide* to the use of the **DATA** instruction. Thus, the discussion here is limited. The corresponding instruction for writing data is **COPY**.

```
data( options )   start   end   list of series
```

Wizard

The *Data* wizards (on the *Data* menu) generate the appropriate sequence of **CALENDAR**, **ALLOCATE**, **OPEN DATA**, and **DATA** instructions to read data from a file.

Parameters

- **start**  **end**  Range of entries to read. If you have not set a **SMPL**, this defaults to the **ALLOCATE** range.
- **list of series**  The list of series names or numbers you want **RATS** to read from the file. If you omit the list, **RATS** reads all of the series on the file. You can take advantage of this with any formats except **FREE**, **BINARY** or **FORTRAN** format.

Options

- **format**=[**free**]/**wks**/**prn**/**xls**/**rats**/**dbf**/**dif**/**portable**/**'(FORTRAN format)'**
  **cdf**/**tsd**/**binary**/**citibase** or **dri**/**haver**/**fame**/**odbc**/**crsp**
  This describes the format of your data set. Note that **RATS** will not attempt to determine the format from the file name or extension. **You must use the correct FORMAT option.** The various formats are described in Chapter 2.
- **organization**=**columns**/**[rows]**  (or **org**=**[variables]**/**observation**)
  This tells **RATS** whether the data series run down the file in columns (one series per column) or across it in rows. Note that the older terminology of **ORG=VARIABLE/OBSERVATION** is still supported. **OBSERVATION** corresponds to **COLUMNS** and **VARIABLE** corresponds to **ROWS**. The **ORG** option is ignored for **RATS** and **PORTABLE** format files, which have a set format.
- **unit**=[**data**]/**input**/**other unit**
  With the default option **DATA**, **RATS** reads the data from the external data file. Use **UNIT=INPUT** when you want to enter data directly into the input file.
- **sheet**="**worksheet name**"
  When reading an Excel workbook containing multiple worksheets, you can use the **SHEET** option to provide the name of the worksheet from which you want to read data. By default, **RATS** will read data from the first sheet in the file.
**skiplines**=number of top lines to skip

You can use this if you have comments or blank lines at the top of a text or spreadsheet file which can’t or shouldn’t be processed as part of the actual data. Don’t count in this a header line which provides column labels.

**missing**=missing value code

Use MISSING if you use a numeric code in your data to represent missing observations. For instance, MISSING=-999 will translate observations with value -999 to the RATS missing value. See *User’s Guide*, Section 2.4, for more information.

**blank**=[zero]/missing

This applies only with FORTRAN formats. If an area where DATA expected a number is blank, DATA treats it either as a zero or as a missing value, depending upon your choice for this option.

**verbose**/[noverbose]

Use VERBOSE if you want RATS to list the name and comments (if any) of each series it reads, along with other information about the file. You can use VERBOSE to verify that RATS is reading date information on your data file properly—this is particularly important when converting from one frequency to another.

**compact**=[average]/sum/geometric/first/last/maximum/minimum

**select**=subperiod

These options, which work with most dated file formats, set the method RATS will use to convert data from higher frequency to lower, such as monthly to quarterly. See Section 2.3 of the *User’s Guide*.

**sql**="SQL string"

Used with FORMAT=ODBC, this allows you to connect to a database and read data using SQL commands. See Chapter 2 of the *User’s Guide* for more details. Note that SQL support is only available in the Professional version of RATS.
Examples

open data states.xls
data(org=col,format=xls) 1 50

reads all series from the Excel file STATES.XLS. The file is organized by column.

cal(m) 1955:1
open data brdfix.dat
data(org=row) 1955:1 1979:12 ip

reads data for the series IP from the file BRDFIX.DAT. This is a free-format file, organized by row. Note, that unlike the “labeled” formats, like XLS and RATS, we are free to give the series any name we wish.

cal(a) 1920:1
all 1945:1
open data klein.dat
data(org=col,format="(7x,9f8.3)") 1920:1 1941:1 consumption $
   profit privwage invest klagged production govtwage govtexp taxes

reads data for nine series in Fortran format over the range 1920:1 to 1941:1.

open data haversample.rat
calendar(a) 1982
all 2006:1
data(format=rats,compact=max) / spdji
set djmax = spdji
data(format=rats,compact=min) / spdji
set djmin = spdji
data(format=rats,compact=last) / spdji
set djlast = spdji

reads data which are monthly on the data file into an annual CALENDAR, taking the maximum in each year (to create DJMAX), the minimum (to create DJMIN) and the final (to create DJLAST).
DBox — Creating User-Defined Dialog Boxes

**DBox** is used to define and execute a dialog box requesting input from the user. A **DBox** can include most of the standard dialog box items, such as check boxes, radio buttons, edit text boxes and combo boxes. In addition, you can control whether items are active and inactive, and can verify input before the user is allowed to dismiss the box.

There are several special purpose instructions which allow more limited forms of dialogs: **QUERY** gets a single piece of information from a “text box”, **SELECT** presents a scrolling list for selection of one or more items, and **MENU** and **CHOICE** combine to offer a set of operations from which the user can select one.

A dialog box is created and executed by a series of **DBox** commands, beginning with **DBox(ACTION=DEFINE)** and ending with **DBox(ACTION=RUN)**. Between these, you do one or more **DBox(ACTION=MODIFY)** commands with the other options to design your dialog box:

```
DBox(action=define)
DBox( options ) output value
DBox(action=run)
```

**Parameters**

- **output value**

   For most dialog box elements, the **output value** should be an integer variable, which will contain the current value of that item.

   The **output value** variables can be used as the arguments on **VERIFY** options, to check that the user has supplied valid input, and on **DISABLE** options, to enable or disable elements of the dialog based on values currently selected or input by the user.

   When the user clicks “OK” to close the dialog box, **output value** will contain the final value of the dialog box element.

   A variable must be declared before it can be used as an **output value** on the **DBox** instruction.

   Note that, because these get set during the course of running the dialog, their values can change even if the user eventually cancels the dialog. Use the **STATUS** option to determine if the user OK’ed the dialog.
Item Types

There are several different types of items you can include within a **DBox**. Some of these are shown below (see “Item Type Options” later in this section for the full list). To add an item to a dialog, use the corresponding option name together with **ACTION=MODIFY** (the default choice for **ACTION**). The samples below generally include only one type of item (although some include an additional Static Text item with descriptive text), but note that a single dialog box can include many items. Below each example, we show the code used to generate the dialog box.

**Static text**

A simple text label which can’t be edited.

```plaintext
dbox(action=define,title="Sample Dialog")
dbox(stext="Static text box")
dbox(action=run)
```

**Edit text**

A text box in which the user types information. You can request that this be interpreted as a number or passed back intact as a string. You do that by setting the type of the **output value**.

```plaintext
declare string result
dbox(action=define,title="Sample Dialog")
dbox(stext="Edit text box")
dbox(action=run)
```

```plaintext
declare string result
dbox(action=define,title="Sample Dialog")
dbox(stext="Edit text box")
dbox(edittext, focus) result
dbox(action=run)
```
Check box  
A descriptive string attached to a box which shows either on (checked) or off.

![Check box example](image)

```plaintext
declare integer on
dbox(action=define,title="Sample Dialog")
dbox(checkbox="A check box") on
dbox(action=run)
```

Radio button  
One of a collection of mutually exclusive choices. Such a collection has to be grouped as consecutive items.

![Radio button example](image)

```plaintext
declare integer button
dbox(action=define,title="Sample Dialog")
dbox(radio="Button 1",default=1) button
dbox(radio="Button 2")
dbox(radio="Button 2")
dbox(action=run)
```
Popup box

A list of choices which has much the same function as the radio buttons, but with only the selected choice showing when the box is at rest. If you click on it, you have access to the other choices.

```
declare integer seriesnum
dbox(action=define,title="Sample Dialog")
dbox(stext="Static text box")
dbox(popupbox,series) seriesnum

dbox(action=run)
```

Combo box

A combination of a text box with a popup box. The user can either type information directly into the text box, or activate the popup and choose one of the choices available there.

```
declare integer seriesnum
dbox(action=define,title="Sample Dialog")
dbox(stext="A Combo box")
dbox(combobox,series) seriesnum

dbox(action=run)
```
Scroll box

A list of choices in a scrolling box. If `output value` is an `INTEGER`, only a single selection is allowed, so it would function much like a Popup box, but with more of the choices visible at any one time. If `output value` is a `VECT[INTEGER]`, the user can select any number of the choices from zero up to all.

```
declare rectangular[integer] selected
dbox(action=define,title="Sample Dialog")
dbox(stext="A Scroll box")
dbox(scrollbox,series) selected
dbox(action=run)
```

General Options

- `action=define/[modify]/run`
  As mentioned above, you start with `ACTION=DEFINE` and end with `ACTION=RUN`. Since `ACTION=MODIFY` is the default, you don’t need an `ACTION` option when defining the items in the dialog box.

- `title="dialog title"` (only with `ACTION=DEFINE`) ["User Dialog"]
  This provides the title at the top of the dialog.

- `status=integer variable set to 0,1 status` (only with `ACTION=RUN`)
  If the user cancels, the status variable is set to zero, and if OK, it’s set to one. You should always include a `STATUS` option.
**default**=

DEFAULT specifies the initial value for a checkbox, radio button, or popup box. The defaults are “off” (zero) for checkboxes, and first (1) for radio buttons or popup boxes.

**width**=

WIDTH specifies the width (in number of character positions) of the item. This defaults to 30 characters for edit text or combo box items, and the width of the string (or set of strings) for other items. HEIGHT allows you to set the number of items to show in a SCROLLBOX. By default, this will be 20 or the total number of items available, whichever is smaller.

**special**=

SPECIAL=SAMELINE puts the item at the same vertical position as the previous one, just over to the right. This is commonly used for a static text item (for a prompt) followed by an edit box. SPECIAL=GAP inserts an extra spacing gap before the item.

**focus**/

Use FOCUS to indicate that the item is to be the one which receives the initial focus.

**initialize**/

For an edit text item, this indicates whether the value already in the output value should be used to initialize the text field. If not, it starts out blank.

**optional**/

This applies to the EDITTEXT and COMBOBOX items. If OPTIONAL, the field can be left blank. If it is blank, the output value will be %NA for a real, 0 for an integer, and an empty string for a label or string variable.

**verify**=

VERIFY allows you to verify the input data. You supply an expression which should evaluate to a non-zero value if the input data are acceptable. DISABLE disables items in the dialog box—supply an expression which evaluates to a non-zero value if you want an item to be disabled.

**series**

**strings**=

VECTOR of STRINGS to list

**list**=

VECTOR of INTEGERS to list

**regressors**

When creating a popup, combo, or scroll box, use one of these four options to set the list to be displayed:
• Use SERIES to request a selection from the list of series currently in memory. The output values will be the selected series numbers. Note that the special series CONSTANT is never included in the list.

• Use STRINGS to request a selection from a list of labels.

• Use the LIST option to request a selection from an arbitrary list of integers.

• Use the REGRESSORS option to request a selection of regressors from the last completed regression. The return values are the selected positions in the list.

**Item Type Options**

One and only one of the “type” options should be used on any given DBOX command with the default ACTION=MODIFY. Some of the types have related options—these are described below with their respective type options.

**stext=** text for static text item (label)

**checkbox=** text of the checkbox item

The output value parameter should be an INTEGER variable. Its value will be zero (not checked) or one (checked).

**radio=** text of a radio button item

The set of items which will be treated as a group of radio buttons must be consecutive. The first (and only the first) in the group has the output value parameter. For input (DEFAULT option) and output value purposes, the radio buttons are numbered from 1 to the number in the group.

**edittext**

*output value* can be a scalar variable of any type. If you use the INITIALIZE option, the text field will be initialized with the value in that variable at the time the DBOX is executed. You can use the WIDTH option to restrict the width of the text box. For instance, if you’re expecting an integer value no larger than 99, you can include WIDTH=2.

**spinbutton**

**lowerlimit=** lower limit on values

**upperlimit=** upper limit on values

**date/[nodate]**

SPINBUTTON must follow immediately after an EDITTEXT item. It does not have its own output value. Instead, it adds a spin button (or up-down control) to the preceding EDITTEXT, which allows the user to make minor adjustments to the text box without retyping. You can provide LOWERLIMIT and/or UPPERLIMIT to prevent the user from moving the value outside of that range. If you use DATE, the information will be displayed in date format, though the output value will still be an integer.
popupbox
scrollbox
combobox

The set of strings for any of these comes from one of the SERIES, STRINGS, LIST and REGRESSORS options described above.

For a COMBOBOX, output value can be a scalar variable of any type, which is filled with the value obtained by processing the text field.

For POPUPBOX and SCROLLBOX, the values returned depend upon which option was used to fill the box:

- SERIES: the series number
- STRINGS: the index in the array (1,...,n)
- REGRESSORS: the index into the list of regressors (1,...,number of regressors)
- LIST: the values of the integers

For POPUPBOX, the DEFAULT option can be used to set which of the choices will be selected initially. By default, it will be the first. For POPUPBOX, output value is an INTEGER.

For SCROLLBOX, output value can be either an INTEGER or a VECTOR of INTEGERS. If the latter, the user can select any number of items in the list, from 0 to all, and the VECT[INT] will be filled with the values of the selections. (If none are selected, it will have 0 rows). You can use the HEIGHT option to control the number of items displayed at one time in the scroll box. If the scroll box is the main element of the dialog box, the default of 20 is probably fine, but if it’s less important, you might want to make the box smaller by setting the limit lower than that.

matrix
picture=picture clause for data [none]
vlabels=VECTOR of STRINGS for row labels
hlabels=VECTOR of STRINGS for column labels

If you use MATRIX, the output value is a real-valued matrix which must already be dimensioned. A MATRIX item can be used for one of two things: it can be used to request input of numerical information into a matrix (which is the default behavior), or it can be used to request that the user select rows or columns based upon the displayed values (described below with the MSELECT option).

A picture clause is most helpful when you’re using this to display data for user selection, rather than having the user enter or edit it. By default, DBOX will use a representation with seven digits right of the decimal. This not only may show
many more digits than are statistically reliable, but also will take up more room per number than a smaller format, so fewer values will be visible at one time. For example, the option PICTURE="*.###" could be used to limit the display to three digits right of the decimal. See DISPLAY for more on picture codes.

You can use VLABELS and HLABELS to supply your own labels for the rows and columns in the matrix box. By default, columns and rows are labeled with integer row or column numbers.

\textbf{mselect=rows/one/byrow/bycol}

\textbf{select=VECTOR[INTEGER] of selections}

If you use the \texttt{MSELECT} option (with \texttt{MATRIX}), \texttt{DBox} will display the data, and the user can select items from the matrix. When the window is closed, the \texttt{VECTOR[INTEGER]} that you provide with the option \texttt{SELECT} will be filled with information regarding the selection, as described below.

You use \texttt{STYPE} to control how selections are made. With \texttt{MSELECT=ROWS}, the user can select one or more rows. The array provided using the \texttt{SELECT} option will have dimension equal to the number of selections made, and will list the rows selected using numbers 1,...,\textit{n}. If \texttt{MSELECT=ONE}, it will have dimension 2 with \texttt{array(1)=row} and \texttt{array(2)=column}. If \texttt{MSELECT=BYROW}, the user selects one cell per row. The \texttt{SELECT} array will have dimension equal to the number of rows, with \texttt{array(i) set equal to the column selected in row i}. \texttt{MSELECT=BYCOL} is similar, but one cell per column is selected.
DDV — Discrete Dependent Variables

The DDV instruction performs several forms of discrete dependent variable estimation, including binary choice probit and logit, ordered choice models, and multinomial and conditional logit.

\texttt{ddv( options ) depvar start end}
# list of explanatory variables in regression format (omit if using \texttt{EQUATION} option)

Wizard

The Limited/Discrete Dependent Variables wizard provides dialog-driven access to most of the features of the DDV instruction.

Parameters

\texttt{depvar}  
Dependent variable. RATS requires numeric coding for this. How to code the values will depend upon the model type. See the description under the \texttt{TYPE} option.

\texttt{start end}  
Estimation range. If you have not set a \texttt{SMPL}, this defaults to the maximum common range of \texttt{all} the variables involved.

Options

\texttt{type=\{binary\}/\texttt{ordered/multinomial/conditional/count}}

Model type. BINARY is a binary choice model. ORDERED is an ordered choice model, MULTINOMIAL and CONDITIONAL are for multiple choice logits, and COUNT is for a Poisson count model. The main difference between MULTINOMIAL and CONDITIONAL is that the former is designed for fixed individual characteristics, with different coefficients for the choices, and the latter is to analyze differing attributes across choices, with a fixed set of coefficients. See Section 14.2 for more information.

For BINARY, one choice is represented (in \texttt{depvar}) by zero, and the other by any non-zero value or values. For ORDERED, each choice must have a distinct value, ordered systematically, but they don’t have to have any specific set of values. For MULTINOMIAL each choice again must have a distinct value. COUNT data are just non-negative integer values.

CONDITIONAL requires a data set with a completely different design. Although there are multiple choices, the dependent variable is treated as a binary choice (zero vs non-zero), as each available option for each individual has a separate observation in the data set. The dependent variable is used to indicate whether the particular choice represented by an observation was the one chosen.
distribution=[probit]/logit/extremevalue
The distribution of the underlying “index” model. Probit is the standard normal. This matters only for BINARY and ORDERED; MULTINOMIAL and CONDITIONAL are always done as logit, and COUNT uses a Poisson model.

indiv=SERIES identifying individuals (for CONDITIONAL only)
modes=SERIES identifying modes (choices) (for CONDITIONAL only)
For TYPE=CONDITIONAL, the INDIV option is required in order to indicate which observations are for each individual. The data set is structured differently, as each individual and each choice available for the individual needs a separate observation in the data set. The MODES series is optional; it provides an identifying series for the choices. This might, for instance, have the coding 1=Car, 2=Train, 3=Air. This isn’t necessary to estimate the model, but does allow for more diagnostics.

cutpoints=VECTOR of estimated cut points (TYPE=ORDERED only)
If you have \(m\) choices, this is an \((m-1)\) vector, with the first element showing the cut point between the choice with the smallest value and the second smallest, the second separated the second and third choices, etc.

coeffs=RECTANGULAR of coefficients (TYPE=MULTINOMIAL only)
If you have \(m\) choices, this is an \(K \times (m-1)\) matrix of coefficients where \(K\) is the number of regressors. The first choice is normalized to the zero vector and is left out.

gresids=SERIES of generalized residuals
If the log likelihood element for an observation can be written \(g(X,\beta)\), the generalized residuals are the series of derivatives \(g'(X,\beta)\). These are useful for diagnostic tests.

[print]/noprint
vcv/[novcv]
smpl=SMPL series or formula (User’s Guide, Section 5.2)
unravel/[nounravel] (Section 5.11)
title="title for output" [depends upon options]
equation=equation to estimate
These are similar to the LINREG options.

iterations=iteration limit [100]
subiterations=subiteration limit [30]
cvcrit=convergence limit [.00001]
trace/[notrace]
initial=VECTOR of initial guesses [vector of zeros]
DDV estimates using non-linear methods. Because the models have well-behaved log likelihood functions, you don’t have (or need) as many choices to control this as with, for instance, MAXIMIZE.
All models except the conditional logit are estimated using Newton-Raphson, that is, they use analytical second derivatives. Conditional logit uses BHHH. See the User’s Guide Chapter 7 for details. ITERATIONS sets the maximum number of iterations, SUBITERS sets the maximum number of subiterations, CVCRIT the convergence criterion. TRACE prints the intermediate results. INITIAL supplies initial estimates for the coefficients. The default values are usually sufficient.

robusterrors/[norobusterrors]  
lags=correlated lags [0]  
lwindow=neweywest/bartlett/damped/parzen/quadratic/[flat]/panel/white  
damp=value of $\gamma$ for lwindow=damped [0.0]  
lwform=VECTOR with the window form [not used]  
cluster=SERIES with category values for clustered calculation

These permit calculation of a consistent covariance matrix. Note, however, that, with the exception of TYPE=COUNT, these models are unlikely to produce consistent estimates of the coefficients themselves if the distributional assumptions are incorrect.

entries=number of supplementary card entries to process [all]  
This allows you to control how many of the elements on the supplementary card are processed. See Section 5.15 in the User’s Guide for details.

Description

The three distributions which you can select for the binary and ordered models are the standard normal (probit), logistic (logit) and extreme value. The distribution function for the logistic is

$$F(z) = \exp(z)/(1 + \exp(z))$$

and for the extreme value it is

$$F(z) = 1 - \exp(-\exp(z))$$

The extreme value distribution isn’t symmetric like the other two; in this form, it is skewed towards negative values.

For TYPE=BINARY, DDV estimates the coefficients by maximizing over $\beta$ the log likelihood function

$$L = \sum_{Y_i \neq 0} \log(F(X_i\beta)) + \sum_{Y_i = 0} \log(1 - F(X_i\beta))$$

As mentioned above, this is done using the Newton-Raphson algorithm, as the first and second derivatives are easy to compute. The estimated covariance matrix of coefficients is
\[ \left( -\frac{\partial^2 L}{\partial \beta^2} \right)^{-1} \] evaluated at \( \hat{\beta} \).

**TYPE=ORDERED** estimates \( m-1 \) cut points in addition to the regressor coefficients. The dependent variable can be coded as any increasing set of numbers; within **DDV** these get mapped to 1, \ldots, \( m \). Don’t include **CONSTANT** among your regressors; it would merely shift the cut points. The likelihood elements are generated from:

\[
P(\text{choose } j | X_i) = \begin{cases} 
F(a_1 - X_i'\beta) & \text{if } j = 1 \\
1 - F(a_{m-1} - X_i'\beta) & \text{if } j = m \\
F(a_j - X_i'\beta) - F(a_{j-1} - X_i'\beta) & \text{otherwise}
\end{cases}
\]

Again, the parameters are estimated by Newton-Raphson.

**TYPE=MULTINOMIAL** will always use the logit model. This estimates a separate set of coefficients for the \( m-1 \) choices with the highest values; the model is normalized on the choice with the lowest numerical value.

**TYPE=CONDITIONAL** will also use the logit model. As indicated above, you have to structure your data set with one observation for each combination of individual and choice. That is, if you have 400 individuals and 4 choices for each, your data set will have 1600 observations. A series identifying the individuals is required, and one identifying the choices is recommended: use the **INDIV** and **MODES** options to tell **DDV** which series they are. The **depvar** series shows whether an observation represents the choice made by the individual. Note that if a choice is unavailable to an individual, you don’t need to include an observation for it in the data set.

**DDV** calculates the likelihood elements individual by individual. If none of the observations are flagged (in **depvar**) as chosen, the individual will be skipped. You can use this to estimate a model over a restricted set of choices: if you use a **SMPL** which knocks out any observation for a particular mode, it will remove from consideration any individual who chose that mode.

**TYPE=COUNT** is the one model within this instruction which doesn’t model “choice.” Instead, it’s used for models of “count” data, where the dependent variable is required to be a non-negative integer, usually a small one. **DDV** estimates a Poisson regression model (see, for instance, Wooldridge (2002, section 19.2)), estimating the log of the mean of the Poisson distribution as a linear function of the regressors. The likelihood for an individual is given by

\[
\exp(-\exp(X_i \beta)) \exp(y_i \times X_i \beta)
\]
(This ignores the factorial term, which doesn’t interact with the coefficients). This is the only model type here where the estimates have some robustness to deviations from the distribution chosen, so using **ROBUSTERRORS** to correct the covariance matrix is reasonable.

**Examples**

```
ddv(smpl=(kids==0)) lfp
# constant wa agesq faminc we
```

estimates a binary choice probit for the observations where KIDS is zero.

```
ddv(dist=logit) union
# potexp exp2 grade married high constant
prj(dist=logit,cdf=fitlgt)
```

estimates a binary choice logit, and uses **PRJ** to get the predicted probabilities from the model.

```
ddv(type=multinomial,smpl=y87) status
# educ exper expersq black constant
```

estimates a multinomial logit.

```
ddv(type=count,robusterrors) children / resids
# educ age agesq evermarr urban electric tv constant
```

estimates a Poisson regression model, with a robust covariance matrix.

**Output**

**TYPE=COUNT** is similar to a linear regression, and has similar output. The rest all model probabilities. The following statistics are included in the output (**N** is the number of individuals, **y_i** is the observed value):

- **Log Likelihood**
  \[ \log L = \sum_{i=1}^{N} \log P(Y_i = y_i | X_i) \]

- **Average Likelihood**
  \[ \exp \left( \frac{1}{N} \sum_{i=1}^{N} \log P(Y_i = y_i | X_i) \right) \]

- **Base Likelihood**
  \[ \log L_b = \sum_{j=1}^{m} N \hat{p}_j \log(\hat{p}_j) \]

- **Pseudo-\(R^2\)**
  \[ 1 - \left( \frac{\log L_b}{\log L} \right)^{(2/N)\log L_b} \]
The average likelihood is the geometric mean of the likelihood elements. The base likelihood is the log likelihood of a model which predicts the observed probabilities of each choices; in most cases, this is the maximum of the likelihood without any slope coefficients. The Pseudo-$R^2$ is the measure of fit from Estrella (1998).

### Variables Defined by DDV

- **%BETA**: vector of coefficients (VECTOR)
- **%BETASYS**: vector of coefficients (includes all estimated parameters, not just the coefficients on the right-side terms) (VECTOR)
- **%CVCRT**: value of the convergence criterion (REAL)
- **%ITERS**: number of iterations completed (INTEGER)
- **%LOGL**: log likelihood (REAL)
- **%NOBS**: number of observations (INTEGER)
- **%NREG**: number of regressors (INTEGER)
- **%RSQUARED**: pseudo $R^2$ measure given above, except standard $R^2$ for TYPE=COUNT (REAL)
- **%SEESQ**: maximum likelihood estimate of the regression variance for TYPE=COUNT (REAL)
- **%STDERRS**: vector of coefficient standard errors (VECTOR)
- **%TSTATS**: vector of $t$-statistics of the coefficients (VECTOR)
- **%XX**: covariance matrix of coefficients (SYMMETRIC)
- **%XXSYS**: system covariance matrix ((includes terms for all estimated parameters, not just the coefficients on the right-side terms) (SYMMETRIC)

### Hypothesis Testing

You can apply the hypothesis testing instructions (**EXCLUDE, TEST, RESTRICT** and **MRESTRICT**) to estimates from DDV except that you can’t use **EXCLUDE** with TYPE=MULTINOMIAL. They compute the “Wald” test based upon the quadratic approximation to the likelihood function. RATS uses the second set of formulas in Section 6.2 of the User’s Guide to compute the statistics. Note that you cannot use the **CREATE** or **REPLACE** options on **RESTRICT** and **MRESTRICT**. You can also do likelihood ratio tests “by hand”: run restricted and unrestricted models and use **COMPUTE** and **CDF** to compute the statistic.
DEBUG — Debugging Tools

DEBUG offers options that can be useful in writing and debugging RATS procedures. Other options are used for developing and debugging the RATS software itself.

\[ \text{debug( options )} \]

Options (For Users)

symbols/[nosymbols]
Good programming practice dictates that variables defined in procedures and user-defined functions generally should be defined as being “local” variables in that procedure, rather than as global variables. This is done to avoid conflicts with global variable names used in the main program. The SYMBOld option helps identify variables in procedures and user-defined functions that are defined as global, rather than local, variables.

Execute the \text{DEBUG(SYMBOL)} instruction first, and then execute the commands that define your procedure or function. For procedures stored on separate files, you can use the \text{SOURCE} instruction to read them in.

RATS will display a list of any global variables found in the procedure or function. You can use \text{LOCAL} instructions in the procedure or function to define these as local variables instead.

table/[notable]
Displays a list of all variables, procedures, and functions currently in use. This includes both user-defined variables, and reserved variable and function names.

Options (RATS Development)

activate/[noactivate]
compile/[nocompile]
execute/[noexecute]
instruct/[noinstruct]
memory/[nomemory]
These are used internally by Estima for developing and testing the software.
DECLARE — Setting Data Types

DECLARE creates new variables with the specified type.

```
declare  type    list of names (and/or dimension fields)
```

**Parameters**

- **type**
  The data type you want to assign to the listed variables. For instance, REAL, RECTANGULAR, or VECTOR\[SERIES\].

- **list of names**
  The list of variables that will have type. Separate the names with blanks. Symbolic names must be unique: if you attempt to assign a new type to a name already in use (by you or by RATS) you will get an error message.

You can also set the dimensions of an array at the time you DECLARE it, by using a dimension field instead of simply the variable name—just provide the dimensions for the array in parentheses immediately after the variable name. Arrays can also be dimensioned later (or redimensioned) using DIMENSION.

**Description**

In most situations, you do not need to declare a variable name before using it. RATS will determine what type of variable is needed from the context. However:

- You will probably have to DECLARE any array whose values will be set in a COMPUTE instruction, to be sure that you get the array type that you want.
- Any “non-standard” aggregate type, such as a VECTOR\[SYMmetric\] or FRML\[COMPLEX\] will need a DECLARE.
- Any variable or array which you initialize with INPUT or ENTER needs DECLARE, since those instructions can handle a number of variable types.

You can, however, often dispense with a separate DECLARE instruction by putting the type specification directly into the instruction. For instance,

```
compute [symmetric] s=tr(a)*a
```
can be used instead of

```
declare symmetric s
compute s=tr(a)*a
```

Variables introduced with DECLARE (as well as any that are created from context by a RATS instruction) have global scope. They can be used anywhere later on in the program, and in procedures. You cannot change the type of such a variable later in the program.
Declare

Data Types Available

Basic Types:
- INTEGER, REAL, COMPLEX, LABEL, STRING, EQUATION, MODEL, PARMSET

Aggregate Types:
- VECTOR, RECTANGULAR, SYMMETRIC, SERIES, FRML

The aggregate types can be layered two deep, so you can have a VECTOR of FRMLs (written VECTOR[FRML]), or a RECTANGULAR of SERIES. All of these, when used by themselves, are for real numbers. However, you can create a VECTOR of COMPLEX numbers (VECTOR[COMPLEX]) or a FRML which evaluates to STRINGS (FRML[STRING]). See the User’s Guide, Section 4.3 for complete details on these.

Examples

```r
declare real a b c d e
declare integer count
declare vector[integer] counters(10)
declarge symm s(nvar,nvar)
```

These statements declare the following variables:

- A through E as scalar real variables.
- COUNT as a scalar integer variable.
- COUNTERS as a vector of integers. COUNTERS is also dimensioned to have 10 elements (referenced as element 1 through element 10).
- S as a symmetric array, with dimensions NVAR by NVAR (where NVAR has been set previously to some value)

```r
declare label product
declare vector[label]  product_codes(25)
```

declares PRODUCT as a label variable (that is, it can contain a single LABEL of up to 16 characters), and PRODUCT_CODES as a VECTOR of LABELS with 25 elements (each element will contain a single LABEL).

Deleting Declared Variables

You can delete variables from memory by using the Show All Symbols wizard, selecting the variable(s), and doing Edit–Delete. Because of their special nature, you cannot delete SERIES except by doing File–Clear Program, which clears everything.

Notes

You can dimension an array either on the DECLARE instruction or with a DIMENSION instruction later on. They can be redimensioned at any time, as references to elements of the array are resolved using the dimensions at the time it is needed, not at the time it was declared.

100  RATS Reference Manual
DEDIT — Initiating RATS Data File Editing

**DEDIT** allows you to create or alter a RATS format data file. Once you have opened a file with **DEDIT**, you can use the **CATALOG, STORE, EDIT, INCLUDE, DELETE, RENAME, PRTDATA, and UPDATE** instructions to work with the file. You must use the **SAVE** instruction to save any changes to the file, or use **QUIT** to close the file without saving your changes.

```
dedit( option )  file name
```

**Wizards**

The *Open RATSDATA* and *New RATSDATA* operations on the *File* menu are an alternative way to work with RATS data files interactively. They allow you to view, edit, and create data series, copy data from one file to another, export data, and more. See Section 1.1.2 in the *User’s Guide* and “RATSData” in the Help system for details.

**Parameters**

*file name* Name of the data file you want to edit or create. If you simply type **DEDIT**, RATS will prompt you for the file name.

**Option**

`new/ [nonew]`

Use the option **NEW** when you are creating a new file. If you forget to put **NEW** when creating a file, RATS will prompt you if it cannot find the file requested.

**Notes**

You can only edit one RATS format file at a time using **DEDIT**. If you have a RATS file open for editing, and then issue another **DEDIT**, RATS will ask you if you want to save the changes to the old file before opening the new file. You can, however, open other RATS format files for *input* (using **OPEN DATA**) while you are editing a RATS file. You can use the instruction `ENVIRONMENT RATSDATA=file name` in place of **DEDIT** for existing files.

**DEDIT** does not allow you to read data from the opened file. Use **OPEN DATA** (or `ENVIRONMENT RATSDATA=`) and **DATA** instructions to read data from RATS format files.

You can also use the command **COPY (FORMAT=RATS)** to write data to a RATS format file. With **COPY**, all the data must be written to the file with a single instruction (if a file with the same name already exists, it will be overwritten). With **DEDIT**, you can use several **STORE** instructions to write series to a file. If you do create a RATS file using **COPY**, you can always go back and use **DEDIT** and related instructions to add, remove, display, or edit series on the file.
Examples

dedit(new) results.rat
store testone testtwo control
save

creates a new RATS format file called RESULTS.RAT, includes the series TESTONE, TESTTWO and CONTROL on it, then saves the file.

RATS File Portability Issues

RATS has used the same file format since Version 4, so files created by versions since then are interchangeable. RATS format files are identical across platforms. For example, files created on a PC can be used without translation on Macintosh, UNIX, and LINUX systems.

When transferring files from one platform to another (by FTP or e-mail, for example), to avoid corrupting the file, be sure to use a “binary” transfer, rather than a “text” transfer.

See Also . . .

UG, Section 2.5 RATS format data files
QUIT Closes RATS file being edited without saving any changes.
SAVE Saves the changes made to a RATS format file.
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=regsamp)
```

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 file with `Open RATSData` on the `File` menu, the `Clear` 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 . . .

- *UG, Section 2.5* 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.
**DENSITY — Estimate Density Function**

**DENSITY** estimates the density function for a series of data. This can be done using kernel methods or by binning and counting for a histogram.

```
density(options) series start end grid density
```

**Parameters**

- **series**
  - The series for which you want to compute the density function.
- **start** and **end**
  - Range to use in computing the density. If you have not set a **SMPL**, this defaults to the defined range of **series**.
- **grid**
  - Series of points at which the density is estimated.
- **density**
  - Series for the estimated density corresponding to **grid**.

If **DENSITY** creates the grid series (**GRID=AUTOMATIC** option), the **grid** and **density** series will be defined from entry 1 until the number of points in the grid.

**Options**

- **type=[epanechnikov]/triangular/gaussian/logistic/flat/parzen/histogram**
  - See **Description** below.
- **counts/[nocounts]**
  - See **Description** below.
- **bandwidth=kernel bandwidth [see below]**
  - BANDWIDTH specifies the bandwidth for the kernel. The default value is
    
    \[0.79 \times N^{-1/5} \times IQR\]
    
    where \(IQR\) is the interquartile range (75%ile–25%ile) of the **series** and \(N\) is the number of data points.
- **grid=[automatic]/input**
  - If **AUTOMATIC**, the **grid** series runs in equal steps from the 1%-ile to the 99%-ile of the input series. If **INPUT**, you fill in the **grid** series with whatever values you want prior to using the **DENSITY** instruction. Note that the **grid** series does not have to be in increasing order. If the density is being used as part of a more involved calculation (see Section 12.4 of the User’s Guide), the **grid** series is usually a series created from the data.
- **maxgrid=number of grid points [100 except for type=histogram]**
  - If **GRID=AUTOMATIC** (the default), **MAXGRID** gives the number of equally spaced points at which the density is estimated.
Density

weight=series of weights for the data points [equal weights]
This can be used if the input data points aren’t weighted equally, due for instance, to oversampling or importance sampling. The weights do not have to sum to one—the rescaling will be done automatically.

derivative=series of estimated derivatives
This saves the estimated derivatives of the density function into a series. This matches up, point for point, with the grid and density series. It requires TYPE=GAUSSIAN or TYPE=LOGISTIC, as the other kernel types aren’t differentiable.

smpl=SMPL series or formula (User’s Guide, Section 5.2)
You can supply a series or a formula that can be evaluated across entry numbers. Entries for which the series or formula is zero or “false” will be skipped, while entries that are non-zero or “true” will be included in the operation.

print/[noprint]
If PRINT, DENSITY produces a table of grid values and the estimated density at each point.

Description
For types other than HISTOGRAM, DENSITY estimates the density function for a series of data $x_1,\ldots,x_T$, by computing at each point $u$ in the grid:

$$\hat{f}(u) = \sum_{i=1}^{T} \left( w_i K\left( \frac{u - x_i}{h} \right) \right) / h \sum_{i=1}^{T} w_i$$

where $K$ is the kernel function, $h$ the bandwidth and $w_i$ are the weights, which, by default, are 1 for all $t$. The kernel types take the following forms:

**EPANECHNIKOV**

$$K(v) = 0.75(1 - v^2) \text{ if } |v| \leq 1, \text{ and } 0 \text{ otherwise}$$

**TRIANGULAR**

$$K(v) = (1 - |v|) \text{ if } |v| \leq 1, \text{ and } 0 \text{ otherwise}$$

**GAUSSIAN**

$$K(v) = \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-v^2}{2}\right)$$

**LOGISTIC**

$$K(v) = e^v / \left(1 + e^v \right)^2$$

**FLAT**

$$K(v) = 0.5 \text{ if } |v| \leq 1, \text{ and } 0 \text{ otherwise}$$

**PARZEN**

$$K(v) = 4/3 - 8v^2 + 8|v|^3 \text{ if } |v| \leq 0.5, \quad 8\left(1 - |v|^3 \right)/3 \text{ if } 0.5 \leq v \leq 1, \text{ and } 0 \text{ otherwise}$$
As you increase the bandwidth, you will get a smoother estimated density function, but you will be less able to detect sharp features. A shorter bandwidth leads to a more ragged estimated density function, but sharp features, such as a truncation at one end, will be more apparent.

For `TYPE=HISTOGRAM`, the grid is turned into a series of “bins” centered at each grid point. `DENSITY` counts the number of data points which are in each bin. If you use the option `COUNTS`, these counts will be the values returned. By default, the counts are divided by the number of data points times the width of the bin to produce an estimate of the density.

**Examples**

```
density gdpgrowth / sgrid sdensity
density(type=histogram) gdpgrowth / hgrid hdensity
scatter(style=bargraph,overlay=line,ovsame,header="GDP Growth") 2
# hgrid hdensity
# sgrid sdensity
```

computes both a density estimate using the Epanechnikov kernel and a histogram. These are graphed, together with the smoothed density shown as a line overlaying the histogram.

```
data(unit=input) 1 12 gridpts
  5 15 25 35 45 55 65 85 105 135 165 235
density(type=histogram,grid=input,print) income / gridpts idensity
```

computes a histogram for an income series using an input set of interval midpoints which give wider intervals for the higher incomes.

```
set testdraw = %rangamma(4.0)
density(bandwidth=1.00) testdraw / x fx
set fg = exp(log(x)*3.0-x-%lngamma(4.0))
scatter(style=polygon,overlay=line,ovsame) 2
# x fg / 4
# x fx / 1
```

draws a series of random gamma variates, and computes an estimate of the density from the sample. Using the same grid points (the X series), FG is set to the true density function. The `SCATTER` instruction then displays the true density as a filled polygon, with the empirical density shown as a line overlaying it.

**Variables Defined**

%EBW  The computed bandwidth
DETERMINISTIC — Listing the “Other” Variables in a VAR

DETERMINISTIC is one of the subcommands of SYSTEM used in defining vector autoregressions. Use this instruction to list all the variables in the VAR which are not lags of the dependent variables. Usually these are deterministic variables, such as CONSTANT, trends and seasonal dummies, but they can be any variables. This puts the same set of variables in all equations of the system.

\[ \text{deterministic} \quad \text{list of deterministic variables} \]

Wizard

The VAR (Setup/Estimate) wizard on the Statistics menu provides an easy, dialog-driven interface for defining and estimating VAR models.

Parameters

\[ \text{list of ...} \quad \text{Lists the variables that you want to include in the equations.} \]
\[ \text{Use regression format (described in Section 1.4).} \]

Examples

```
system(model=deumodel)
variables deuip deum1 deutbill deucpi
determ constant seasonal{-10 to 0}
   lags 1 2 3 4 6 9 12 13
end(system)
```

This adds a constant and eleven seasonal dummies to the equations of a four variable VAR. See SEASONAL for details on using leads of a single seasonal dummy variable.

```
system(model=unrestricted)
variables gdpq unemprate gpdi
   lags 1 to 4
det constant gdpdefl{1 to 4} m2{1 to 4}
end(system)
```

This example includes the lags of GDPDEFL and M2 as exogenous variables in the unrestricted model, allowing us to test the hypothesis that they do not enter the equations for GDPQ, UNEMPRATE, and GDPI.

Notes

The ECT instruction is available as a subcommand of SYSTEM to define error correction terms. These no longer have to be put into the system using DETERMINISTIC.

See Also . . .

User’s Guide, Ch. 10 Vector Autoregressions.
SYSTEM First instruction for setting up a VAR.
DIFFERENCE — Time Series Differencing

DIFFERENCE takes a combination of consecutive \((X_t - X_{t-1})\) and seasonal \((X_t - X_{t+s})\) differences, or a fractional difference, of a series. It can also extract the mean.

\[
\text{difference( options ) series start end newseries newstart}
\]

Wizard

The Differencing wizard on the Data menu provides an easy, dialog-driven interface to the differencing capabilities of the DIFFERENCE instruction.

Parameters

- **series**: Series to difference.
- **start** and **end**: Range of entries over which to difference the series. **start** must allow for the lagged values needed. If you have not set a SMPL, these default to the maximum allowed by **series**. If you have set a SMPL, you probably will need to reset it before using the differenced series.
- **newseries**: Resulting series. By default, **newseries=series**.
- **newstart**: Starting entry for **newseries**. By default, **newstart=start**.

Options

- **differences=number of regular differences** [see below]
- **sdifferences=number of seasonal differences** [0]
- **span=span for seasonal differences** [CALENDAR seasonal]

Use these to specify the differencing operators to apply. The default, if you use neither option, is a single regular difference (DIFFERENCES=1). If you use the SDIFFERENCES option, DIFFERENCES defaults to zero. So, if you want to use a combination of regular and seasonal, you must use both options.

- **standardize/ [nostandardize]**
- **center/ [nocenter]**

The CENTER option centers the series by subtracting its mean from each observation \((Y_t = X_t - \bar{X})\). STANDARDIZE subtracts off the mean and divides by the standard deviation \((Y_t = (X_t - \bar{X})/s)\). If you use one of these with the DIFFERENCES or SDIFFERENCES options, it will be applied to the results of the differencing operation. If you use one of these with the FRACTION option, the centering or standardization will be done first.
fraction=d (fractional differencing parameter) [unused]
padding=padding value for out-of-sample observations [0.0]

To do fractional differencing, use the FRACTION option to supply a value for the fractional differencing parameter d.

Fractional differencing is an infinite filter in the time domain, so finite real-valued data must be padded at either end before the filter can be computed. You can use the PADDING option to supply the value to be used to pad the data—normally you would make this the mean value of the series being filtered. The default value of zero is appropriate for a series with a zero mean. See Section 15.9 to see how the differenced series is calculated.

Description

If \( d = \text{regular differences} \) and \( e = \text{seasonal differences} \), and \( s = \text{span} \), and \( L \) represents the lag (or backshift) operator, DIFFERENCE computes

\[
Y_t = (1 - L)^d (1 - L^s)^e X_t
\]

For frequencies defined in terms of the number of periods per year, SPAN defaults to the CALENDAR seasonal (for example, SPAN=12 for monthly data). For frequencies like weekly and daily, where there is no clear definition of a seasonal span, SPAN defaults to 1.

Examples

data 1959:1 2002:12 mlnsa
set mlnsa = log(mlnsa)
diff mlnsa / m11diff
diff(sdiffs=1) mlnsa / m11sdiff
diff(diffs=1, sdiffs=1) mlnsa / m1both

This computes the first difference, the first seasonal difference and a combination of one difference and one seasonal difference for the log of M1NSA. M11DIFF is defined from 1959:2, M11SDIFF from 1960:1 and M1BOTH from 1960:2.

diff(fraction=d, pad=uupresample) uu / uufilter

does fractional differencing on seriesUU producing UUFILTER. The out-of-sample values are treated as having the value UUPRESAMPLE.
Notes

The instruction **BOXJENK** includes the **DIFFS**, **SDIFFS** and **SPAN** options which allow you to use the *undifferenced* form of the dependent variable in the estimation procedure. **BOXJENK** handles the differencing itself prior to estimating the ARMA parameters, then rewrites the estimated equations in terms of the original dependent variable.

If you estimate a regression on a differenced series (whether with **BOXJENK** or **LINREG** or any other instruction), and want the equivalent equation using the original series, use the instructions **MODIFY** and **VREPLACE**.

For instance, the following

```
diff mlnsa / m1ldiff
linreg(define=diffeq) m1ldiff
# constant m1ldiff{1 to 12}
modify diffeq
vreplace m1ldiff with mlnsa diff 1
```

estimates a twelve lag autoregression on the first difference of M1NSA, then recasts the equation **DIFFEQ** in terms of M1NSA itself by substituting out **M1LDIFF**.

The instruction **FILTER** can handle more general types of linear filtering.

Variables Defined (only with **STANDARDIZE** or **CENTER** options)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%NOBS</td>
<td>Number of observations</td>
</tr>
<tr>
<td>%MEAN</td>
<td>Mean of the series</td>
</tr>
<tr>
<td>%VARIANCE</td>
<td>Variance of the series</td>
</tr>
</tbody>
</table>
DIMENSION — Setting Array Sizes

**DIMENSION** sets the sizes (dimensions) of arrays. You must introduce an array prior to dimensioning it. For convenience, you can also do the dimensioning on a `DECLARE` or `LOCAL` instruction.

```
dimension  array(dimensions)  (separate multiple arrays by blanks)
```

**Parameters**

`array(dim.)` The list of arrays you want to dimension, with the dimensions for each array listed in parentheses. You can dimension any number of arrays with a single instruction. The arrays dimensioned do not have to be of the same type.

**Dimension fields**

A dimension field has one of the forms:

- `name(exp)` for one-dimensional arrays (VECTOR arrays).
- `name(exp1,exp2)` for the two-dimensional arrays: RECTANGULAR, and SYMMETRIC, and PACKED. This sets up an $exp1 \times exp2$ matrix. ($exp2$ is ignored for SYMMETRIC, since such a matrix must be square).

The dimension expressions can be any integer valued expression, not just constants.

**Examples**

```
compute n=6
declare rect  s1 s2
declare vector  v1
dimension v1(10) s1(n,3) s2(n**2+n,n+17)
```

This declares and dimensions the arrays $V1$, $S1$, and $S2$. $V1$ is a 10-element VECTOR, $S1$ is a 6 by 3 RECTANGULAR and $S2$ is a 42 by 23 RECTANGULAR.

```
declare vector[rectangular] vv(10)
do i=1,10
   dim vv(i)(5,5)
end do i
```

When you have an array of arrays, as in this case, you must dimension each of the component arrays separately.

**Notes**

You can **DIMENSION** an array several times during a program, but re-dimensioning an existing array erases the previous contents of the array. The **OVERLAY** instruction is an alternative to **DIMENSION**. It dimensions an array while setting its location within an existing array.
**DISPLAY — Displays Computed Values**

**DISPLAY** prints scalars and expressions in a flexible way. You can include strings of characters in quotes to label the results, alter the position of fields and add editing information.

```plaintext
display( options ) variables, strings, picture/position/tab codes
```

**Parameters**

- **variables**
  - These can be variables (such as those set with **COMPUTE**) or expressions to be evaluated.

- **strings**
  - Strings of characters enclosed in single or double quotes. Any trailing blanks on a string are discarded: use a position code if you want more than one space to separate two fields.

- **picture codes**
  - A picture code takes a form such as `#####` for integers and `#####.##` or `######` for reals. It provides the representation format for the next variable printed. See below for details.

- **position/tab codes**
  - By default, each of the fields (variables or strings) is followed by exactly one space. Position and tab codes provide control over the positioning of fields, and are particularly useful in formatting tables of output. See the next page for details.

**Options**

- **unit=** [output]/copy/Other Unit
  - Use the **UNIT** option if you want the results to go somewhere other than the current output.

- **store=** string variable
  - This saves the created line in a string variable; it does not print it. You can use this variable, for instance, as a header for a graph. (Often, you can do the same operation with a **COMPUTE** instruction.)

- **hold/ [nohold]**
  - If you use **HOLD**, **DISPLAY** creates the string and waits for another **DISPLAY** instruction to add more to it. This can be very useful when the number of objects that you want to display is not known in advance.

**Picture Codes**

A picture code takes a form such as `#####` for integers or `###.##`, `#####` or `*.##` for reals. It provides the format for the next variable or variables displayed. A picture code with no decimal point causes the number to be printed right-justified in a field.
whose width is the number of # signs. A code with a decimal point produces a number the width of the total string, with the decimal point in the indicated location. If you use * left of the decimal point, it uses as many places as needed to show the number and no more. For instance, ###.## will display 1.23 with two blanks padding the field on the left, while * .## will display it using just the four characters.

You can get a picture code from a string variable or expression using the notation &expression, where the expression should evaluate to a string with one of the forms shown above. Two functions which can prove quite useful in formatting tables are %BESTREP(x,w) and %MINIMALREP(x,w), where x is a real array and w is the field width (note that the new REPORT instruction may be easier to use than a sequence of DISPLAYs). %BESTREP returns a picture code which best represents the data in x in a field of width w. %MINIMALREP returns the smallest picture code which represents the data in x accurately using at most w positions. The difference between the two is that %BESTREP will always use up the whole width, usually by adding digits right of the decimal place, while %MINIMALREP will give a shorter picture if it can.

A picture code stays in effect in a particular DISPLAY instruction until overridden by another picture code. To “cancel” a picture code, just use a field with * by itself.

Position Codes

By default, each of the fields is followed by exactly one space. Position codes allow you to alter the placement of the field immediately following it. They take one of the forms:

@n display next field at position n
@+n display next field n positions to the right
@-n display next field n positions to the left

The last two move from the blank space after the previous field. For instance, to run two fields together, use a position code of @-1. Position codes can use expressions, for instance, @13*I will place the next field at position 13*I.

Tab Codes

A tab field takes the form @@[flag]spacing. The flags are

. decimal
> right
< left (the default if no flag is included)
^ center

The spacing starts relative to the current position. Subsequent fields will be positioned with the “flagged” feature placed spacing positions apart. For instance,

display @@.12 *.## xvect

will display the elements of xvect with the first decimal place in position 12, the second at 24, etc; each number displayed with two digits right of the decimal point.
Examples

display  "Test statistic is"  ((rssr-rssu)/q)/(rssu/%ndf)
display  "Degrees of freedom are"  q  "and"  %ndf

which will produce something like

Test statistic is  3.92092
Degrees of freedom are  13 and  123

If you want to make this look a little cleaner (when it’s displayed), you could do something like

display  "F( q , %ndf ) ="  ((rssr-rssu)/q)/(rssu/%ndf)

which will give you

F( 13 , 123 ) =  3.92092

A further improvement (which you might do if you’re writing this as part of a procedure which others will use), is to squeeze out the extra spaces and drop some of the excess digits on the result by using

display  "F(+q+","+%ndf +") ="  *.###  ((rssr-rssu)/q)/(rssu/%ndf)

to produce

F(13,123) = 3.921

display  @12 "Method 1" @32 "Method 2"
display  @10 ######.#####  t1 @30 t2

will display

Method 1          Method 2
-3.17235       11234.20921

dec vect[labels] methodlabl(n)
do  i=1,n
    compute methodlabl(i)="Method "+i
end  do i
display  @@^12 methodlabl

shows the strings “Method 1”, “Method 2”, ... centered every 12 positions.

See Also ...

REPORT   Organizes output into a table format.
WRITE    Displays arrays and other variables. It is more general but not as flexible as DISPLAY.
MESSAGEBOX Displays information messages requiring user response.
INFOBOX  Displays informational messages and progress bars—does not require a user response.
DLM — Dynamic Linear Models

The instruction `dlm` applies one of several methods to dynamic linear models. These are models with a linear transition from one period to the next. You can use `dlm` to solve for unknown states or control variables, or to estimate unknown parameters in the transition matrices. See Section 12.2 in the *User's Guide* for more information.

```
dlm( options ) start end state vectors state variances
```

### Parameters

- **start**  **end**  Range to use in estimation. *Make sure you define these.*
- **state vectors**  A SERIES of VECTORS into which the output state vectors are placed. If you call this STATES, for instance, STATES(2007:1) is the complete estimated state vector for 2007:1. To get component $k$ of this, use STATES(2007:1)(k).
- **state variances**  A SERIES of SYMMETRIC arrays into which the variance matrices of the states are saved.

### Options

Most of the options define information needed in the model. For state-space/signal extraction, the model takes the form

$$X_t = A_t X_{t-1} + Z_t + F_t w_t,$$

while for optimal control the model takes the form

$$X_t = A_t X_{t-1} + B_t U_t + F_t w_t,$$

while the objective function being

$$E\left( X_0' Q_0 X_0 + \sum_{i=1}^{T} [X_i' Q_i X_i + U_i' R_i U_i] \right).$$

In both types of models, $X_t$ is an unobserved vector of states. $Y_t$ is observable, and gives information about $X_t$ through the measurement equation (2) or (4). $w_t$ and $v_t$ are shocks to the transition process and noise in the measurement equation, respectively. $F_t$ are the loadings from the transition shocks to the states; typically the identity matrix. The $Z_t$ term in (1) allows for exogenous shifts in the state equation (Note: you can incorporate exogenous variables into the measurement equation by simply adjusting the $Y$ formula to account for the shift). In the optimal control problem, $U_t$ are the control variables which are being calculated by `dlm`. 

116  *RATS Reference Manual*
In these descriptions, \( N \) is the size of \( X \), \( M \) is the size of \( Y \), \( L \) is the size of \( W \), and \( P \) is the size of \( U \).

\( a = \text{RECTANGULAR or FRML[RECT]} \quad \text{[identity matrix]} \)

This gives the \( A \) matrices. The expression should evaluate to an \( N \times N \) matrix.

\( c = \text{RECTANGULAR or FRML[RECT]} \quad \text{[zero matrix]} \)

This gives the \( c \) matrices—it should evaluate to an \( N \times M \) matrix.

\( y = \text{VECTOR or FRML[VECT]} \quad \text{[no measurement equation]} \)

This gives the \( Y \) vectors—it should evaluate to an \( M \) vector.

\( z = \text{VECTOR or FRML[VECT]} \quad \text{[no exogenous state equation terms]} \)

This gives the \( Z \) vectors, used to specify exogenous shifts in the state equation—it should evaluate to an \( N \) vector.

\( \text{sw} = \text{SYMMETRIC or FRML[SYMMETRIC]} \quad \text{[zero]} \)

This gives the covariance matrices of the \( w \)'s—it should evaluate to an \( L \times L \) SYMMETRIC.

\( \text{sv} = \text{SYMMETRIC or FRML[SYMMETRIC]} \quad \text{[zero]} \)

This gives the covariance matrices of the \( v \)'s—it should evaluate to an \( M \times M \) SYMMETRIC.

\( f = \text{RECTANGULAR or FRML[RECTANGULAR]} \quad \text{[\( N \times N \) identity]} \)

These are the loadings from the \( w \)'s to the \( X \)'s—it should evaluate to an \( N \times L \) matrix.

\( x_0 = \text{VECTOR or FRML[VECTOR]} \quad \text{[zero vector]} \)

This gives the initial \( X \) vector—it should evaluate to an \( N \) vector.

\( \text{sx}_0 = \text{SYMMETRIC or FRML[SYMMETRIC]} \quad \text{[zero]} \)

This gives the covariance of \( X_0 \)—it should evaluate to an \( N \times N \) SYMMETRIC array.

\( b = \text{RECTANGULAR or FRML[RECT]} \quad \text{[zero matrix]} \)

This gives the \( B \) matrices. The expression should evaluate to an \( N \times P \) matrix.

\( q = \text{SYMMETRIC or FRML[SYMMETRIC]} \quad \text{[zero]} \)

This gives the \( Q \) matrices—it should evaluate to an \( N \times N \) SYMMETRIC.

\( r = \text{SYMMETRIC or FRML[SYMMETRIC]} \quad \text{[zero]} \)

This gives the \( R \) matrices—it should evaluate to a \( P \times P \) SYMMETRIC.
type=[filter]/smooth/control/simulate/csimitate
This determines what technique is to be used. FILTER is the Kalman filter, SMOOTH is the Kalman smoother and CONTROL solves the control problem.

TYPE=SIMULATE does a random (Normal) simulation of the DLM drawing randomly from the presample distribution, the state disturbances and the measurement equation disturbances. TYPE=CSIMULATE does a conditional simulation, drawing the states from their distribution conditional on the observed Y’s.

smpl=SMPL series or formula (User’s Guide, Section 5.2)
You can supply a series or a formula that can be evaluated across entry numbers. Entries for which the series or formula is zero or “false” will be skipped, while entries that are non-zero or “true” will be included in the operation. See the note later concerning missing values.

exact/[noexact]
g=matrix (RxN, R<N) reducing model to stationarity [not used]
sh0=variance of diffuse part of the prior [identity if EXACT]
presample=ergodic/[x0]/x1
These options can be used to initialize the pre-sample states of the Kalman filter. See Section 12.2 of the User’s Guide for the technical details.

The EXACT option does Exact Initial Kalman filter/smoothing as described in Koopman (1997). If you use EXACT, the G option can provide a matrix which maps the states to a set of stationary states; by default, the entire state vector is treated as non-stationary. Alternatively, you can use the SH0 option to set directly a (proportional) matrix for the variance of the diffuse part of the prior.

If PRESAMPLE=X0 (the default), the initial (finite) state mean and covariance matrix are supplied by the X0 and SX0 options. If PRESAMPLE=X1, X0 and SX0 are still used, but they provide X10 and Σ10, not X00 and Σ00. If you use PRESAMPLE=ERGODIC, DLM assumes that A and Z are time-invariant and uses the stationary solution. If A has unit eigenvalues, PRESAMPLE=ERGODIC will automatically switch to a fully or partially diffuse initialization.

variance=[known]/concentrated/chisquaredinverse
scale/[noscale] (old option—use VARIANCE instead)
VARIANCE=KNOWN assumes all variances are known (or being estimated).
VARIANCE=CONCENTRATED assumes all variances are known up to a single unknown scale factor (usually the variance of the measurement equation) which is to be concentrated out. VARIANCE=CHISQUARED assumes all variances are known up to an unknown scale factor (again, usually the variance of the measurement equation) which has an informative (inverse) chi-squared prior distribution—see the PDF and PSscale options below.
When you use VARIANCE=CONCENTRATED or VARIANCE=CHISQUARED, you will usually peg one of the variances to 1.0, such as with SV=1.0.

The SCALE/NOSCALE option used in versions before 7.0 has been replaced (though it will still work) by the VARIANCE option. SCALE is equivalent to VARIANCE=CONCENTRATED and NOSCALE to VARIANCE=KNOWN.

pdf=prior degrees of freedom [not used]
pscale=prior scale factor [not used]
sighistory=series of estimated scaling variances [not used]
dfhistory=series of degrees of freedom [not used]

These are all used only with VARIANCE=CHISQUARED. PDF and PSSCALE are the prior degrees of freedom and prior scale factor for the scaling variance, respectively. SIGHISTORY returns a series of estimated scaling variances, while DFHISTORY returns a series of degrees of freedom.

discount=discount value [not used]
Multplies ∑(t|t−1)×discount. This is an alternative to using SW; instead of the change in variance being additive, it will be multiplicative.

startup=expression evaluated at period "start" [not used]
You can use this to provide an expression which is computed only for the first entry of the estimation range, before the regular formulas are computed. The main value of this is to allow you to do just once per function evaluation any time-consuming calculations which depend upon the parameters, but not upon time. This can be an expression of any type.

free=number of (fixed) states which are to be freely estimated [0]
Use this for models where part of the state vector is a set of unknown regression coefficients which are fixed over the sample, and which you want to be, in effect, estimated freely. This will adjust the likelihood to be the likelihood conditional on those parameters, rather than the unconditional one. (The states themselves are unaffected by this). You must arrange your state vector so that these will be at the end. Adding these to the state vector and using FREE is generally much more efficient than putting them in the parameter set and estimating them that way.
Options for Extracting Information from DLM

[print]/noprint
vcv/[novcv]
title="title for output" ["DLM"]
These are the same as for other estimation instructions.

likelihood=series of log likelihoods
This saves the cumulated log likelihoods into a SERIES.

yhat=SERIES[VECT] of predicted or simulated values of Y
Series of VECTORS of one-step predicted values of Y (with TYPE=FILTER or TYPE=SMOOTH), or simulated values of Y (with TYPE=SIMULATE).

what=SERIES[VECT] for predicted observables [not used]
swhat=SERIES[SYMM] for pred. cov. matrices [not used]
WHAT returns a SERIES[VECT] of the one-step prediction errors for Kalman filtering, the smoothed prediction errors for Kalman smoothing, and the simulated disturbances for the measurement equation for the simulations. SVWHAT returns a SERIES[SYMM] of the one-step prediction error variance for Kalman filtering, and the smoothed prediction variance for Kalman smoothing.

what=SERIES[VECT] of state disturbances
swhat=SERIES[SYMM] of state variances
These save the expected values of the disturbances and variances in the state equation given information through time $t$ (with TYPE=FILTER) or through time $T$ (with TYPE=SMOOTH), or the simulated disturbances and variances when doing simulations (with TYPE=SIMULATE).

Options for Estimating Parameters

method=bfgs/simplex/genetic/gauss/[solve]
pmethod=bfgs/[simplex]/genetic/gauss
piter=number of PMETHOD iterations to perform [none]

METHOD selects the estimation method used by DLM. If you choose any of these other than SOLVE, it is assumed that there are free parameters to be estimated (these need to be defined ahead of time with NONLIN).

BFGS is Broyden, Fletcher, Goldfarb and Shanno; SIMPLEX is the simplex algorithm; GENETIC is a genetic search algorithm; and GAUSS is Gauss–Newton. See Chapter 7 in the User’s Guide for a technical description of these. SIMPLEX and GENETIC are derivative-free methods which can compute point estimates of the coefficients but not standard errors.

Use PMETHOD and PITER if you want to use a preliminary estimation method to refine your initial parameter values before switching to one of the other estima-
tion methods—RATS will automatically switch to the “METHOD” choice after completing the preliminary iterations requested via PMETHOD and PITERS.

**parmset=PARMSET** to use [default internal]
This option selects the parameter set to be estimated. See the User’s Guide, Section 7.6, for more on PARMSETs. RATS maintains a single unnamed parameter set which is the one used for estimation if you don’t provide a named set.

**hessian=initial guess for inverse Hessian** (METHOD=BFGS only)
You can use this with METHOD=BFGS. Without it, DLM will start with a diagonal matrix whose elements are the reciprocals of the (numerically computed) second derivatives of the function. See Section 7.2 in the User’s Guide.

**iterations=iteration limit [100]**
**subiterations=subiteration limit [30]**
**cvcrit=convergence limit [.00001]**
**trace/[notrace]**
ITERATIONS sets the maximum number of iterations, SUBITERS sets the maximum number of subiterations, CVCRIT the convergence criterion. TRACE prints the intermediate results. For METHOD=SIMPLEX, an “iteration” is actually defined as $K$ vertex changes, where $K$ is the number of free parameters. This makes the number of calculations per “iteration” similar to the other methods.

**condition=number of early sample data points to skip [0]**
If you have a non-stationary model, it usually takes several data points for the variance of the state vector to become “finite” (assuming a diffuse prior). You can use the CONDITION option to indicate how many early data points should be left out of the calculation of the criterion function. The alternative to CONDITION is to use the EXACT option to deal explicitly with the diffuse initial conditions.

**reject=expression with “rejection” zone for parameters [not used]**
If the expression evaluates to a non-zero (“true”) value, the function is immediately assigned the missing value.

**Missing Values**
A missing data point can be represented by either a $c_t$ or $Y_t$ which has a missing value. You can also use a SMPL option to skip data points. Note that the missing time periods are still considered part of the overall sample. The Kalman filter and smoother will estimate the state vector at those time periods and optimal control will generate a control value for them.

Note that if $Y$ has more than one component, some can be missing, while others aren’t. DLM will adjust the calculation in those cases to use the information available.
Declaring the Options

Most of the time, the options which construct your model will either take the default value or they will be constant over the sample. If this is the case, you can put them in as a matrix or as a constant value (for a 1×1 matrix). If, however, the matrix is time-varying (at minimum, $Y$ usually is), or it depends upon a non-linear parameter which you are estimating, you will either have to define it as a FRML of the correct type, or put its definition directly into the DLM option.

Examples

(6) $y_t - \mu = \phi(y_{t-1} - \mu) + \epsilon_t + \theta \epsilon_{t-1}$

can be defined by the following matrices

\[
X_t = \begin{bmatrix} y_t - \mu \\ \epsilon_t \end{bmatrix}, \quad A = \begin{bmatrix} \phi & \theta \\ 0 & 0 \end{bmatrix}, \quad c' = [1 \ 0], \quad w_t = \begin{bmatrix} \epsilon_t \\ \epsilon_{t-1} \end{bmatrix}
\]

If the variance of $\epsilon_t$ is set to one, we can use VARIANCE=CONCENTRATE in estimating this model. Assuming that $|\phi| < 1$, this is a stationary model, which can be initialized using PRESAMPLE=ERGODIC. This sets up and estimates the model.

```plaintext
nonlin mu phi theta
dec frml[rect] af
compute [symm] sw=||1.0|1.0,1.0||
frml af = ||phi,theta|0,0||
sstats(mean) / y>>mu
compute phi=theta=.10
dlm(var=concentrate,presample=ergodic,method=bfgs,$
   a=af,y=y-mu,c=||1.0|0.0||,sw=sw) 1960:1 2006:4
```

(8) $y_t = \varphi_1 y_{t-1} + \varphi_2 y_{t-2} + \cdots + \varphi_p y_{t-p} + \epsilon_t$

If $p$ is known and fairly small, it’s probably easiest to just code the $A$ matrix directly. We show here how to do it for a general $p$. The transition equation matrices are

\[
X_t = \begin{bmatrix} y_t \\ y_{t-1} \\ \vdots \\ y_{t-p+1} \end{bmatrix}, \quad A = \begin{bmatrix} \varphi_1 & \varphi_2 & \cdots & \varphi_{p-1} & \varphi_p \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}, \quad w_t = \begin{bmatrix} \epsilon_t \\ 0 \\ \vdots \\ 0 \end{bmatrix}
\]

If the coefficients are known and aren’t going to be estimated, $A$ can be set up with an EWISE instruction with an $\%IF$ to handle the first row. Assume that the coefficients are in a vector named PHI.
dec rect a(p,p)
ewise a(i,j)=\text{if}(i==1,\phi(j),(i==j+1))

However, if the coefficients are to be estimated, the “ewise” has to be done within the defining formula. The easiest way to do that is to create a \text{FUNCTION} which returns the matrix. Because the \( A \) matrix doesn't depend on time (just on the parameters), it's a good candidate for being computed using the \text{STARTUP} option. In the \text{DLM} instruction below, this calls \text{AFUNC} with the current values of the parameters, puts the result into the matrix \( A \), which is then used by the \( A \) option.

\begin{verbatim}
compute p=5
dec vect phi(p)
nonlin phi *
function afunc phi
type vector phi
type rect afunc
local integer i j
dim afunc(%rows(phi),%rows(phi))
ewise afunc(i,j)=\text{if}(i==1,\phi(j),i==j+1)
end afunc

dlm(var=concentrate,presample=ergodic,method=bfgs,startup=(a=afunc(phi)),$ a=a,y=y,c=%unitv(p,1),f=%unitv(p,1),sw=1.0) 1960:1 2006:4
\end{verbatim}

\textbf{Variables Defined by DLM}

\begin{itemize}
\item \%FUNCVAL, \%LOGL \hspace{1cm} The log likelihood for \text{TYPE=FILTER} or \text{TYPE=SMOOTH}, or the value of equation (5) for \text{TYPE=CONTROL} (real)
\item \%VARIANCE \hspace{1cm} If \text{SCALE}, this is the (maximum likelihood) estimate of the variance scale (real)
\end{itemize}

The following are only defined when estimating coefficients:

\begin{itemize}
\item \%BETA \hspace{1cm} Coefficient vector (\text{VECTOR})
\item \%XX \hspace{1cm} Covariance matrix of coefficients, or \( XX^{-1} \) (\text{SYMMETRIC})
\item \%TSTATS \hspace{1cm} Vector containing the \( t \)-stats for the coefficients (\text{VECTOR})
\item \%STDERRS \hspace{1cm} Vector of coefficient standard errors (\text{VECTOR})
\item \%NOBS \hspace{1cm} Number of observations (integer)
\item \%NREG \hspace{1cm} Number of regressors (integer)
\end{itemize}
DO — Simple Loops Over an Index

A `DO` loop repeats a set of instructions with an integer-valued index changing in a systematic way. It is like a FOR–NEXT loop in BASIC or a Fortran DO loop. `DOFOR` is a similar instruction which loops over a list of values. See Section 16.1 of the User’s Guide for a discussion of the various looping instructions available in RATS.

```
do  index = startvalue, endvalue, increment
    Instructions executed for each value taken by index
end do
```

### Parameters

- **index**: An integer-valued variable or array element. At the end of each pass through the loop, RATS increments this variable by the `increment` value.

- **startvalue**: The initial (integer) value for `index` to take. It can be a constant, another variable, or an integer-valued expression.

- **endvalue**: The terminating (integer) value for the `index`. RATS executes the instructions in the loop as long as:
  
  ```
  index ≤ endvalue when increment > 0
  index ≥ endvalue when increment < 0
  ```

  Note that RATS will not execute the loop a single time if `startvalue` is greater than `endvalue` (or less than `endvalue` for negative `increment`).

- **increment** (Optional: Default is 1). This is an (integer) value by which `index` changes with each pass through the loop. It can be a constant, variable or expression, and it can be positive or negative.

### Examples of Syntax

- `do step=0,6
  STEP=0,1,2,...,6`

- `do count=5,0,-1
  COUNT=5,4,...,0`

- `do j=1979:1,enddata,4
  J=1979:1,1980:1 (with quarterly data).... as long as J ≤ ENDDATA`

### Nesting Loops

You can nest loops, that is, put one loop inside of another. Each `DO` must have its own `END DO` instruction. For example:
do iters=1,500
   do steps=0,6
      ...
   end do steps
end do iters

The indenting that we used in this example is not necessary, but makes the code easier to follow. RATS ignores any text (on the same line) after an END DO, so you can add comments to indicate which loop is “ended” by each END DO. In the example above, we’ve used the names of the two index variables (STEPS, ITERS) as comments.

Notes

You should never change the value of the index variable within the loop. RATS determines the number of trips that it will make through the loop when it first executes the DO instruction. This will not be affected by changes you make to the index. If you need a more flexible loop form, use WHILE or UNTIL.

Also, avoid using the reserved variable T as a loop index, because it is reserved for use by SET and FRML instructions (which can change the value of T).

The instruction DOFOR is similar to DO, but loops over a list of values, and can have a loop index which isn’t an integer. The function %DO can be used to put “looping” subcalculations into a larger calculation. For instance,

frml archpart = v=0.0, %do(i, 1, p, v=v+b(i)*u(i)**2), v

will compute (for a value of T) a sum involving P lags of the series U.

Examples

do ar=0,3
   do ma=0,3
      boxjenk(diffs=1,constant,ar=ar,ma=ma,maxl) $
         rate 1975:4 2002:4
   end do ma
end do ar

estimates 16 ARIMA models, one for each possible combination of p and q between 0 and 3.

dec vect[symm] xxx(10)
do i=1,10
   compute xxx(i) = %zeros(n,n)
end do i
creates an array of 10 SYMMETRIC arrays and makes each of them an NxN matrix of zeros.
DOFOR — Looping Over a List

A **DOFOR** loop repeatedly executes all of the instructions down to the matching **END**, with *index* taking a new value from the *list of values* on each pass through the loop. This continues until the list is exhausted.

While the *index* of a **DO** loop must be an integer variable, a **DOFOR** *index* can be almost any type of variable. This can be very useful in a wide variety of situations. See the examples for suggestions.

```
dofor index = list of values
    Instructions executed for each value taken by index
end dofor
```

**Parameters**

*index*  
The variable that takes each of the values in the list in turn. This can have any data type but if not previously declared it will be an **INTEGER**.

*list of values*  
The list of values *index* is to take. These values must be compatible with *index*. See the rules in the next paragraph.

**List of Values**

- if *index* is an **INTEGER**, they may be any integer-valued variables or expressions. You can use “n TO m” as shorthand for consecutive integer values.
- If *index* is a **SERIES** or an **EQUATION**, they may be series or equation names, or any integer-valued variables or expressions. Integer values are interpreted as series or equation numbers.
- if *index* is any other type, they must be variables or expressions which can be converted to the correct type.

With any type of index, you can use an array aggregate of that type in the *list of values*: for example, you can use **VECTOR** of **INTEGERS** with an **INTEGER** index. **DOFOR** will loop over all the elements in that array.

**Notes**

*index* is used as a “placeholder”: its value at the end of execution of the loop is unchanged from what it was at the beginning.

You can use an **INTEGER** index even if you intend to loop over a list of series. That is the preferred way of handling a list of series, since you don’t have to create a new series (which **DECLARE SERIES** will do) solely for the purpose of serving as a loop index. **RATS** converts a series name into its “number” and will convert it back to reference a series if necessary. If you need to refer to the series in a **SET** or **FRML**
DoFor

instruction, however, explicitly tell RATS that you intend the index to represent a series by using \texttt{index(0)} rather than \texttt{index} alone the way you could with an ordinary series name.

**Examples**

\begin{verbatim}
smpl 1947:1 2006:12
dofor i = gnp m1 cpi wage trddol importpr
    set i = log(i{0})
    stats i
end dofor i
\end{verbatim}

replaces six series by their natural logarithms and computes their sample statistics. In this example, \texttt{I} is an integer which is the \textit{number} of a series on each pass through the loop. On the left side of the \texttt{SET} and on the \texttt{STATS}, RATS \textit{expects} a series, so it automatically treats the integer into the series with that number. In the expression in \texttt{SET}, however, \texttt{I} by itself is perfectly acceptable as a single integer. We need to include the lag notation in \texttt{I(0)} so RATS treats it as a reference to the current entry of series number \texttt{I}.

\begin{verbatim}
smpl 1947:1 2006:12
dofor i = gnp m1 cpi wage trddol importpr
    compute [label] logs = "log"+%l(i)
    set %s(logs) = log(i{0})
    stats %s(logs)
end dofor i
\end{verbatim}

This is almost identical to the last example, except it uses the \%L and \%S functions to create new series LOGGNP, LOGM1, etc. which are used to hold the logged variables.

\begin{verbatim}
declare model gg
smpl 2001:5 2002:4
dofor gg = model1 model2
    forecast(model=gg)
    graph 2
    # foregnp
    # gnp 2000:1 2002:4
end dofor
\end{verbatim}

forecasts using two different models (previously set up) and graphs the forecasts.

\begin{verbatim}
declare vector values(5)
ewise values(i)=.10**i
dofor [real] scalefac = values
    ...
end dofor
\end{verbatim}

loops over values for SCALEFAC of .1, .01, .001, .0001 and .00001. Note the use of the \texttt{VECTOR[REALS]} in place of the list of reals.
ECT — Error Correction Terms

ECT is a subcommand of SYSTEM used to add error correction term(s) to a vector autoregression. These are input using equations, one per needed term. For more information, see Section 10.8 of the User’s Guide.

```
ect list of equations describing the error correction terms
```

Wizard

The VAR (Setup/Estimate) wizard on the Statistics menu includes support for defining and estimating error-correction models.

Parameters

```
equations lists equations describing the “stationary” relationships among variables in the VAR model. You can use a VECTOR of equations if you might need to vary the number. The equations can also include exogenous variables if desired.
```

Form of the Equations

The equations listed can either have one of the endogenous variables of the VAR as the dependent variable, or it can have a constructed variable, in which case the explanatory part of the equation shows the stationary relationship. For example,

```
equation ecteq y1
  # y2 y3
  system(model=cointmodel)
  variables y1 y2 y3
  lags 1 2 3 4
  ect ecteq
  end(system)
  linreg(equation=ecteq)
```

uses the first form. The cointegrating relationship in this case will be \(\beta_2 y_2 + \beta_3 y_3 - y_1\), where the \(\beta\) are coefficients from the preliminary LINREG. Note how the equation is renormalized to have a \(-1\) coefficient on the dependent variable.

Suppose that, instead, a RECTANGULAR matrix \(B\) has been estimated which lists the cointegrating vectors. The example on the following page creates a VECTOR of EQUATIONS to hold the equations. In this case, the dependent variables of the equations are ignored in forming the stationary conditions.
dec vect[equations] ecteqs(r)
do i=1,r
    equation(coeffs=%xrow(b,i)) ecteqs(i)
    # y1 y2 y3
end do i
system(model=cointmodel)
variables y1 y2 y3
lags 1 2 3 4
ect ecteqs
end(system)

The following excerpt is taken from the example program KPSW5.PRG, which reproduces Table 5 from King, Plosser, Stock and Watson, “Stochastic Trends and Economic Fluctuations”, AER, 1991, Vol. 81, pp. 819-840. You can find the program file in the “Paper Results/KPSW AER 1991” subdirectory.

Define the error correction equations

equation(coeffs=| |-betay,0.0,0.0,1.0,-betar,0.0|) mdemand
# y c in mp r dp
equation(coeffs=| |-1.0,1.0,0.0,0.0,-phi1,phi1|) cratio
# y c in mp r dp
equation(coeffs=| |-1.0,0.0,1.0,0.0,-phi2,phi2|) iratio
# y c in mp r dp

Estimate the cointegrated VAR

system(model=varmodel)
variables y c in mp r dp
lags 1 to 9
det constant
ect mdemand cratio iratio
end(system)
estimate(noprint) 1954:1 *
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 RATSData* (or *New RATSData*) operations on the *File* menu, and double-clicking on the series you want to edit, or clicking on the “new series” toolbar icon to edit a new series.

You can also edit series stored in memory rather than on a file by using the *Show Series Wizard* and double-clicking on the series you want to edit.

Both operations invoke the same series editing interface described here.

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, you must open or create a RATS format file using the *File-Open RATSData* or *File-New RATSData* operations or the **DEDIT** or **ENVIRONMENT RATSDATA=** instructions.

When you are finished, execute a **SAVE** instruction to save the changes to the RATS data file before quitting RATS or closing the file. If you quit RATS without doing a **SAVE**, the program will ask you if you want to save changes to the data file.
Creating a New Series

If you supply a new series name, 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. Enter data as described on the following pages. To save the series, just close the window and reply “Yes” when asked if you want to save the changes.

Editing a Series

Once you have opened or created a RATS format data file, you can edit or create a series by issuing a command like:

```
edit usargdps
```

RATS will display a series editing window, similar to the one shown below:

![Series Editing Window]

The dates (or entry numbers for undated data) at the left correspond to the first element in that row. For example, the first column of row three contains the value for the first quarter of 1962. Data for quarters two, three, and four for 1962 are in the other three columns of the row.

Moving Around the Table

You move around the table using the mouse or the standard cursor control keys. The cell in the upper left corner displays the value of the currently selected cell (the value 2279.2 in this example is the data for the first quarter of 1960).
Adding Numbers

To add entries to the end of the series, just click on the last entry in the table and hit the right arrow key to add one cell. To insert an entry anywhere else in the series, move to the desired point and use the insert button (issa) to insert one or more blank cells (see below for details). This shifts all remaining entries one observation later.

To add entries at the beginning of the series prior to the current starting date of the series, click on or move to the first cell in the table and use the left arrow to add one or more new cells. Note that you cannot add entries prior to entry 1 for undated data.

Editing Features

The EDIT screen provides several toolbar buttons for editing series, plotting graphs, and locating extreme values.

**Edit**

To edit an existing entry without retyping it completely, click on the desired cell and then edit the value in the cell at the top of the window. Press <Enter> to copy it to the selected cell.

(undo)

To Undo the most recent change, click on the Undo button or select Undo from the Edit menu.

(insert)

To insert a blank cell at the current cursor position, click the Insert button. This will move the data from that point one position to the right.

(remove)

To remove the current cell, click the Remove button. RATS shifts the remaining data one position to the left to fill its place.

(missing)

To replace the current cell with a missing value (represented as "NA"), select the cell and click on the Missing button.

Use these buttons to move the focus to the cell with the maximum or minimum value in the series, respectively.

(graph)

This button generates a time series graph of the series.

(freq plot)

This button generates a histogram (frequency plot) of the series

When you are done editing, just close the window. You will see a dialog box asking if you want to save the changes to the file. Even if you answer “Keep Changes”, no changes are made to the disk copy of the file until you use the SAVE instruction.
EIGEN — Eigen Decomposition of Matrices

**EIGEN** computes the eigen decomposition of a **SYMMETRIC** or **RECTANGULAR** matrix, 
or the generalized eigen decomposition for an expression of the form \( |A - \lambda B| = 0 \). 
The function `%EIGDECOMP` can also do eigen decompositions, but without some of the 
options provided by the **EIGEN** instructions.

```
eigen(options) array eigenvalues eigenvectors
```

**Parameters**

**array**
RATS computes the eigen decomposition for this array. In 
general, it can be any \( N \times N \) **RECTANGULAR** or **SYMMETRIC** 
matrix. However, if you use either the **SCALE** or **EXPLAIN** 
options, it must be a **positive definite** **SYMMETRIC** array.

**eigenvalues**
(Optional) For an \( N \times N \) array, RATS stores in this **VECTOR** the 
(real parts of the) \( N \) eigenvalues. You do not need to **DECLARE** 
or **DIMENSION** it. Use * for this parameter if you want **eigenvectors** but not **eigenvalues**. 
For a **SYMMETRIC** array, the eigenvalues are ordered from 
largest to smallest; for a **RECTANGULAR** array, they are sorted 
according to the choice for the **SORT** option. Note, by the way, 
that they are ordered by their **true** value, not absolute value. 
For example, .00001 is before −10000.0.

**eigenvectors**
(Optional) RATS stores in this \( N \times N \) **RECTANGULAR** array the 
(real parts of the) eigenvectors. The columns correspond to the 
elements in the eigenvalue vector. You do not need to **DECLARE** 
or **DIMENSION** this array. For a **SYMMETRIC** array, RATS nor-
malizes each eigenvector based upon your choice for the 
**DMATRIX** option. For a general **RECTANGULAR** array, the col-
umns will have unit length if the eigenvalue is real.

**Options**

**dmatrix=[eigenvalues]/identity**
**scale/[noscale]**

For a **SYMMETRIC** array, the eigen decomposition for \( A \) takes the form \( A = PDP' \) 
where \( D \) is a diagonal matrix. If **DMATRIX=IDENTITY** (the older option **SCALE** is a 
synonym), the matrix of eigenvectors (\( P \)) is normalized so that \( D \) is the identity 
matrix. This is only possible if **array** is **SYMMETRIC and** positive definite. For the 
default **DMATRIX=EIGENVALUES**, each column of \( P \) has unit length and \( D \) has the 
eigenvalues on the diagonal). With **DMATRIX=IDENTITY**, **EIGEN** produces as the
eigenvectors an array $F$ such that $A = FF'$. You can use this on the FACTOR option for IMPULSE and ERRORS. For instance,

```r
eigen(dmatrix=identity) %sigma * eigdec
impulse(model=rmpy, factor=eigdec, steps=24)
```

**explain** = RECTANGULAR array of fractions explained
When $array$ is a cross moment or covariance matrix, the $(i, j)$ element of the array set by the EXPLAIN option is the fraction of the variance in variable $i$ that is explained by principal component $j$. Technically, if $array$ is called $V$, EXPLAIN produces

$$ E(i, j) = \frac{\lambda_j e_j^2}{V_{ii}} $$

where $\lambda_j$ is the $j$th eigenvalue, $e_j$ is the $i$th element of eigenvector $j$ and $V_{ii}$ is the $i$th diagonal element of the $array$. This can only be used $array$ is SYMMETRIC and positive definite.

**general** = $B$ matrix for generalized eigenvalues [not used]
For computing generalized eigenvalues of the form $|A - \lambda B| = 0$, use GENERAL to supply the $B$ matrix, while the $array$ parameter supplies the $A$ matrix. $A$ and $B$ must be symmetric. See the QZ instruction for computing generalized decompositions with non-symmetric matrices.

**cvalues** = VECTOR array of complex eigenvalues
**cvectors** = RECTANGULAR array of complex eigenvectors
**sort** = real/imag/[absval]
Use **cvalues** if you want the complex eigenvalues of a RECTANGULAR array, rather than just the real part available with the **eigenvalues** parameter. Use **cvectors** if you want the complex eigenvectors. You do not need to DECLARE or DIMENSION these arrays ahead of time. They are sorted according to the **sort** option. Sorting is always from highest to lowest. **sort**=REAL sorts on the real part, **sort**=IMAG on the imaginary part and **sort**=ABSVAL on the absolute value.

**Examples**

```r
eigen %cmom eigenval
compute condition=%maxvalue(eigenval)/%minvalue(eigenval)
```

computes the condition number for the (positive-definite) SYMMETRIC array %CMOM.

```r
eigen(cvalues=cxvalues, sort=absval) a
dec vector absval(%rows(a))
ewise absval(i) = %real(%cabs(cxvalues(i)))
```

computes the absolute values of the eigenvalues of $A$, producing the VECTOR **ABSVAL**.
ENCODE — Restricted Regressions

ENCODE, combined with the UNRAVEL option on estimation instructions such as LINREG, provides one of two methods available in RATS for computing restricted regressions. For the other method, see the instructions RESTRICT and MRESTRICT and Section 5.11.

```
encode( options ) array start end list of new series
# list of variables in regression format
```

Parameters

array
A RECTANGULAR array of dimension new series x regressors. Each row of array provides the coefficients of a different linear combination of the regressors.

start end
This is the range of entries over which ENCODE will compute the transformation. If you have not set a SMPL, this defaults to the maximum range allowed by the variables being transformed.

list of series
List of series that ENCODE will construct from the linear combinations of regressors. There should be one series per row of array. Omit the list if you use the SCRATCH option.

Options

[clear]/noclear
Unless you use NOCLEAR, ENCODE erases any information from previous ENCODE instructions. If a regression requires more than one ENCODE, the ENCODEs after the first must use NOCLEAR.

scratch=VECTOR[INTEGER] of series numbers
The ENCODEd variables have no particular use other than as a group in a regression. If you use SCRATCH, ENCODE generates as many series as are needed and fills the array you name with information to reference them. Just use the name of your SCRATCH array in your regressions.

Description

ENCODE and UNRAVEL work by estimating a regression on specially constructed variables. These new variables are linear combinations of your original set of regressors, incorporating the desired restrictions. When you UNRAVEL, the regression is rewritten in terms of the original regressors.

To illustrate: suppose you want the coefficients on X4 and X5 to be equal. You can run the regression with Z=(X4+X5) as an explanatory variable instead of X4 and X5 separately. This will give you the desired regression, except that X4 and X5 don’t
show up in your output—Z does. ENCODE and UNRAVEL don’t change this procedure, they just ensure that the regression is reported using the original regressors. Basically, ENCODE computes the “Z” variable, and remembers how it was constructed. When you UNRAVEL a regression, the “Z” is replaced with equal coefficients on X4 and X5.

Example

This computes a 3rd order polynomial distributed lag on two different explanatory variables. See Section 5.9 and Example 5.6 in the User’s Guide for more information on PDLs.

```
declare rect r(4,13)
ewise r(i,j)=j**(i-1)
encode r / enc1 enc2 enc3 enc4
# shortrate{0 to 12}
encode(noclear) r / enc5 enc6 enc7 enc8
# inflation{0 to 12}
linreg(unravel) longrate
# constant enc1 enc2 enc3 enc4 enc5 enc6 enc7 enc8
```

Using SCRATCH, this would finish with

```
encode(scratch=encs) r
# shortraten{0 to 12}
encode(noclear,scratch=encp) r
# inflation{0 to 12}
linreg(unravel) longrate
# constant encs encp
```

Notes

When you UNRAVEL a regression, the result is a covariance matrix which is not full-rank. For instance, in the example above, the 27 coefficients in the final regression are actually linear combinations of the 9 coefficients estimated. You can run hypothesis tests on the restricted regressions as you would unrestricted ones. However, RATS may have to adjust the degrees of freedom of the test. For instance,

```
exclude
# shortraten{0 to 12}
```

will actually have just 4 numerator degrees of freedom, not 13. RATS will issue the warning:

```
X13. Redundant Restrictions. Using 4 Degrees, not 13
```

and automatically adjust the degrees of freedom accordingly.

See Also . . .

- RESTRICT Tests or imposes general linear restrictions.
- MRESTRICT Tests or imposes general linear restrictions using matrices.
END — Ending a Program

The instruction END has several different purposes in RATS, depending upon the context:

- **END(SYSTEM)** indicates the end of a system definition, begun with SYSTEM.
- **END** terminates procedures, functions, loops, and other compiled sections.
- **END(RESET)** clears the memory.

We describe the third purpose here. See the discussions of SYSTEM, looping instructions, and procedures elsewhere for examples of the other uses.

If RATS encounters an END instruction that does not include either the SYSTEM or RESET options, and does not terminate a loop or other compiled section, it is simply ignored. This is a change from versions prior to 7.0, where a lone END command would terminate the program.

If you wish to clear the memory, use END(RESET), as described below.

### end(option)

#### Options

**SYSTEM**

When END is used with this option, it signals the end of a SYSTEM block used to define a VAR model or other system of equations. See the SYSTEM instruction or User’s Guide Chapter 10 for details.

**RESET**

This clears all variables and other information from RATS memory. When working in interactive mode, you can also do this using the File–Clear Program menu operation.

#### See Also . . .

**HALT**

Terminates execution from inside a compiled section.
ENTER — Defining Your Own Supplementary Cards

ENTER permits you to define your own supplementary cards for a PROCEDURE. You can also use ENTER to build up a list of unknown length in a VECTOR. ENTER can be very helpful in writing high-level procedures which mimic other RATS instructions.

However, in recent versions of RATS, the new regression list functions, such as %RLADDLAG and %RLCONCAT take up part of the role of ENTER.

```
enter( options )  variables list
#  values for the variables
```

Parameters

variables list  ENTER sets the values of the variables you list using the information on the supplementary card. These can be INTEGER, REAL, COMPLEX, LABEL or STRING variables or array elements. You can use any combination of these types.

You can enter a complete array only if it is the only object in the variables list.

Options

varying/[novarying]

entries=INTEGER for number of entries

You can only use VARYING when you are entering a VECTOR of one of the basic data types. It allows you to input a list of unspecified length, such as a regression supplementary card. If you use VARYING, you can include the option ENTRIES. ENTRIES saves the number of entries processed in an INTEGER.

Supplementary Card

ENTER has a number of uses, and the placement of the supplementary card depends upon how it is being applied:

- If you use ENTER to bring in specific information when the procedure is used (a regressor list, for instance), omit the card from the procedure. When there is no supplementary card immediately after ENTER, RATS expects to see it after the instruction which called the procedure.
- If you use ENTER within the procedure itself to build up a list in a VECTOR, include the supplementary card in the procedure, right after the ENTER.

If you are reading in a complete array, ENTER takes it from a single supplementary card according to the internal arrangement of the array:

- For RECTANGULAR, by columns
- For SYMMETRIC, by rows, lower triangle only.
Examples

procedure cnstrain nconstr r
  type integer nconstr
  type vector *r
  local integer nser nper i
  local real value
  *
  do i=1,nconstr
    enter nser nper value
    compute r(i)=value-([series]nser)(nper)
  end do i
end

cnstrain will have NCONSTR supplementary cards, each with two integer values and one real value. The first integer should be a series name or number. For example:

@cnstrain 3 r
# tbill  1999:4  5.6
# tbill  2000:1  5.8
# tbill  2000:2  5.8

You can use ENTER(VARYING) along with regressor list functions to build up a list of regressors, which is particularly useful in writing procedures. The following code, taken from the VARLAGSELECT procedure, builds a VAR regressor list:

enter(varying) list
compute n=%rows(list)
compute ntotal=n*(lags+1)

* Build the regressor list, starting with the dependent variables:
dim reglist(0)
do i=1,n
  compute reglist=%rladdone(reglist,list(i))
end do i

* Now add the lagged variables.
do j=1,lags
  do i=1,n
    compute reglist=%rladdlag(reglist,list(i),j)
  end do i
end do j

See Section 16.2.3 in the User’s Guide for more on the regressor list functions.

Notes

Other instructions for initializing data are INPUT, READ and COMPUTE, which are superior to ENTER when you have known values. QUERY and MEDIT also can be used to get information from a user.
Environment

ENVIRONMENT — Error Modes and Other Options

**ENVIRONMENT** permits you to save graphs using automatically generated file names, load a library of procedures, and control other aspects of program execution.

| environment | environment strings (separated with blanks) |

### Environment Strings

The following are the available environment strings. *Note that blanks are significant.*

**gsave**=filename template for saving graphs

**gformat**=rgf/portrait/landscape/wmf/pict

Use GSAVE=template to automate the process of saving graphs to files. If your filename template includes an asterisk (*) somewhere in the filename, RATS will save each subsequent graph to a disk file using that template, replacing the * with a sequence number. For example, to save graphs as MYGRAF1.RGF, MYGRAF2.RGF, etc., do:

```plaintext
env gsave="mygraf*.rgf"
```

Without the asterisk, RATS will use the explicit filename provided, and add each graph to that file. By default, graphs are saved in RGF (RATS Graph Format). Use the GFORMAT parameter to select a different form from the choices listed above (PORTRAIT is PostScript format with a portrait orientation, and LANDSCAPE is PostScript format in landscape orientation).

**[showgraphs]/noshowgraphs**

**printgraphs/[noprintgraphs]**

RATS normally displays any graphs you generate on the screen. If you don’t need to see the graphs while RATS is executing, you can use ENV NOSHOWGRAPHS to suppress the displaying of graphs. Note: graphs will be still be saved to disk if you use OPEN PLOT or ENV GSAVE instructions in your program. Use ENV PRINTGRAPHS if you want RATS to automatically print each graph you generate as they are created. Graphs are printed even if you use NOSHOW as well.

**procedure**=procedure library file

If you have a suite of procedures that you use regularly, you can use ENV PROCEDURE as a convenient way of loading those procedures into RATS. This functions much like a SOURCE command, in that it executes the commands stored on an external file. The difference is that RATS will automatically pull in the library file whenever you do anything that clears the memory (File–Clear Program or an END(RESET) command). The library file will normally contain either a set of procedures, or a set of SOURCE commands that source in various procedures. You can accomplish the same thing by using the option /procedure=libraryfile on the command line used to start RATS.
echo/noecho

When running in batch mode, RATS normally prints out (echoes) each line it reads from the input file. If you don't want the input lines echoed, use `ENV NOECHO` at the beginning of your program. `ENV ECHO` will restart the echoing.

ratsdata=RATS Data File Name

Opens an existing RATS data file for either editing or reading. It is the equivalent to the combination of an `OPEN DATA` and a `DEDIT`. This same operation is also available as `Open RATSDData` on the `File` menu. Opening the file with `ENVIRONMENT` speeds up execution, especially if you use several `DATA` instructions for the file, because the directory is only read once.

data=data file name

copy=copy file name

plot=plot file name

These are equivalent to `OPEN DATA`, `OPEN COPY` and `OPEN PLOT` instructions.

columns=new output file width  [default is 80]

Changes the width (maximum number of characters) of the output file. RATS will adjust the formatting of output accordingly. Choose a width between 80 and 135.

traperrors/notraperrors

subscripterrors/nosubscripterrors

`TRAPERRORS` suppresses direct error messages. If you are writing a program or procedure and want to handle errors yourself, use `TRAPERRORS` and test the `%ERRCODE` variable where appropriate. Subscript errors occur when you reference an out-of-range array or series element in an expression. Suppressing subscript errors with `NOSUBSCRIPTERRORS` will speed up (somewhat) programs which make heavy use of subscripted expressions, such as `COMPUTE` instructions within loops or `EWISE` instructions. You should never do this until you are sure the program is running correctly.

Examples

```
environment gsave="basics*.eps" gformat=portrait
graph(key=upleft) 3
# rate
# ip
```

This saves a graph to disk in PostSript format with the “Portrait” orientation. The file will be called `BASICS1.EPS`.

Variables Defined

`%ERRCODE` An `INTEGER` variable. If you use `ENV TRAPERRORS`, this will be set to the appropriate error code if RATS generates an error. You can look up the error code in the text file `RATSERRS.MSG` to see the corresponding error message.
**EQUATION — Defining Linear Equations**

`EQUATION` is the general command for setting the form of a (linear) equation. For each equation, it specifies the dependent variable, the list of explanatory variables, and whether or not the equation is an identity. You can also provide the coefficients if they are known already (such as for an identity).

```plaintext
equation(options) equation depvar ARlags MAlags
# list of explanatory variables in regression format (omit if using EMPTY or LASTREG)
```

**Wizard**

You can use the `Equation/FRML` wizard on the `Statistics` menu to define equations.

**Description**

`EQUATION` takes slightly different forms for ARMA (autoregressive–moving average) equations compared with standard regression relationships.

- For standard equations, you supply only the `equation` and `depvar` parameters. The explanatory variable are listed on the supplementary card.
- For ARMA equations, you use the `ARlags` and `MAlags` parameters in addition to `equation` and `depvar`.

**Parameters**

- `equation` Name or number of the equation being defined.
- `depvar` Dependent variable of the equation.
- `ARlags` (Optional) AutoRegressive lags. If this is a single number, it means consecutive lags from 1 to `ARlags`. To skip lags, use `||list of lags||`; for instance, `||1,4||` for lags 1 and 4. You can also use a `VECTOR of INTEGERS`.
- `MAlags` (Optional) Moving Average coefficients. These work the same way as the `ARlags` parameter.

You can use the `AR` and `MA` options as an alternative to the `ARlags` and `MAlags` parameters for specifying ARMA parameters.

**Supplementary Card**

- **Standard form** For standard regression equations, `EQUATION` needs one supplementary card listing the explanatory variables in regression format. Omit it if you use the `LASTREG` or `EMPTY` options.
ARMA models

With ARMA equations, you only need a supplementary card if you use the REGRESSORS option to include extra variables not handled by the ARMA parameters and the automatic CONSTANT.

Options

identity/[noidentity]

Use IDENTITY when you are defining an identity. Some of the forecasting instructions (IMPULSE, ERRORS, SIMULATE and HISTORY) need this information.

lastreg/[nolastreg]

With LASTREG, EQUATION assigns to equation the contents of the last computed regression (LINREG only). This copies both the structure and the coefficients. Omit the supplementary card if you use this option.

countant/noconstant

For an ARMA equation, RATS includes the CONSTANT series among the explanatory variables in the equation unless you use NOCONSTANT. For non-ARMA models, NOCONSTANT is the default—if a constant term is needed, it is usually supplied using the keyword CONSTANT on the supplementary card along with the other explanatory variables.

ar=list of autoregressive lags [0]

ma=list of moving average lags [0]

You can use these as an alternative to the ARlags and MAlags parameters for specifying an ARMA equation. As with the parameters, for N consecutive lags (all lags from 1 through N), use the format AR=N or MA=N. For non-consecutive lags, use ||list of lags|| or a VECTOR of INTEGERS.

regressors/[noregressors]

You can use the option MORE with ARMA models when you want to include variables in addition to the CONSTANT and the ARMA part. Models like this are sometimes called ARMAX models (ARMA with eXtra variables). List the additional regressors (in regression format) on a supplementary card. This option was called MORE in previous versions of RATS.

frml=FRML to associate with the equation

Use the FRML option to associate a FRML with the equation. Whenever the equation is estimated, the FRML will be updated as well.

coeffs=VECTOR of coefficients [not used]

You can use the COEFFS option to put the coefficients in as you are defining the equation. This would typically be done if you are defining an identity. You can use a VECTOR or ||coefficients separated by commas|| for this.
Equation

\[ \text{variance}=\text{residual variance [not used]} \]

Residual variance for this equation. You only need to supply this if you are going
to use \textit{SIMULATE}, \textit{IMPULSE} or \textit{ERRORS}. It is usually set when the equation is
estimated.

\[ \text{empty/[noempty]} \]

Use the \texttt{EMPTY} option to create an equation which has only a dependent variable,
with no right-hand-side terms. This can be useful if you need to apply shocks to an
“exogenous” variable in an impulse response analysis.

\[ \text{entries}=\text{number of supplementary card entries to process [all]} \]

This allows you to control how many of the elements on the supplementary card
are processed. This can be useful in repetitive-analysis tasks, where you may
want to add additional entries on each trip through a loop, for example. See
Section 5.15 in the \textit{User’s Guide} for details.

**Example Using Standard Equations**

This sets up two of the equations for Klein’s model I. The second one is the identity
\( Y = C + I + G \).

```
equation wageeq privwage
  # constant prdcteq\{0 1\} trend
  equation(identity,coeffs=||1.0,1.0,1.0||) prdcteq prdcteq
  # consmptn invest govtexp
```

**ARMA Equations**

The RATS forecasting instructions cannot handle directly equations with the multi-
plicative structure permitted by the \texttt{BOXJENK} instruction. They require equations
with the simpler parametric form:

\[
\Phi(L)y_t = \alpha + \Psi(L)x_t + \Theta(L)u_t
\]

where \( \Phi, \Psi \) and \( \Theta \) are simple polynomials in the lag operator \( L \). The \texttt{DEFINE} option
on \texttt{BOXJENK} generates such equations by multiplying through by all polynomials
which appear in the denominators of the estimated form and expanding all products.

If you want to generate a series which follows a particular ARMA structure, you can
write the equation in the form above and input the equation with \texttt{EQUATION}.

**Estimating ARMAX Equations**

The instruction \texttt{BOXJENK} is the simplest way to estimate most ARIMA models.
However, if your model is a standard regression with an ARMA error term, you may
find that \texttt{BOXJENK} won’t estimate the model in the form desired. \texttt{BOXJENK} is based
upon separate “mean” and “noise” models, rather than combining them.
You can estimate simple ARMA equations (with no ARMAX components) by **INITIAL** followed by **ITERATE.** **INITIAL** does not compute coefficients for non-ARMA variables, so you need to do one of the following for ARMAX equations:

- Use **INITIAL** to set initial guesses for the ARMA parameters, while zeroing out the extra coefficients. Use **ITERATE** from there. This is most likely to be successful when most of the explanatory power is in the serial correlation model.

- For equations with MA components, but no AR components, first estimate a standard regression. Use **EQUATION** and **LINREG(EQUATION=...)** or just **LINREG(DEFINE=...)** to create the base equation. Then use:

  ```
  MODIFY equation
  VADD %MVGAVGE list of lags
  ```

  to add the MA components. This works better when the regression part of the model dominates.

### Examples With ARMA Equations

An ARIMA(2,1,1) model:  

\[(1 - \Phi_1 L - \Phi_2 L^2)(1 - L) y_t = (1 + \Theta_1 L) u_t\]

**difference** y / ydiff  
**equation(ar=2,ma=1)** ydiff eq ydiff

The equation is now written in terms of the first difference of y, which is the form in which it would generally be estimated. After it has been estimated, you can get the equivalent equation in terms of y itself using the instructions **MODIFY** and **VREPLACE**:

```
modify ydiff eq yeq
vreplace(print) ydiff with y diff 1
```

The next example defines an ARIMA(2,0,1) with AR lags 1 and 4 and MA lag 4:

\[(1 - \Phi_1 L - \Phi_2 L^4) y_t = \alpha + (1 + \Theta_1 L^4) u_t\]

and assigns the coefficient values \(\Phi_1 = 0.5, \Phi_2 = 0.4,\) and \(\Theta_1 = 0.7\) (and a 0 intercept). Equations are set up with the **CONSTANT** first, then the AR lags, then the MA lags.

**equation(coeffs=||0.0,0.5,0.4,0.7||,ar=||1,4||,ma=||4||)** yeq y

### See Also . . .

- **ASSOCIATE** Sets coefficients for an **EQUATION**
- **FRML** Defines a **FRML**—a (possibly) non-linear relationship
- **BOXJENK** Estimates ARIMA, transfer function, and intervention models.
- **MODIFY** With **VREPLACE** and **VADD**, changes the structure of an equation.
EQV  — Attaching Names to Numbered Series

EQV (short for EQuiValence) attaches a name to a numbered series. It can also assign an alternate name to an existing series. It is largely obsolete.

\[ \text{eqv} \quad \text{list of series} \quad (\text{usually listed by number}) \]
\[< \text{text card}> \quad \text{names for series} \]

Parameters

\textit{list of series}  \hspace{1em} \text{List of series to be given names}

Text Card

This is the list of the names, separated by blanks, that you want to assign to the \textit{list of series}. The names on this card must be legal variable names:

- The name must begin with a letter.
- The only characters are letters, digits, $ and _.
- The maximum length is sixteen characters.

\textbf{EQV} used to be an important instruction, and you may see it in programs written for RATS 3 or earlier. Now, if you need an instruction like this, \textbf{LABELS} will probably be the better choice. While \textbf{EQV} assigns series \textit{names}, which can be used on input and output, \textbf{LABELS} only sets output \textit{labels}, which offers several advantages:

- Series \textit{names} (done with \textbf{EQV}) must be unique: you can’t have two series called RESIDS or FORECAST, but any number of series can share a \textit{label}.
- Labels are not subject to the restrictions put on symbolic names—you can use any combination of characters (up to sixteen).
- You can set labels in a more flexible fashion. For example, you can use string expressions and \textit{LABEL} variables.

Example

\begin{verbatim}
open data g7oecd.rat
cal(q) 1956:1
allocate 9 1997:4
eqv 1 to 9
  usashort frashort gbrshort usagbond fragbond gbrgbond $
  usardiff frardiff gbrrdiff
data(format=rats) / 1 to 6
do i=1,3
  set i+6 = (i+3){0}-i{0}
end do i
\end{verbatim}

This uses ALLOCATE and EQV to produce a uniform numbering relationship between the short and long rates of three countries. The DO loop uses this relationship to construct series of differences for each of the countries.
ERRORS — Decomposition of Forecast Variance

The instruction **ERRORS** has two very different functions:

- It computes the standard error of forecasts from a dynamic (linear) model.
- It computes the decomposition of variance for a Vector Autoregression (VAR).

You can only apply **ERRORS** to linear models (sets of equations).

```
errors(options) equations steps VCV matrix
# equation stderrors newstart column (one card per equation)
```

**Wizard**

The **VAR (Forecast/Analyze)** wizard on the **Statistics** menu provides an easy, dialog-driven interface for computing variance decompositions.

**Parameters**

- **equations**
  Number of equations in the system. You can use a * if you are using the **MODEL** option.

  The **steps** and **VCV Matrix** parameters used in older versions have been replaced by the **STEPS** and **CV** options, respectively. Version 7 will still recognize the older parameters, but we recommend that you use the new options.

**Supplementary Cards (if MODEL option isn’t used)**

There is one supplementary card for each equation.

- **equation**
  The equation (name or number).

- **stderrors**
  (Optional) **ERRORS** will fill this series with the standard errors of forecast for the dependent variable of **equation**. You can also use the **STDERRORS** option to save the standard errors.

- **newstart**
  (Optional) Starting entry for **stderrors** series. If you’ve set a **SMPL**, this defaults to the start of the **SMPL**.

- **column**
  If you use the **CV** or **FACTOR** options, this is the column in that matrix which corresponds to this equation. By default, **ERRORS** assumes that you’ve listed the equations in the same order as the columns of the covariance matrix.
Errors

Options

With the exception of model, PRINT/NOPRINT, STEPS, and STDERRS, these all apply to ERRORS only when used for decomposition of variance.

**model=** model name

This is an alternative way to specify the model. This must be a model of linear equations (that is, none of the components can be a FRML). Typically, this is a model for a VAR. Because this must be a fully linear model, there are no options like CVCRIT which govern solution of more general models.

**print/noprint**

PRINT is the default if you have more than one equation, and NOPRINT is the default if you have just one. ERRORS can produce a great deal of output for a big VAR. We provide a sample of the output later in the section.

**stderrs=** VECTOR[SERIES] for standard errors

This option saves the standard errors of forecast into a VECTOR of SERIES.

**steps=** number of steps to compute

This sets the number of steps (periods) for which you wish to compute responses. If you have set a SMPL, this defaults to the number of steps implied by it. Otherwise, you must supply a value, using either the steps parameter or the STEPS option.

**cv=** SYMMETRIC covariance matrix of residuals [%sigma]

**factor=** RECTANGULAR decomposition matrix

Use CV if you want the decomposition computed using a Choleski factorization of the covariance matrix (User’s Guide, Section 10.5). If you are using the MODEL option and omit this option, RATS defaults to using the estimated covariance matrix for the MODEL (stored in %SIGMA).

As an alternative, you can use FACTOR to supply a non-standard factorization of the covariance matrix, such as the factor matrix produced by a CVMODEL instruction. (User’s Guide, Section 10.5.4). *The form of the factorization does not affect the forecast variance, only the decomposition.* This option was called DECOMP in earlier versions. DECOMP is still recognized as a synonym for FACTOR.

**impulses/[noimpulses]**

Use the IMPULSES option to print the impulse responses which go into the decomposition. An IMPULSE instruction of similar form would produce *exactly* the same output, so if you want the responses in addition to the decomposition, you need only use the ERRORS instruction with IMPULSES. If you need more flexibility, or if you need to be able to save the impulse responses, use the IMPULSE instruction instead. Note: for a big VAR, this produces a lot of numbers.
results=RECTANGULAR[SERIES] for decomposition
save=VECTOR[RECTANGULAR] array for decomposition

RESULTS saves the decomposition into an $M \times N$ array of SERIES, where $M$ is the total number of equations, and $N$ is the number of estimated equations. The series in entry $(M,N)$ of the array contains the fraction of variance of equation $M$ that is explained by variable $N$.

SAVE saves the decomposition in an $M$-vector of $L \times N$ arrays, where $L$ is the number of steps computed. If SAVE=DECOMP, DECOMP(i)(k,j) is the fraction of the variance of equation $i$ at step $k$ that is explained by variable $j$.

Both options save fractions, not percentages.

window="Title of window"

If you use the WINDOW option, a (read-only) spreadsheet window is created with the indicated title and displayed on the screen. This will display $N$ blocks of $N+1$ columns, in a format similar to the standard output.

labels=VECTOR[STRINGS] with column header labels [variable names]

Use the LABELS option if you want to supply your own labels for the top of each column in the variance decomposition output. Otherwise, RATS will use the names of the variables.

Example of Decomposition

This example computes decompositions for a system of four equations using two orderings. This is done using both the newer and older methods of coding.

system(model=canmodel)
variables cpr ml ppi gdp
lags 1 to 4
det constant
end(system)
estimate(cvout=v)
*
* The first decomposition is done in the original order:
* CPR-M1-PPI-GDP. The second is GDP-PPI-M1-CPR.
*
errors(model=canmodel,steps=24,cv=v)
errors(model=canmodel,factor=%psdfactor(v,||4,3,2,1||,steps=24)
Errors

For versions of RATS before 5.0, the orderings could only be done by altering the order of the equations on LIST. The combination of LIST and CARDS allows for a convenient way to describe a multiple equation model.

```
system 1 to 4
variables cpr m1 ppi gdp
lags 1 to 4
det constant
end(system)
estimate(cvout=v)
list ieqn = 1 2 3 4   Ordering CPR-M1-PPI-GDP
equations(cv=v,steps=24) 4
cards ieqn * * ieqn
list ieqn = 4 3 2 1   Ordering GDP-PPI-M1-CPR
equations(cv=v,steps=24) 4
cards ieqn * * ieqn
```

Example of Computing Forecast Standard Errors

```
boxjenk(ma=1,diffs=2,constant,define=cpieq) cpitran $
   1947:3 1979:12 resids
smpl 1980:1 1981:12
forecast 1
# cpi eq logcpi
errors(noprint) 1
# cpi eq fcsterr
```

estimates an ARIMA(0,2,1) model, then computes forecasts (into LOGCPI) and forecast errors (into FCSTERR). This can also be done with the UFORECAST instruction.

Technical Information

For a static model such as

(1) \[ y_i = X_i \beta + u_i \; \text{Var}(u_i) = \sigma^2 \]

the variance of the error in using \( X_{t+1} \hat{\beta} \) to forecast \( y_{t+1} \) is

(2) \[ \sigma^2 + \sigma^2 X_{t+1} (XX)^{-1} X_{t+1}' \]

(See, for instance, Pindyck and Rubinfeld (1998), p. 226). The first term is due to the equation error \( u_{t+1} \), and the second is due to sampling error in using \( \hat{\beta} \) to estimate \( \beta \). For simple projections, you can get this variance by using the STDERR option of PRJ.

The situation is much more complicated for a multiple step forecast in a dynamic model. Take the simplest possible case:

(3) \[ y_i = \alpha y_{i-1} + u_i \; \text{Var}(u_i) = \sigma^2 \]
For the one-step forecast, there is no difference between this and the static model. However, the two step forecast (forecast for \( t + 1 \) given \( t - 1 \)) that uses an estimated \( \hat{\alpha} \) is \( \hat{\alpha}^2 y_{t-1} \) while the actual value is

\[
(4) \quad y_{t+1} = \alpha^2 y_{t-1} + \alpha u_t + u_{t+1}
\]

Thus, the error is

\[
(5) \quad \alpha u_t + u_{t+1} + \left( \alpha^2 - \hat{\alpha}^2 \right) y_{t-1}
\]

Note that the effect of sampling error (the last term) depends upon the squares of the coefficients. This term becomes extremely complicated as the size of the model and the number of steps increases.

**ERRORS** ignores the sampling error term and concentrates on the others: the ones due to the effects of the innovations (\( u \)'s). Note that the two step forecast error depends not only upon the second period’s innovation, but also upon the first period’s innovation as well. More generally, in the moving average representation,

\[
(6) \quad y_t = X_t \beta + \sum_{s=0}^{\infty} \Psi_s u_{t-s} = X_t \beta + \sum_{s=0}^{\infty} \Psi_s G v_{t-s}
\]

the error in the \( K \)-step ahead forecast is:

\[
(7) \quad \sum_{s=0}^{K-1} \Psi_s u_{t-s} = \sum_{s=0}^{K-1} \Psi_s G v_{t-s}
\]

where \( GG' \) is a factorization of the covariance matrix of \( u \), and the \( v \)'s are orthogonalized innovations. The covariance matrix of the \( K \)-step ahead forecasts is

\[
(8) \quad \sum_{s=0}^{K-1} \Psi_s G G' \Psi_{s}' = \sum_{s=0}^{K-1} \Psi_s \Sigma \Psi_{s}'
\]

This does not depend upon which factorization of \( \Sigma \) is chosen. However, the decomposition of variance, which breaks this sum done into the contributions of the component of \( v \), does. See **User’s Guide** Section 10.6 for a more detailed discussion on the decomposition of variance.

**Output**

This is part of the output from an **ERRORS** instruction applied to a six variable VAR. There will be one such table for each endogenous variable.
The first column in the output is the standard error of forecast for this variable in the model. This is computed using (8). Since the computation assumes the coefficients are known, it is lower than the true uncertainty when the model has estimated coefficients. The remaining columns provide the decomposition. In each row they add up to 100%. For instance, in the sample above, 81.84% of the variance of the one-step forecast error is due to the innovation in CANRGDPS itself.

Notes

If you want to compute the true uncertainty of forecast, you need to apply the technique of Monte Carlo integration (User’s Guide, Section 13.5) to generate draws from the posterior distribution of the coefficients of the model. You then use SIMULATE to draw random shocks for the innovations during the forecast period.
ESMOOTH — Exponential Smoothing

ESMOOTH performs one of nine possible exponential smoothing techniques on a series. It can forecast, smooth or seasonally adjust a series. See Section 9.3 in the User’s Guide for more on the use of exponential smoothing for forecasting.

\[ \text{esmooth( options ) series start end} \]

Wizard

The Exponential Smoothing wizard on the Statistics menu provides dialog-driven access to most of the features of the ESMOOTH instruction.

Parameters

- **series**: Series to smooth, seasonally adjust or forecast.
- **start** and **end**: Range to smooth. If you have not set a SMPL, this defaults to the defined range of series.

Two parameters (forecasts and steps) used in older versions have been replaced by the FORECASTS and STEPS options described below. Version 7 will still recognize the older parameters, but you should use the options for new programs.

Options

- **trend=[none]/linear/exponential/select**
- **seasonal=[none]/additive/multiplicative/select**

  These jointly determine the type of model. You can choose any combination of the two. The linear trend model is the Holt–Winters two parameter model. If you choose SELECT, ESMOOTH tests all three choices for that option, and chooses the best fitting model. Used together, TREND=SELECT and SEASONAL=SELECT gives you the best fitting combination.

- **smoothed=smoothed output series**
- **resids=series of residuals (in-sample forecast errors)**
- **fitted=in-sample fitted values**
- **factors=seasonal factors series**

  These options allow you to save some of the series produced by the smoothing process. Each is defined over the period start to end. For seasonal data, the smoothed output is the seasonally adjusted series.

- **forecasts=series for forecasts**
- **steps=number of forecast steps**

  If you want to generate forecasts, use FORECASTS to provide a series name to hold the forecasts, and STEPS to set the number of forecasts steps (periods). ESMOOTH forecasts the steps periods after end.
[print]/noprint
This matters only if you use ESTIMATE or one of the SELECT choices. PRINT outputs the squared-error statistics and the final estimated coefficients.

estimate/[noestimate]
alpha=constant level smoothing parameter [.3]
gamma=trend smoothing parameter [.3]
delta=seasonal smoothing parameter [.3]
constrain/[noconstrain]
initialonly/[noinitialonly]
If you use the ESTIMATE option, ESMOOTH finds the values for $\alpha$, $\gamma$ and $\delta$ which produce the best fit with the data by minimizing the sum of squared in-sample forecast errors. By default (with NOESTIMATE), the three other options provide the values of $\alpha$, $\gamma$ and $\delta$ in Gardner’s notation (see Technical Information). In all cases, values close to zero provide the most smoothing; values close to one the least. Note: if you use SELECT for TREND or SEASONAL, ESMOOTH always estimates the parameters.

When estimating parameters, use CONSTRAN to constrain the estimated parameter values to the range $[0, 1]$. With INITIALONLY, RATS will use only a minimal number of early observations for initializing recursions for TREND and SEASONAL components. Otherwise, it will use estimates from the full sample.

span=seasonal span [CALENDAR seasonal]
You can use SPAN to change the number of periods per “year” used in the seasonal models.

Description
You choose which of the nine methods you want by choosing a trend model: no trend, linear trend (Holt–Winters) or exponential trend; and a seasonality model: no seasonal, additive seasonal, multiplicative seasonal. You can, of course, allow TREND=SELECT and/or SEASONAL=SELECT to help you make the decision. See the Technical Information below for a description of the models and formulas.

Missing Values
ESMOOTH simply smooths over missing values, assuming (in effect) that the missing datum is the forecast value for that period. This permits you to use ESMOOTH for patching gaps in a time series, provided:

- the series is reasonably smooth, so an exponential smoothing representation is adequate.
- the gaps are not too near the start of the data, since exponential smoothing relies solely on the past for the generation of the smoothed data.
Examples

```
esmooth(estimate,fore=forecast,steps=12) tbill3mo 1980:1 2006:12
```

This uses data from 1980:1 through 2006:12 to fit a non-trending, non-seasonal model with estimated coefficients, and then uses that model to produce forecasts for 2007:1 through 2007:12.

```
esmooth(alpha=1.0,gamma=0.3,trend=exponential,forecasts=ship_f, $ steps=21) shipment 1994:1 2007:3
```

forecasts 2007:4 to 2008:12 using an exponential trend, non-seasonal model, with assigned parameters.

```
esmooth(trend=select,seasonal=select,smooth=canretsax) canrett
```

smooths Canadian Retail Sales using the best fitting model of the nine possibilities. This saves the smoothed (seasonally adjusted) data in the series CANRETSAX. See Section 9.3 of the User’s Guide for a complete version of this example.

Output

The following is the output from the Canadian example above. The table at the start shows the selection of the model. For each of the nine possible models, the sum of squared errors and the Schwarz (BIC) criterion are shown. The Schwarz criterion penalizes the models which have extra parameters. The chosen model is the one which minimizes the value of Schwarz. Here, it is the linear trend with multiplicative seasonal. The data are much clearer about the choice of seasonal model than trend model, as exponential trend with multiplicative seasonal has a very similar value. The second section shows the estimated coefficients for the chosen model.

<table>
<thead>
<tr>
<th>Exponential Smoothing for Series CANRETT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model Selection</strong></td>
</tr>
<tr>
<td>TREND</td>
</tr>
<tr>
<td>None</td>
</tr>
<tr>
<td>None</td>
</tr>
<tr>
<td>None</td>
</tr>
<tr>
<td>Linear</td>
</tr>
<tr>
<td>Linear</td>
</tr>
<tr>
<td>Linear</td>
</tr>
<tr>
<td>Exponential</td>
</tr>
<tr>
<td>Exponential</td>
</tr>
<tr>
<td>Exponential</td>
</tr>
</tbody>
</table>

Model with TREND=Linear, SEASONAL=Multiplicative

| Alpha (level) | 0.215380 |
| Gamma (trend) | 0.069351 |
| Delta (seasonal) | 0.355414 |
Notes on SELECT

You should not automatically use SELECT for the options. ESMOOTH sees a series as just a set of numbers. It has no knowledge of how the series is expected to behave—you do! If it sees a general upward movement over the data set, it may very well select a trending model over a non-trending one. It can do this even for a series (such as U.S. interest rates) where you probably would not choose to include a trend yourself. With enough data, ESMOOTH will probably pick the model which is truly the best of those available, but with small data sets, your judgment becomes very important.

Technical Information

The table below lists the error-correction forms of the models used for the different combinations of SEASONAL (top) and TREND (left). We are using Gardner’s (1985) notation:

- $S_t$: smoothed level of the series
- $T_t$: trend rate
- $I_t$: seasonal index (factor)
- $e_t$: period $t$ forecast error
- $p$: seasonal span

<table>
<thead>
<tr>
<th>No Seasonal</th>
<th>Additive Seasonal</th>
<th>Multiplicative Seasonal</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>No Trend</strong></td>
<td>$S_t = S_{t-1} + \alpha e_t$</td>
<td>$S_t = S_{t-1} + \alpha e_t$</td>
</tr>
<tr>
<td></td>
<td>$I_t = I_{t-p} + \delta (1-\alpha) e_t$</td>
<td>$I_t = I_{t-p} + \delta (1-\alpha) e_t / S_t$</td>
</tr>
<tr>
<td><strong>Linear Trend</strong></td>
<td>$S_t = S_{t-1} + T_{t-1} + \alpha e_t$</td>
<td>$S_t = S_{t-1} + T_{t-1} + \alpha e_t$</td>
</tr>
<tr>
<td></td>
<td>$T_t = T_{t-1} + \alpha \gamma e_t$</td>
<td>$T_t = T_{t-1} + \alpha \gamma e_t$</td>
</tr>
<tr>
<td></td>
<td>$I_t = I_{t-p} + \delta (1-\alpha) e_t$</td>
<td>$I_t = I_{t-p} + \delta (1-\alpha) e_t / S_t$</td>
</tr>
<tr>
<td><strong>Exponential Trend</strong></td>
<td>$S_t = S_{t-1} T_{t-1} + \alpha e_t$</td>
<td>$S_t = S_{t-1} T_{t-1} + \alpha e_t$</td>
</tr>
<tr>
<td></td>
<td>$T_t = T_{t-1} + \alpha \gamma e_t / S_{t-1}$</td>
<td>$T_t = T_{t-1} + \alpha \gamma e_t / S_{t-1}$</td>
</tr>
<tr>
<td></td>
<td>$I_t = I_{t-p} + \delta (1-\alpha) e_t$</td>
<td>$I_t = I_{t-p} + \delta (1-\alpha) e_t / S_t$</td>
</tr>
</tbody>
</table>

ESMOOTH uses the simplex method to estimate parameters, minimizing the sum of $e_t^2$. It obtains initial values for $I_t$ by a regression of the data on seasonal dummy variables and for $T_t$ by a regression on a simple time trend.
Note, by the way, that while some programs limit the smoothing parameters to the range of [0,1], the smoothing model is stable for a wider range than that (for instance, [0,2] for $\alpha$), and the optimal values for many economic series are, in fact, greater than one. Thus RATS does not constrain the values to the [0,1] range by default. If you do want to impose the [0,1] constraint, use the CONSTRAINT option.

**Variables Defined**

- `%NOBS` Number of observations
- `%RSS` Sum of squared errors
- `%ESALPHA` $\alpha$, the level smoothing parameter
- `%ESGAMMA` $\gamma$, the trend smoothing parameter
- `%ESDELTA` $\delta$, the seasonal smoothing parameter

**See Also . . .**

- `UG`, Chapter 9 Univariate Forecasting
ESTIMATE — Estimating a VAR System

`ESTIMATE` computes estimates for all the equations in the most recently created `SYSTEM`. `ESTIMATE` also initializes the Kalman filter for use by the instruction `KALMAN`, which does sequential coefficient estimation.

```
estimate( options ) start end residuals coeffs
```

Wizard

The VAR (Setup/Estimate) wizard, located on the Statistics menu, provides an easy, dialog-driven interface for defining and estimating VAR models.

Parameters

```
start   end
```

Estimation period. If you have not set a `SMPL`, this defaults to the maximum range that `ESTIMATE` can use, taking into account the required lags.

Two additional parameters (`residuals` and `coeffs`) used in older versions have been replaced by the `RESIDUALS` and `COEFFS` options described below. Version 7 will still recognize these older parameters, but use the options for new programs.

Options

```
dfc=Degrees of freedom correction (User’s Guide, Section 5.15)
smpl=SMPL series or formula (Section 5.2)
spread=Residual variance series (Section 5.4)
```

These are the same as for `LINREG`, except that for each option the single value applies to all equations in the system. If you need differing spreads, for instance, you must use a set of `LINREG` instructions instead.

```
[print]/noprint
ftests/noftests
```

Use `NOPRINT` to suppress the standard regression output. For a vector autoregressive system, `FTESTS` prints a set of `$F$–tests after each estimated equation. This tests (for each regression separately) the block of included lags of the dependent variables of the system. `NOPRINT` will also suppress the `$F$–tests unless you use the `FTESTS` option explicitly.

```
sigma/[nosigma]
cvout=Symmetric covariance matrix of residuals
```

Respectively, these compute and print, or compute and save, the covariance matrix of the residuals. If you use `SIGMA`, `ESTIMATE` prints a covariance/correlation matrix of the form shown in Section 5.1 of the User’s Guide.

In previous versions, the `CVOUT` option was called `OUTSIGMA`. RATS will still recognize the old name.
**residuals=VECTOR[SERIES] for residuals**

This is the most convenient way to get the residuals from the equations of a VAR or other multiple equation system. The option RESIDUALS=RESVAR will create series RESVAR(1), ..., RESVAR(n) which will have the residuals from the n equations in the system. Note that residuals are saved internally by default.

**coeffs=RECTANGULAR for coefficients**

For a VAR, this saves the estimated coefficients in a RECTANGULAR array. Column i of this will be the coefficients from the ith equation.

**cohistory=VECTOR[SERIES] for coefficient history**

This can only be used with a single equation model. It is designed to work together with KALMAN to save the coefficient estimates as they are recomputed by the Kalman filter. There will be one series in the VECTOR[SERIES] for each coefficient. ESTIMATE will put its estimates into entry end of these series.

**model=model to estimate**

ESTIMATE normally estimates the VAR system defined by the most recently executed SYSTEM/END(SYSTEM) instructions. If you prefer, you can use the MODEL option to estimate a particular MODEL. This can be useful if you are working with multiple VAR specifications.

Note: MODEL objects do not store Bayesian prior information, so ESTIMATE with the MODEL option always does ordinary least squares estimation, ignoring any priors set by a SPECIFY command. Also, you can not use the MODEL option to estimate error-correction model systems defined using the ECT command.

**Advanced Options**

**cmom=SYMME\text{TRIC }X'X\text{ array**}

**dummy=RECTANGULAR array of dummy observations**

**ols/[nools]**

These apply only to systems set up with a prior. CMOM saves the X'X array of the regressors. DUMMY saves the matrix of dummy observations in exactly the form you need for a FULL matrix for SPECIFY. Because the array already includes all the scale factors, don’t use SCALE on SPECIFY if you use this for a FULL array. Use the OLS option if you want ESTIMATE to do ordinary least squares rather than mixed estimation (that is, if you want ESTIMATE to ignore the prior).

**Comments**

In a vector autoregression, all equations have the same explanatory variables, so ordinary least squares applied equation by equation is efficient. For systems with mixed equations, RATS still estimates by single equation OLS, so there may be some gain in using SUR instead.

You can’t do any direct hypothesis tests after an ESTIMATE. You can use RATIO to test certain cross-equation hypotheses, such as lag length restrictions or block exogeneity (User’s Guide, Section 10.4). If you want to use any restrictions on a single equation, you will have to do a LINREG to estimate that equation in isolation.
Degrees of Freedom Corrections

To compute the standard errors shown in the output, **ESTIMATE** uses the degrees of freedom correction \((T - K)\), where \(K\) is the number of parameters in the equation. This is done to match the results that would be obtained by using **LINREG** to estimate the same equation. However, the residual variance/covariance matrix computed by the **SIGMA** and **OUTSIGMA** options and saved as the \(%SIGMA\) matrix (used for constructing impulse responses, variance decompositions, etc.) uses a divisor of \(T\):

\[
\Sigma = \frac{1}{T} \sum_{i=1}^{T} u_i u_i'
\]

This is the more general form for \(\Sigma\), as using the degrees of freedom correction is not appropriate for Bayesian or near–**VAR** models.

Variables Defined by **ESTIMATE**

- \(%BETASYS\) stacked coefficient vector (REAL).
- \(%LOGDET\) log determinant of the estimate of \(\Sigma\) (REAL).
- \(%LOGL\) Normal log likelihood (REAL).
- \(%NFREE\) free coefficients, including the covariance matrix (INTEGER)
- \(%NOBS\) number of observations (INTEGER)
- \(%NREG\) number of regressors in the first equation (INTEGER)
- \(%NREGSYSTEM\) total number of regressors in the model (INTEGER)
- \(%NVAR\) number of equations (INTEGER)
- \(%SIGMA\) covariance matrix of residuals (SYMMETRIC)
- \(%VARLAGSUMS\) (for a VAR only) the \(n \times n\) matrix: \(I - \sum_{s=1}^{l} \Phi_s\), where \(\Phi_s\) is the matrix of VAR coefficients for lag \(s\). This helps compute long-run effects, like the Blanchard–Quah decomposition. See User’s Guide Section 10.5.3 and 10.5.4.
- \(%XX\) the \(X'X^{-1}\) matrix (SYMMETRIC)

Output (from **FTESTS** option)

For vector autoregressions (systems set up with **VARIABLES** and **LAGS**), a set of \(F\)-tests is printed after the regression output for each equation:

<table>
<thead>
<tr>
<th>Variable</th>
<th>F-Statistic</th>
<th>Signif</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANRGNP</td>
<td>5.0365</td>
<td>0.0004608</td>
</tr>
<tr>
<td>CANM1S</td>
<td>6.4634</td>
<td>0.0000429</td>
</tr>
<tr>
<td>CANTBILL</td>
<td>20.8706</td>
<td>0.0000000</td>
</tr>
<tr>
<td>CANCPINF</td>
<td>2.7013</td>
<td>0.0262556</td>
</tr>
<tr>
<td>CANUSXSR</td>
<td>0.3259</td>
<td>0.8959781</td>
</tr>
<tr>
<td>USARGNP</td>
<td>3.6541</td>
<td>0.0049852</td>
</tr>
</tbody>
</table>

These test the significance of the block of lags associated with each of the variables in turn. In this one, for instance, the significance level of the block of \(CANCPINF\) lags in the \(CANTBILL\) equation is .0262556.
Examples

```plaintext
system(model=canmodel)
variables usargdps canusxsr cancd90d canm1s canrgdps cancpinf
lags 1 to 4
det constant
dermend(system)

* estimate(noprint,cvout=v,residuals=resblock) * 1997:4

sets up and estimates a six-variable VAR with four lags, saving the residual covariance
matrix in V and the residuals in the VECTOR[SERIES] RESBLOCK. The estimation
range runs from the earliest possible time through 1997:4.

system(model=canmodel)
variables usargdps canusxsr cancd90d canm1s canrgdps cancpinf
lags 1 to 4
det constant
specify(tight=.15) .5
end(system)
estimate

estimates a VAR with a prior.

equation checkeq gdp
# constant  gdp{1 to 4}
system checkeq
dermend(system)
estimate(cohistory=coh,noprint) 5 9
do time=10,1999:3
   kalman(rtype=recursive,cohistory=coh) resids
dermend do
dmend

The ESTIMATE computes the regression over the first five usable time periods (allow-
ing for the four lags) and initializes the Kalman filter. The filter then generates
recursive residuals for the remainder of the sample. COH will be a vector of five series
which will contain the coefficient estimates from period 9 to 1999:3. COH(1) will be
the estimates for the CONSTANT, COH(2) for GDP{1}, etc.

See Also . . .

Chapter 10  Vector Autoregressions
KALMAN  Executes the Kalman filter
SYSTEM  Sets up a vector autoregression
SPECIFY  Sets the prior for a VAR
EQUATION  Defines a single equation
**Ewise**

**EWISE — Elementwise Operations on Matrices**

**EWISE** (short for Element–WISE) is a convenient method for setting all entries of an array. It is similar to the **SET** instruction for data series.

The %DO function can now be used to replace **EWISE** in many situations. This is especially true when you need an elementwise operation within a larger calculation, such as when defining a formula with **FRML**.

### Description

You must dimension the array before using **EWISE**. For each I or I,J combination within the bounds of the array, **EWISE** carries out the indicated calculation (you can actually use any INTEGER variables in place of I and J). The “function” can include multiple expressions, separated by commas—the values of the array will be set according to the last expression in the list. This can be useful when you need to do intermediate computations to produce the final values.

Note that you can only use **EWISE** if you want to set all the entries of an array. An instruction like

```
ewise a(i,1) = ....
```

(intending to set only column 1) is not permitted. Use a DO loop or the %DO function for this type of operation. For example:

```
do i=1,%rows(a)
    compute a(i,1) = myseries1(i)
end do i
```

or

```
compute %do(i,1,%rows(a),a(i,1) = myseries1(i))
```

### Examples with Real-Valued Arrays

**dec rect r(3,5)**

**ewise r(i,j) = j**(i-1)**

**dec rect r(3,3)**

**ewise r(i,j) = %if(i>=j,x(i-j+1),0.0)**

The second example sets the elements of R below the diagonal to 0, while setting the elements on or above the diagonal to selected values out of X.
Examples with Other Arrays

You will generally use `EWISE` with real-valued arrays, since they form the basis for most calculations. You can, however, apply it to other types of arrays. In the next example, `EWISE` inverts 10 `SYMMETRIC` arrays which are stored in a `VECTOR` of `SYMMETRIC` arrays. The trickiest part about this is making sure that you get the correct arrays dimensioned before you do the `EWISE`. Here, it isn’t necessary to dimension the ten arrays stored in `VSINV` `[VSINV(1),...,VSINV(10)]`, since each will get the dimension of the corresponding `VS`. It is `VSINV` itself that needs to be dimensioned.

```plaintext
dec vect[symm] vs(10) vsinv(10)
...
ewise vsinv(i) = inv(vs(i))
```

Notes

Anything that you can do with `EWISE`, you can also do with `DO` loops. If the `DO` loops would just loop over a `COMPUTE` instruction, use the more efficient `EWISE`. We recommend `DO` loops in several situations:

- The `EWISE` expression gets so complex (for instance, if it requires several nested `%IF` functions) that you cannot easily tell what it is doing.
- You can set up the `DO` loops to run over a reduced set of entries, where `EWISE` must set the entire array.

For instance, you might prefer to use

```plaintext
do i=1,%rows(a)
   compute a(i,1)=b(i)
   compute a(i,2)=c(i)
   compute a(i,3)=d(i)
end do i
```

rather than

```plaintext
ewise a(i,j)=%if(j==1,b(i),%if(j==2,c(i),d(i)))
```

because the `DO` loop is significantly easier to read.
Exclude

EXCLUDE — Testing Exclusion Restrictions

EXCLUDE computes the test statistic for the restriction that all of the listed variables have zero coefficients in the preceding regression. You use EXCLUDE after you estimate the regression.

You cannot use EXCLUDE with NLLS, NLSYSTEM, MAXIMIZE, FIND, DLM or CVMODEL instructions, that is, any instruction which uses NONLIN rather than a regressor list. TEST is a more general testing instruction which can be applied in these cases.

```
exclude( options )    no parameters
#    list of variables in regression format  (omit if using WHOLE option)
```

Wizard

In the Regression Tests wizard on the Statistics menu, select the Exclusion Restrictions checkbox.

Supplementary Card

The supplementary card lists the collection of variables from the previous regression which you want tested (as a block). List them in regression format.

Options

[print]/noprint

NOPRINT suppresses the printing of the test information. This is useful if you only need the \%CDSTAT or \%SIGNIF variable, and don't need to see the output.

all/[noall]

Use ALL to test whether all of the coefficients can be excluded. Omit the supplementary card if you use this. This option was called WHOLE in previous versions.

form=f/chisquared

This determines the form of the test statistic used. By default, RATS will select the appropriate form based upon the estimation technique used last. You can use FORM to manually select a distribution if you have made changes to the regression that require a different distribution, such as altering the \%XX matrix in a way which incorporates the residual variance into \%XX. See Section 5.13 in the User's Guide.

entries=number of supplementary card entries to process [all]

ENTRIES can be helpful if you are doing a nested set of exclusion restrictions on one regression, as you may be able to put a single EXCLUDE in a loop, using ENTRIES to take a different number of regressors on each pass (User's Guide, Section 5.15).
title="string for output title"

You can use the TITLE option to include information in the output to identify what is being tested.

Technical Information

The formulas used are described in Section 6.2 of the User’s Guide.

The main test statistic is usually shown as an $F$, but will be shown as a chi-squared when EXCLUDE is applied to estimates from a logit or probit model (DDV instruction), or from any instruction for which the ROBUSTERRORS option was used during estimation. You can also control the distribution yourself using the FORM option.

For $F$ tests with one degree of freedom, EXCLUDE will report a two-tailed $t$ test in addition to the $F$ test. For chi-squared tests with more than one degree of freedom, EXCLUDE will report an $F$ with an infinite number of denominator degrees of freedom (that is, the chi-squared statistic divided by the numerator degrees of freedom) in addition to the chi-square.

Examples

```
linreg gdp
# constant m1{ -4 to 8 }
exclude(title="Sims Causality Test")
# m1{-4 to -1}
exclude
# m1{ 6 to 8 }
```

This regresses GDP on 4 leads, current and 8 lags of M1. The first EXCLUDE tests the joint significance of the leads, and labels the test as “Sims Causality Test.” The second tests the joint significance of lags 6, 7 and 8.

```
linreg(robusterrors) cge
# constant fge ige
exclude
# fge ige
```

This tests the significance of the regressors with the covariance matrix of the regression corrected for possible heteroscedasticity using ROBUSTERRORS.

Variables Defined by EXCLUDE

- **%CDSTAT**: the computed test statistic (REAL).
- **%SIGNIF**: the marginal significance level (REAL).

See Also...

- **TEST**: Tests for equality with specific constants.
- **RESTRICT**: Tests more general linear restrictions.
- **MRESTRICT**: Tests more general linear restrictions (using matrices).
EXECUTE — Executing a PROCEDURE

EXECUTE invokes a PROCEDURE created (or SOURCE’d in) previously in the program. The two lines below are equivalent, with the second preferred in situations where the procedure is mimicking a RATS instruction. See Sections 1.10 and 16.2 of the User’s Guide for more information on using PROCEDURES.

```
execute procname(options) parameters
or
@procname(options) parameters
```

Parameters

- **procname**  
  Name of the procedure invoked. You must use the full name, *not* an abbreviation. Before RATS can EXECUTE a procedure, it must compile the code for the procedure itself. See “Compiling Procedures” below for details.

- **parameters**  
  List of actual parameters. These pass information (series, arrays, scalars, etc.) to matching formal parameters listed on the PROCEDURE instruction which defined the procedure.

Options

Procedure options (defined with the OPTION instruction) are selected in the same fashion as are options for any standard RATS instruction. You can abbreviate to three or more letters the option name, and the choices for a CHOICE option. For the three option types:

- **SWITCH** options use option name alone for “On” (translated as 1), and NOoption name for “Off” (translated as 0). That is, if you define an option PRINT, the user would use PRINT or NOPRINT to turn it on or off.
- **CHOICE** options use option name=keyword for choice. For instance, if you define a CHOICE option TYPE with FLAT and TENT as the choices, the user would select the type by means of TYPE=FLAT or TYPE=TENT.
- Value options use option name=variable or expression. RATS handles value options in the same fashion as procedure parameters of the same type.

Compiling Procedures

Before you can use a procedure, you must execute, or “compile”, the code that defines the procedure. If the procedure code is included in the same file as the program being executed, you just need to make sure the procedure code precedes the EXECUTE command that calls the procedure.
More commonly, the procedure code is stored on a separate file. In that case, you can:

- compile the procedure explicitly, using a **SOURCE** instruction or by including the file in your “procedure library” using the File-Preferences menu operation, the **ENVIRONMENT** instruction, or the /PROC command-line switch, or
- let RATS search for the file. Given a a procedure called “procname”, RATS will search for a file called name procname.src. It will check your “Procedure Directory” (defined via the File-Preferences), the current default directory, and the directory containing the RATS executable file.

**Notes**

If you have nested procedures, that is, if one procedure contains an **EXECUTE** for another procedure, RATS **needs to compile the code for the inner procedure first**.

When control is returned from the procedure by a **RETURN** or **END** instruction, execution of the main program continues with the instruction that follows **EXECUTE**.

**Parameters Passed by Value**

If a **PROCEDURE** expects a parameter passed by value, you can use any variable or expression which has the correct type. For example:

```plaintext
proc foo tvector treal
type vector tvector
type real treal
disp %cols(tvector) treal
end

@foo ||6.0,5.0,3.0|| %rss
```

**Parameters Passed by Address**

If a **PROCEDURE** expects a parameter passed by address, you may use a variable or array element or series element which has the expected type, or a new variable name which will be defined as a global variable of the required type. **You should only pass by address if you need to set or change the parameter passed.** See Chapter 16 of the User’s Guide for details.

```plaintext
procedure change intpar matpar
type integer *intpar
type symmetric *matpar
compute intpar=%rows(matpar)
compute matpar=inv(matpar)
end change
*
dec symm s(3,3)
ewise s(i,j)=.9**(i-j)
exec change n s
```

inverts the matrix S and sets N to 3.
Execute

Undefined Parameters/Options

If you either omit a parameter or use * in its place, the procedure will treat it as if you put a * wherever that parameter occurs. If it is used in a calculation, that calculation will be skipped. If certain parameters must be assigned values for your procedure to make sense, you should use the %DEFINED function to check this and exit if it returns 0.

```r
call quickie matpar
type symmetric *matpar
if .not.%defined(matpar) {
    disp "Syntax: @quickie matrix"
    return
}
compute matpar=inv(matpar)
end
```

Example

```r
procedure specfore series start end forecast
  type series series
  type integer start end
  type series *forecast
  *
  option integer diffs 0
  option integer sdiffs 0
  option switch const 1
  option choice trans 1 none log root

shows the parameters and options for the procedure SPECFORE. The following is an example of this being executed:

``specfore(diffs=1,noconst,trans=root) longrate $
  2007:1 2007:12 flong``

LONGRATE is the SERIES parameter, 2007:1 and 2007:12 are START and END, and the forecasts are returned to the series FLONG. DIFFS is equal to one, CONST will be zero (the effect of the use of NOCONST), TRANS will be 3—the value for the ROOT choice.

See Also...

UG, Chapter 16
General information about RATS procedures.

SOURCE
Runs a set of RATS instructions on a text file. Use SOURCE for making available a procedures distributed with RATS.

PROCEDURE
Sets up a procedure.

TYPE
Sets data types for formal parameters.

OPTION
Defines options for a procedure.

LOCAL
Declares local variables and arrays in a procedure.

%DEFINED(x)
Returns 1 if x (a parameter or option) was defined and 0 otherwise.
EXP — Taking the Anti-Log of a Series

**EXP** creates a series as the exponential (anti-log) of the entries of another series. It is the inverse of the **LOG** instruction, which takes natural logs. Although you can do this using **SET**, we have, for convenience, provided a separate instruction.

```
exp  series  start  end  newseries  newstart
```

**Parameters**

- **series** Series to transform.
- **start** Range to transform. If you have not set a **SMPL**, this defaults to the defined range of **series**.
- **newseries** Series for the result. By default, **newseries**=**series**.
- **newstart** New starting entry of transformed series. By default, **newstart**=**start**, which is usually the case.

**Examples**

```
exp  forecast
```

replaces the series FORECAST by its exponential.

```
exp  forecast / eforecast
```

sets the new series EFORECAST equal to the exponential of the series FORECAST. Here’s how you can do the same thing using **SET** and the **EXP** function:

```
set  eforecast = exp(forecast)
```

**See Also**

- **LOG** Takes the natural log of a series.
- **EXP (x)** Exp of a single real value.
- **%EXP (A)** Exp of all entries of a real array.
- **LOG (x)** Log of a single real value.
- **%LOG (A)** Log of all entries of a real array.
EXTREMUM — Extreme Values

**EXTREMUM** locates the maximum and minimum values of a *single* series. You can use the instruction **TABLE** to get the range on each in a list of series, as well as the overall maximum and minimum values for the entire group of series.

```plaintext
extremum( options )  series  start  end
```

**Wizard**

From the *Univariate Statistics* wizard on the *Statistics* menu, choose *Extreme Values*.

**Parameters**

- **series** Series you are analyzing.
- **start** Range to use. If you have not set a **SMPL**, this defaults to the defined range of **series**.

**Options**

- **[print]/noprint** You can use **NOPRINT** to suppress the output.
- **smpl=** *SMPL series or formula* (*User's Guide*, Section 5.2)
  You can supply a series or a formula that can be evaluated across entry numbers. Entries for which the series or formula is zero or “false” will be omitted, while entries that are non-zero or “true” will be included in the operation.

**Variables Defined by EXTREMUM**

- **%MAXIMUM** Maximum value (real)
- **%MINIMUM** Minimum value (real)
- **%MAXENT** Entry number associated with **%MAXIMUM** (integer)
- **%MINENT** Entry number associated with **%MINIMUM** (integer)

**Sample Output**

```plaintext
ext  gdpgrowth

Extreme Values of Series GDPGROWTH
Quarterly Data From 1948:01 To 1998:04
Minimum Value is  -3.173784136 at 1958:01  Entry 45
Maximum Value is  12.964114885 at 1950:04  Entry 16
```

**See Also ...**

The **STATISTICS**, **SSTATS**, **MVSTATS**, and **TABLE** instructions, and the functions **%MAXVALUE(X)** and **%MINVALUE(X)**.
**FFT, IFT: Fourier Transforms**

**FFT** does a Finite Fourier transform and **IFT** does an inverse Fourier transform. RATS uses a Fast Fourier transform algorithm which can transform series of any length, although it is much faster for lengths which are products of powers of two, three and five.

Note that real-valued data series need to be moved to the frequency domain before you can apply **FFT** (see **RTOC**).

<table>
<thead>
<tr>
<th>Command</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fft</strong></td>
<td>cseries, start, end, newcseries, newstart</td>
</tr>
<tr>
<td><strong>ift</strong></td>
<td>cseries, start, end, newcseries, newstart</td>
</tr>
</tbody>
</table>

**Parameters**

- **cseries**: The complex series to transform.
- **start** and **end**: The range of entries to transform. By default, the defined range of **cseries**. See the comment below.
- **newcseries**: New complex series for the result. By default, same as **cseries**.
- **newstart**: The starting entry for the result. For **FFT**, this is the entry where frequency 0 is placed. By default, same as **start**.

**Description**

The Finite Fourier transform of the series $X(t)$, $t = 1, \ldots, T$ is

$$\tilde{X}(2\pi j/T) = \sum_{t=1}^{T} X(t) \exp(-2\pi i j (t - 1)/T); j = 0, 1, \ldots, T - 1$$

The frequencies computed run from 0 to $2\pi(T - 1)/T$ by increments of $2\pi/T$. Frequency 0 goes into entry newstart.

**Comments**

You will rarely need to use any parameters other than **cseries** and **newcseries**. The most common forms of the instructions are

- **fft** cseries
- **fft** cseries / **newcseries**

the first transforming **cseries** onto itself, the second transforming it to a new series.
Examples

The following passes series x through a high-pass filter, zeroing out a band around 0 frequency. The frequencies which aren’t zeroed are from $\pi/2$ (1/4 of the number of ordinates) to $3\pi/2$.

```
freq 2 256
rtoc
#  x
#  1
fft 1
cset 1 = %z(t,1)*(t>64.and.t<=192)
ift 1
```

This example computes a covariogram by inverse transforming the periodogram of a series. To get covariances computed correctly, the number of ordinates has to be at least double the number of data points, which is why the $%\text{FREQSIZE}$ recommendation is doubled.

```
compute nords = %freqsize(1997:12)*2
freq 1 nords
rtoc
# employ
#  1
fft 1
cmult(scale=1.0/1997:12) 1 1
ift 1
```

See Also . . .

TRFUNC computes a transfer function for a filter.
FILTER — General Linear Filtering

FILTER passes a series through a linear filter. It includes a wide variety of standard filters, or you can provide the form of the filter yourself, either with a pair of supplementary cards or with an EQUATION. FILTER also includes an option for removing the trend or seasonal from a series by linear regression.

In addition the techniques provided by FILTER, there are many RATS procedures available that implement other filtering techniques. Some are included with RATS, while others are available in the Procedures and Examples section at www.estima.com.

```
filter( option )  series  start  end  newseries  newstart
#  list of lags in the filter  (Omit both supplementary cards if you use the
#  filter coefficients  EQUATION option or any TYPE other than GENERAL)
```

Wizard

The Filter/Smooth wizard on the Data menu provides dialog-driven access to many of the filtering capabilities of the FILTER instruction.

Parameters

- **series**: Series to transform
- **start**  **end**: Range of entries to transform. You must set `start` to allow for the lags in the filter and `end` to allow for the leads, if any, unless you use the TRUNCATE option.
  
  If you have not set a SMPL, this defaults to the maximum range allowed by the defined portion of `series`. This range is from `(series start) + (highest lag)` to `(series end) – (highest lead)`.

- **newseries**: Resulting series. By default, `newseries=series`.
- **newstart**: The starting entry for `newseries`. By default, `newstart=start`.

Supplementary Cards

You need two supplementary cards unless you use one of the pre-defined filter types, or the EQUATION option. Note that TYPE=CENTERED or LAGGING with input WEIGHTS will usually be simpler than using these.

1. The first card lists the lags in the filter. Represent leads as negative numbers. In filters which involve lags only, lag 0 gets a coefficient of 1.0 unless you set it explicitly.

2. The second card lists the filter coefficients. They correspond, in order, to the lags on the first card.
Options

type=[general]/flat/henderson/spencer/hp/lagging/centered
width=base span (for TYPE=FLAT,HENDERSON,SPENCER,LAGGING)
by=repetitions of simpler filter, used with TYPE=FLAT
tuning=tuning value for TYPE=HP [depends on CALENDAR]

TYPE indicates the type of filter to be used. See the Technical Information section below for details on each of these.

GENERAL allows any pattern—if used, it requires use of either the supplementary cards or the EQUATION option to indicate the form of the filter.

TYPE=FLAT is a centered flat moving average, with width given by SPAN. It can also be used for an NxM convolution of flat filters by combining the SPAN and BY options.

TYPE=HENDERSON gives a centered Henderson filter with width given by the SPAN option. This filter is used extensively by the X11 seasonal adjustment procedure.

TYPE=HP implements the Hodrick-Prescott filter. You can use the TUNING option to supply a value for the tuning parameter for the HP filter. The default values for this depend on the frequency of the current CALENDAR setting (100 for annual data, 1,600 for quarterly, and 14,400 for monthly).

TYPE=SPENCER gives the Spencer moving average filter. WIDTH must be 15 or 21 (the default is 15).

TYPE=LAGGING gives a filter on a list of lags. If you use the WEIGHTS option, it will use those values. Otherwise it will put equal weights on the lags 0 through L-1, given WIDTH=L.

TYPE=CENTERED is the same as TYPE=FLAT, except that you can supply filter weights with the WEIGHTS option. Note that for TYPE=CENTERED, you only need to supply the weights for half the window, starting with 0 lag and working out—the same lag weights will automatically be used for the leads.

equation=Equation to use to construct filter

The form of the filter comes from the equation. See the description later. Omit the supplementary cards when you use it.

remove=mean/trend/seasonal/both

Removes from series by linear regression the mean, mean and trend, seasonal (using dummies) and mean, or trend and seasonals, respectively. (This is not technically a linear filter, but serves a similar purpose to many of the other filters).
weights=VECTOR of filter weights (TYPE=LAGGING, CENTERED)
unitsum/[nounitsum] (Used with WEIGHTS)

WEIGHTS provides the filter weights used with the TYPE=LAGGING or TYPE=CENTERED options. Use UNITSUM if you want RATS to normalize the weights to sum to 1—this allows you to just supply a shape using WEIGHTS and let RATS figure out the normalization. For example: TYPE=CENTERED, WEIGHTS=| 5.0, 3.0, 2.0 |, UNITSUM gives filter coefficients: 2/15, 3/15, 5/15, 3/15, 2/15.

truncate/[notruncate]
extend=zeros/repeat/rescale/msereweight

If you set start and end manually, you normally need to allow for lags and leads. For example, if your filter includes lags, you could not start at entry one, since the lag terms would refer to non-existent entries. TRUNCATE and EXTEND offer two alternative ways to handle out-of-sample values.

With TRUNCATE, you can specify any start and end values. RATS will assume that all unavailable entries outside the data range are equal to 0 for the purposes of computing the filtered series (Missing values in the middle of the data are still treated as missing).

EXTEND=ZEROS is identical to TRUNCATE. EXTEND=REPEAT repeats the nearest end-point value. EXTEND=RESCALE uses available data, but reweights the filter coefficients near the endpoints to maintain the sum of weights. For example, a centered filter with weights (1/4, 1/2, 1/4) will be reweighted as (1/3, 2/3) at the left end point. EXTEND=MSEREWIGHT implements the minimum mean square revision technique—the same method used for the Henderson Moving Average terms in the Census X12 Seasonal Adjustment process (see Findley (1998)).

Missing Values

If any of the values required to compute an entry are missing, FILTER sets the entry in the output series to missing. Exception: data that are unavailable at the beginning or end of the filter are treated differently when using TRUNCATE or EXTEND options.

Technical Information

For TYPE=FLAT, if the span is odd (SPAN=2N+1), the filtered series is

\[ \hat{x}_t = \frac{1}{(2N+1)} \left( x_{t-N} + x_{t-N+1} + \ldots + x_{t+N-1} + x_{t+N} \right) \]

If it’s even (SPAN=2N), the output is

\[ \hat{x}_t = \frac{1}{(2N)} \left( 0.5x_{t-N} + x_{t-N+1} + \ldots + x_{t+N-1} + 0.5x_{t+N} \right) \]
This is generally used with seasonal data where the **SPAN** is the length of the seasonal, and is also known as a \(2 \times N\) filter.

If **TYPE=FLAT** and you use both the **SPAN** and **BY** options, the result is the same effect as a flat filter of width given by the **SPAN** applied to the output of a flat filter of width given by the **BY** option. This concentrates the filter a bit more towards the center than a flat filter of the same overall length.

A Henderson filter is a centered filter whose lag coefficients are selected to minimize the sum of squared third differences of the lag coefficients for the class of symmetric lag polynomials which pass third order polynomials through without change. It’s used within the X11 adjustment procedure to estimate local trends.

A Spencer filter is similar to the Henderson filter, but allows only the two widths (15 and 21).

The Hodrick-Prescott Filter (Hodrick, Prescott (1980)) computes the estimated growth component \(g\) of a series \(x\) that minimizes the sum over \(t\) of:

\[
(x_t - g_t)^2 + \lambda (g_t - 2g_{t-1} + g_{t-2})^2
\]

The value of \(\lambda\) is set by the **TUNING** option. This is solved internally by Kalman smoothing. See Example 12.3 for additional technical details.

**Examples**

```r
data 1955:1 2007:6 ip
filter(type=flat,span=12) ip 1955:7 2006:12 trdcycle
```

This sets \(TRDCYCLE_t = (\frac{.5 \times IP_{t-6} + IP_{t-5} + \ldots + IP_{t+5} + .5 \times IP_{t+6}}{12})\). Note that the range is set to run from (1955:1)+6 to (2007:6)–6. This use of an even span means that each calendar month will get the same weight in the average, and thus a seasonal will tend to be flattened.

```r
filter(type=hp,tuning=1600) y / yhpt
set detrend = y-yhpt
```

makes \texttt{DETREND} equal to the difference between \texttt{Y} and its Hodrick-Prescott trend estimate.

```r
filter(type=centered,width=5,extend=repeat) strikes / sstrikes
graph(footer="Simple 5-term Moving Average of Strike Data",$美貌 Overlay=dots,ovsame) 2
# sstrikes
# strikes
```

does a 5-term centered flat moving average, with the input series extended out of sample by repeating the end values.
filter(type=lagging,weights=||4,3,2,1||,unitsum) data / wma
computes \( WMA_t = 0.4 \times DATA_t + 0.3 \times DATA_{t-1} + 0.2 \times DATA_{t-2} + 0.1 \times DATA_{t-3} \)

filter(type=henderson,width=23,extend=mse) data / tc
do a 23-term Henderson MA with a minimum mean square error revision handling of the end points.

filter(remove=seasonal) van / deseas
filter(type=centered,width=5) deseas / trend
removes the seasonal from VAN by regression on seasonal dummies, then smooths with a five term centered moving average to get a simple estimate of the trend-cycle.

Using the EQUATION Option
The equation must have one of the following forms:

1. \( y_t = A(L)y_t \) (univariate autoregression)
2. \( y_t = B(L)x_t \) (univariate distributed lag)

The only other variable allowed in the equation is the CONSTANT. FILTER uses \( 1 - A(L) \) for type 1 and \( B(L) \) for type 2, for instance,

Equation \( y_t = 1.3y_{t-1} - 0.6y_{t-2} \) is converted into the
Filter \( 1 - 1.3L + 0.6L^2 \)

The variables in the equation do not have to be related to the series you are filtering. For instance, the equation in the next example is an autoregression on residuals, after which FILTER is applied to the dependent variable and regressors.

Example
This filters the series loggpop and logpg with a linear filter derived from a second order AR:

linreg(define=ar2) resids
# resids{1 2}
filter(equation=ar2) loggpop / fgpop
filter(equation=ar2) logpg / fpg
Using Vectors

For long and complex filters, it is probably simplest to put the filter coefficients into a vector. The supplementary cards on FILTER will accept a vector of integers for the list of lags and vector of real numbers. For instance, the following generates a fifty lag expansion of a \((1-L)^7\) fractional difference filter. (DIFFERENCE has an option for a specific form of fractional differencing).

```
dec vect[int] lags(50)
dec vect coeffs(50)
ewise lags(i)=i
ewise coeffs(i)=binomial(.7,i)*(-1)**i
filter(truncate) series / fracdifs
# lags
# coeffs
```

See Also . . .

DIFFERENCE Performs simple and seasonal differencing operations.
FIND — General Optimization

FIND is the “catch-all” optimization instruction. You can use it to find the maximum, minimum, or root (zero) of almost anything that can be computed with a RATS instruction or set of instructions. The only restriction is that the parameters must be continuous and real-valued. Before you can use it, you must:

- Set the list of free parameters using NONLIN.
- Set initial values for the parameters (usually with COMPUTE or INPUT).

```
find  maximum/minimum/root  expression
    instruction or block of instructions used in computing expression
end find
```

Parameters

- **max/min/root** Choose which of the operations you wish to do.
- **expression** This is the (real-valued) expression that FIND is optimizing. This will usually be a variable set within the block of instructions. You must declare any variables in this expression which are set within the instruction block.

Options

- **parmset=PARMSET to estimate [default internal]**
  This tells which PARMSET is to be estimated by the FIND. If you don’t provide a PARMSET, RATS uses the last one created by NONLIN (User’s Guide, Section 7.6)

- **[print]/noprint**
  **vcv/[novcv]**
  These are the same as for other estimation instructions (see LINREG for details). VCV only works with METHOD=BFGS and only if you use the STDERRS option.

- **method=[simplex]/genetic/bfgs**
- **pmethod=simplex/genetic/bfgs**
- **piter=number of PMETHOD iterations to perform [none]**
  SIMPLEX is the simplex algorithm, GENETIC is the genetic search method and BFGS is Broyden, Fletcher, Goldfarb and Shanno. These are described in Sections 7.2 and 7.3 in the User’s Guide. SIMPLEX and GENETIC require only a continuous function while BFGS assumes the function is twice continuously differentiable. Only BFGS is capable of producing standard errors for the estimates, and it is also the only method capable of carrying out constrained optimization (see Section 7.4).
Use PMETHOD and PITERS if you want to use a preliminary estimation method to refine your initial parameter values before switching to one of the other estimation methods. For example, to do 20 simplex iterations before switching to BFGS, you use the options PMETHOD=SIMPLEX, PITERS=20, and METHOD=BFGS.

\textbf{hessian=initial guess for inverse Hessian} (METHOD=BFGS only)

You can use this with METHOD=BFGS. Without it, FIND will start with a diagonal matrix whose elements are the reciprocals of the (numerically computed) second derivatives of the function. See Section 7.2 in the User’s Guide.

\textbf{stderrs/[nostderrs]}

With this you can decide whether or not to show the computed standard errors of the coefficients. This is only an option if you use METHOD=BFGS. The other two methods can’t compute standard errors at all. If you use METHOD=BFGS, you can use STDERRS to make the output show standard errors, \( t \)-statistics and significance levels, just like other estimation instructions. However, if the function that you are maximizing isn’t a likelihood or quasi-likelihood function, the numbers reported (computed as described in Section 7.5 of the User’s Guide) are unlikely to be interpretable as standard errors.

\textbf{iterations=iteration limit [100]}

\textbf{subiterations=subiteration limit [30]}

\textbf{cvcrit=convergence limit [.00001]}

\textbf{trace/[notrace]}

\textbf{ITERATIONS} sets the maximum number of iterations, \textbf{SUBITERS} sets the maximum number of subiterations, \textbf{CVCRIT} the convergence criterion. \textbf{TRACE} prints the intermediate results. For \textbf{METHOD=SIMPLEX}, an “iteration” is actually defined as \( K \) vertex changes, where \( K \) is the number of free parameters. This makes the number of calculations per “iteration” similar to the other methods.

\textbf{title=“description of optimization being done”}

FIND will, by default, label the output as “FIND optimization.” You can use the \textbf{TITLE} option to replace this with something more descriptive.

**Statement Block**

\textbf{FIND} is actually very similar to the instruction \textbf{LOOP} (User’s Guide, Section 16.1). A function evaluation will execute all instructions between the \textbf{FIND} and the \textbf{END FIND}. A \textbf{BREAK} instruction within the statement block will cause \textbf{FIND} to abort estimation.
Technical Information

**FIND ROOT** actually minimizes the absolute value of the expression. This allows the optimization algorithms to be used, but it can be thrown off if initial values are near a local minimum of the function. **FIND ROOT** is provided as a convenience, but is not a dedicated “root finder.” Make sure you check the function value (either in the output or in the %FUNCVAL variable) to see if it is near zero before using the results.

Notes

**FIND** can be very slow. You have to remember that the instructions it controls will have to be executed once for each function evaluation, and it may take several hundred function evaluations to reach convergence even with just four or five free parameters.

Examples

This is a trivial example which shows the basic steps required to use **FIND**. It finds the minimum of \( x^2 + x + 3 \)

```
nonlin x
compute x=0.5
find minimum x**2+x+3
end find
```

The **NONLIN** sets \( x \) as the only parameter. Notice that you don’t have to bracket the minimum: a single point is enough. This example is so trivial that we don’t need to do any computations in the statement block. You still need the **END FIND**, however.

The next example is more complex example. It determines an optimal scale factor for a logistic in creating draws for a truncated Normal. This finds the minimum of a maximum, and thus requires one **FIND** instruction inside another, and two **PARMSET**’s—one being estimated by the outer **FIND**, one by the inner one.

The inner **FIND** has a **NOPRINT** option. Without this, you’ll get an output from the estimation for each trial value of the outer **FIND**. In this case, we are using the slower **METHOD=GENETIC** because of a (justified) fear that the inner **FIND** will end up at a local and not global maximum for some of the trial values for **SFAC**.

Note that the inner **FIND** has an **IF** which sets the value to **%NA** if the scale factor goes out of range. This method allows you to steer the optimization away from restricted regions.
Find

\begin{verbatim}
nonlin(parmset=xset) xm
nonlin(parmset=sset) sfac
*
compute xm=1.0,sfac=1.0
*
compute value=0.0
find(parmset=sset,trace) min value
  find(parmset=xset,method=genetic,noprint) max value
    if sfac<=0.0
      compute value=%na
    else
      compute value = -.5*xm**2+xm/sfac + log(sfac) + 2*log(1+exp(-xm/sfac))
    end find
end find
\end{verbatim}

Output

This is a typical output from a FIND instruction. Because there is no connection to "data" in the basic instruction, there aren't any goodness of fit statistics or the like. Standard errors, t-statistics and significance levels are included only if you use METHOD=BFGS and the STDERRS option and you should do that only if you are quite sure that the use of the inverse Hessian as an estimate of the covariance matrix is proper (see Section 7.5 in the User’s Guide).

FIND Optimization - Estimation by Genetic
Convergence in 43 Iterations. Final criterion was 0.0000061 < 0.0000100
Function Value 0.99668260

<table>
<thead>
<tr>
<th>Variable</th>
<th>Coeff</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. XM</td>
<td>-0.999996023</td>
</tr>
</tbody>
</table>

Variables Defined

\begin{itemize}
  \item [%FUNCVAL] Final value of expression (REAL)
  \item [%BETA] The estimated parameter values (VECTOR)
  \item [%CONVERGED] = 1 or 0. Set to 1 if the process converged, 0 if not.
  \item [%CVCGRIT] final convergence criterion. This will be equal to zero if the subiterations limit was reached on the last iteration (REAL)
  \item [%LAGRANGE] VECTOR of Lagrange multipliers if estimating with constraints and METHOD=BFGS.
\end{itemize}
**FIXED — Setting Values of Arrays in PROCS and FUNCTIONS**

*FIXED* is used in *PROCEDURES* and *FUNCTIONS* to create and set the values of arrays whose values will remain constant within the procedure or function.

```
fixed  data type  array name(dimensions)
values for the array (separated by spaces)
```

**Parameters**
- `data type`: can indicate an array or array of arrays of integer, real, complex, label or string values.
- `name(dim)`: the name of the array you want to create, followed by the dimensions of the array in parentheses. Only a single array can be initialized on a *FIXED* instruction. If you have more than one array to create, use multiple *FIXED* instructions.

**Text Card**
- `values for the array`: This is a collection of numbers or strings to fill the elements of the array. They can be separated by blanks, tabs or commas, and can cover more than one line. The array is filled by rows. If a row has been filled, and there are more values on the input line, they will be used for the next row.

  If you use an array of arrays, such as:
  
  ```
  fixed vect[rect] cval(5)(3,3)
  ```
  
  `CVAL(1)` will be filled first, followed by `CVAL(2)`, etc.

**Description**

*FIXED* can only be used inside a *PROCEDURE* or *FUNCTION*. It is used to set the values for an array whose size and values are fixed; for instance, a lookup table for critical values. The arrays declared using *FIXED* are considered to be local to the procedure.

Note that no expressions can be used either in the dimensions or the values. If you need something more general, use the combination of *LOCAL*, *DIMENSION* and *ENTER*. 
Examples

```plaintext
fixed vect[int] dfssize(6)
  25 50 100 250 500 9999
fixed vect dfsigval(8)
  .01 .025 .05 .10 .90 .95 .975 .99
fixed vect[rect] dftcval(3)(6,8)
  -2.66 -2.26 -1.95 -1.60 0.92 1.33 1.70 2.16
  -2.62 -2.25 -1.95 -1.61 0.91 1.31 1.66 2.08
  -2.60 -2.24 -1.95 -1.61 0.90 1.29 1.64 2.03
  -2.58 -2.23 -1.95 -1.62 0.89 1.29 1.63 2.01
  -2.58 -2.23 -1.95 -1.62 0.89 1.28 1.62 2.00
  -2.58 -2.23 -1.95 -1.62 0.89 1.28 1.62 2.00
```

(plus two more 6x8 tables)

This provides critical values for the Dickey-Fuller $t$-test. There are three cases, each having a separate 6x8 table. Each row is for a different sample size, the values of which are given by the `DFSSIZE` array, while the columns represent the desired significance level, shown in the `DFSIGVAL`. One way to use this would be as follows: suppose `DFCASE` is a variable choosing which of the three cases applies, and suppose also that the row to be used has been chosen and is in `ISIZE`. The following brackets the significance level of the test statistic. (Anything smaller than the .01 level will show as between 0 and .01).

```plaintext
compute lower=0.0, upper=1.0
do i=1,8
    if teststat<dftcval(dfcase)(isize,i) {
        compute upper=dfsigval(i)
        break
    }
else
    compute lower=dfsigval(i)
end do i
disp "Dickey-Fuller Test Statistic" teststat
disp "Marginal Significance Level is between" lower "and" upper
```
FMATRIX — Matrix from a Filter

**FMATRIX** creates matrices which, as linear operators, are like a **FILTER** or **DIFFERENCE** instruction. Such matrices occur frequently in advanced regression techniques with time series data. **FMATRIX** also is the simplest way to generate banded matrices. See, for instance, the second example on the next page.

**FMATRIX** forms the lag polynomial for the filter and sets the coefficients for the polynomial into the rows of the new matrix.

```
fmatrix( options )  array   startrow   startcolumn   endrow
# list of lags in filter   (omit if using DIFF, SDIFF or EQUATION options)
# filter coefficients      (omit if using DIFF, SDIFF or EQUATION options)
```

**Parameters**

- **array**
  - RECTANGULAR array to set. *You must dimension this array at some point before the FMATRIX instruction.*
- **startrow**
  - The starting row for the process. By default, `startrow`=1.
- **startcolumn**
  - The column in `startrow` for the coefficient on the zero lag of the filter. By default, `startcolumn`=1.
- **endrow**
  - The ending row for the process. By default, the last row in `array`.

**Description**

You set the form of the filter using the options described on the next page.

**FMATRIX** puts the zero lag of the filter into `startcolumn` of `startrow`. The coefficient for lag `m` is `m` columns to the right of `startcolumn`. Leads (negative lags) are to the left. In the next row, the coefficients are laid out beginning one column further to the right. This continues until `endrow`.

**FMATRIX** skips filter coefficients if they would end up outside the bounds of the array. Thus, you have to be careful about your choice of dimensions so the filter doesn’t get truncated unintentionally.

**Options**

**DIFFERENCES, SDIFFERENCES** and **SPAN** (shown on the next page) set differencing filters as in the instruction **DIFFERENCE**. The **EQUATION** option is identical to that for the instruction **FILTER**. See the discussion there. The third way to specify the filter is to use a pair of supplementary cards, again as described with the instruction **FILTER**.
differences=number of differences
sdifferences=number of seasonal differences
span=seasonal span [CALENDAR seasonal]

equation=equation name or number

scale=scale factor for filter coefficients [1.0]
This multiplies all the filter coefficients by scale factor.

[zeros]/nozeros
The ZEROS (the default) option sets to zero all entries in array, other than those set as filter coefficients. If you use NOZEROS, these other entries keep the values that they had before FMATRIX. If you need several FMATRIX instructions to set up the array, use NOZEROS on FMATRIX instructions after the first.

Examples

dec rect amatrix(4,7)
fmatrix(diffs=3) amat

creates the matrix

```
1.0  -3.0   3.0  -1.0   0.0   0.0   0.0
0.0   1.0  -3.0   3.0  -1.0   0.0   0.0
0.0   0.0   1.0  -3.0   3.0  -1.0   0.0
0.0   0.0   0.0   1.0  -3.0   3.0  -1.0
```

The 1, -3, 3, -1 entries are the coefficients of a third order differencing operator. With each new row, these coefficients move over one column.

dec rect s(5,5)
fmatrix s 1 1
  # -1   0  1
  # .25 .5 .25

The 1 1 parameters mean that the lag 0 coefficient goes in the first column in the first row:

```
.50  .25  .00  .00  .00
.25  .50  .25  .00  .00
.00  .25  .50  .25  .00
.00  .00  .25  .50  .25
.00  .00  .00  .25  .50
```

Note how the filter coefficients are truncated in the first and last rows.

See Also . . .

DIFFERENCE  Differences or seasonally differences a series.
FILTER      General linear filter.
FOLD: Folding a Spectrum

**FOLD** computes an implied spectral density function for a lower sampling rate. For instance, it takes a spectrum based on monthly data and computes the spectrum you would observe if data were only available quarterly.

```
fold( option ) cseries start end newcseries newstart
```

**Parameters**

- **cseries**: Complex series to transform.
- **start end**: Range of entries to process. By default, the defined range of **cseries**.
- **newcseries**: Series for the result. For **FOLD**, this must be different from **cseries**.
- **newstart**: Starting entry for the result. By default, same as **start**.

**Option**

```
factor= reduction factor [1]
```

- **reduction factor** is the ratio of sampling rates: 3 (=12/4) for monthly to quarterly. **FOLD** transforms \(N\) frequencies to \(N/reduction\ factor\) frequencies.

**Description**

**FOLD** sets each frequency in **newcseries** equal to the sum of all the frequencies for **cseries** aliased with it at the lower sampling rate.

**Example**

```
*  nords is the number of ordinates,
*  nobs is the number of actual data points
*
compute nords=384
freq  5 nords
rtoc  1966:1 1991:11
#  ipx
#  1
compute nobs=%nobs
fft  1
cmult(scale=1./(2*pi*nobs))  1 1
fold(factor=3)  1 1 nords 2
```

Series 2 has 384/3=128 entries, which are entries 1+129+257, 2+130+258,... of series 1.
**FORECAST — Dynamic and Static Forecasts**

**FORECAST** is the workhorse forecasting instruction. It normally computes dynamic forecasts—that is, it feeds forward forecasts of the early periods for use in later periods—but it can also do static forecasting. For forecasting a single equation, you may find **UFORECAST** more convenient. See Chapter 8 of the *User's Guide* for more.

**FORECAST** assumes zero shocks across the forecast period. You can override this with one of the options **INPUT**, **SHOCKS**, **MATRIX** or **PATHS**.

You need to build your model before doing **FORECAST**. You can either supply a list of equations on supplementary cards, or use the **MODEL** option to forecast a model defined using the **MODEL** option on **SYSTEM** or by a **GROUP** instruction.

```
forecast( options ) equations
# equation forecasts newstart one for each equation
# first period shocks (only with INPUT option)
# list of path series (only with PATHS option)
```

**Wizards**

The **VAR (Forecast/Analyze)** wizard on the *Statistics* menu can actually forecast any **MODEL**, and thus provides access to most of the functionality of **FORECAST**. If you only need to forecast a single equation, you can use the *Single-Equation Forecasts* wizard.

**Parameters**

```
equations
```

Number of equations in the system. Omit this when using the **MODEL** option.

Two additional parameters (**steps** and **start**) used in older versions have been replaced by the **STEPS**, **FROM**, and **TO** options described below. Version 7 will still recognize these older parameters.

**Supplementary Cards**

There is one supplementary card for each equation, unless you use the **MODEL** option. The other two types of supplementary cards shown above are only used with special options for adding shocks: **INPUT** and **PATHS**.

```
equation
forecasts (Optional) The series for the computed forecasts of the dependent variable of **equation**. If you find it is convenient, **forecasts** may be the same as the dependent variable.
newstart (Optional) The forecasts will be saved starting in entry **newstart** of the **forecasts** series. By default, this will be the same as the starting period of the forecast range.
```
Options

model=model name
This is an alternative way to specify the system of equations and is the only way to forecast with a set of FRMLs. MODELS are usually created by GROUP or SYSTEM.

from=starting period of the forecast interval
to=ending period of the forecast interval
steps=number of forecast steps to compute
These determine the periods for which forecasts will be computed. If you have set a SMPL, these default to forecast over that range. Otherwise you can use:

- FROM and TO to set the starting and ending periods for the forecasts, or
- FROM and STEPS to set the starting date and number of steps (periods)

If you use STEPS, but not FROM, the default for FROM will be one entry past the end of the estimation range that was used last.

print/[noprint]
window="Title of window"
Use PRINT if you want RATS to display the forecasts in the output window or file; this is not done automatically. Use the WINDOW option to have RATS display the results in a (read-only) spreadsheet window with the indicated title. This will show the forecasts in columns below the name of the dependent variable. Note that you can use the File–Export... operation to export data directly from such a window.

results=VECTOR[SERIES] for result series
stderrs=VECTOR[SERIES] for standard errors
These allow you to save the forecasts and standard errors of forecast into VECTORS of SERIES. For instance, RESULTS=FCASTS will make FCASTS(i) the series of forecasts for the ith equation in the model. See the ERRORS instruction for details on how the standard errors of forecast are computed.

static/[nostatic]
errors=VECT[SERIES] of one-step forecast errors
Use STATIC if you want static forecasts rather than dynamic forecasts—that is, if you want to use actual values rather than forecasted values for lagged dependent variable terms at later forecast horizons.

When doing static forecasts, you can use the ERRORS option to save the forecast errors into a VECTOR of SERIES.

skipsave=number of steps to skip when saving forecasts [0]
This option makes it easy to save forecasts only at the desired horizons when doing multi-step-ahead forecasts in a rolling regression loop. See “Using the SKIPSSAVE Option” later in this section.
Forecast

**iders** = iteration limit for solution algorithm [50]
**cvcrit** = convergence criterion [.00001]
**damp** = damping factor (1.0 = no damping) [1.0]

These apply if you are simulating a model which contains FRMLs and not just EQUATIONS, and thus requires an iterative solution (*User’s Guide*, Section 11.3)

**input/[noinput]**
**shocks** = VECTOR of first period shocks
**matrix** = RECTANGULAR array of shock paths
**paths/[nopaths]**

These options (which are mutually exclusive) add shocks to the equations. **INPUT** and **SHOCKS** options do this only at the first period, **MATRIX** and **PATHS** over the whole forecast horizon. You can use them either with linear or general models. They have a number of uses, from add factoring to bootstrapping. They are described in greater detail later in this section.

**Description**

**FORECAST** solves the model for each time period requested. If there are any lagged dependent variables, **FORECAST** will use the actual data if the lag reaches back before the first forecast period, and will use the forecasted values if not.

For instance, in forecasting the range \( T+1 \) to \( T+n \), a one period lag will come from the actual data at period \( T \) when forecasting \( T+1 \), but will come from the period \( T+1 \) forecast when forecasting \( T+2 \). Note, however, that this only applies for variables which are dependent variables of one of the equations of the system. Any exogenous variables have to be treated as described below.

If you have an equation with a moving average part (for instance, one estimated with **BOXJENK** with MA or SMA terms), the lagged residuals come from the series of residuals saved when estimating the equation.

If you have a **MODEL** which includes FRML’s, **RATS** uses the Gauss–Seidel solution technique described in Section 11.3 of the *User’s Guide*.

If you get forecasts which show as NA (missing values), the usual cause is that an exogenous variable is not defined into your forecasting period. Other possibilities are the absence of a lagged dependent variable. For instance, you try to forecast beginning at 2007:1, but one of your series has no data for 2006:4 and you need a lag of it.

**Exogenous Variables**

Exogenous variables require some care, as you need values for them throughout the forecast period. If a variable is not the dependent variable of one of the equations, **RATS** takes its values over the forecast horizon from its data series. If you are forecasting out-of-sample, you have two basic ways to handle exogenous variables:
• Close the model by adding to it equations or formulas, such as univariate autoregressions, that forecast the exogenous variables.
• Set up time paths for the variables (prior to forecasting) with `SET` or `DATA` or some other instruction that will directly provide the values.

Using All Available Data

`FORECAST` in its simplest form sometimes cannot take advantage of all available information in applied forecasting situations. Data for different series may become available at slightly different times. As it is described above, `FORECAST` cannot make forecasts when only part of the data are available for a time period. See Section 10.14 in the User’s Guide for information on methods of incorporating all known data into the forecasts.

Examples

```r
system(model=canmodel)
variables canrgdps canm1s cancd90d cancpinf canusxsr usargdps
lags 1 to 4
det constant
specify(tightness=.15,type=symmetric) 0.50
end(system)
estimate
forecast(model=canmodel,results=forecasts,from=2006:3,steps=10)
```

forecasts 2006:3 to 2008:4 using a six-variable VAR. The forecasts are in series `FORECASTS(1)` (for `CANRGDPS`) to `FORECASTS(6)` (for `USARGDPS`).

Below, equation `UNEMPEQ` is a forecasting equation for the series `UNEMP`. The `FORECAST` instruction computes and prints forecasts for 24 months beginning with 2007:7. The `GRAPH` graphs both the forecasts and the last year of actual data.

```r
smpl 2007:7 2009:6
forecast(print) 1
# unempeq foreun
graph 2
# unemp 2006:7 2007:6
# foreun
```

Below, equation `M1EQ` has the log of `M1` (`LOGM1`) as its dependent variable, `RATEEQ` is an equation for an interest rate, and `PRICEQ` is for `LOGPRICE`. This computes forecasts over 2003:2 to 2007:4 (fifteen quarters) and appends them to the original data series. The `SET` instructions take the anti-logs of `LOGM1` and `LOGPRICE`, and the `PRINT` instruction prints the forecasts along with the last five quarters of data.
forecast\( (\text{from}=2003:2, \text{steps}=15) \) 3

\begin{verbatim}
# mleq logm1
# rateeq rate
# priceq logprice
smpl 2003:2 2007:4
set m1 = exp(logm1)
set price = exp(logprice)
print 2002:1 2007:4 m1 rate price
\end{verbatim}

You can do the same operation using a GROUPed system by incorporating definitional identities for the log relationships:

\begin{verbatim}
frml(\text{identity}) m1iden m1 = exp(logm1)
frml(\text{identity}) priceiden price = exp(logprice)
group smallmod mleq rateeq>>rate priceeq $
    m1iden>>m1 priceiden>>price$
forecast(\text{model}=\text{smallmod}, \text{from}=2003:2, \text{to}=2007:4, \text{print})
\end{verbatim}

### Options for Adding Shocks

- **matrix**=RECTANGULAR array of shock paths
- **paths**/[nopaths]

You can use MATRIX or PATHS to input paths of shocks over more than one period. With MATRIX, the columns of the RECTANGULAR array provide the paths for the equations. These correspond to equations in the supplementary card order, or the order in which the model was formed. The number of rows does not have to be equal to steps. Shocks will be zero for steps beyond those supplied by the array.

With \textsc{paths}, a supplementary card lists the series which provide the paths of the shocks. These series must be defined for \textit{steps} entries beginning with the entry given by the \textit{start} parameter. Put a * on the supplementary card for an equation whose shocks you want to be zero.

- **input**/[noinput]
- **shocks**=VECTOR for first period shocks

You can use either of these options to input general first period shocks. With \textsc{input}, a supplementary card provides the values; with \textsc{shocks}, the indicated \textsc{vector} provides the shocks.

### Examples with Added Shocks

\begin{verbatim}
group macro cons>=>f_cons investeq=>f_invest $
    rateeq=>f_rate gnpid=>f_gnp
set m = m{1}*1.015
forecast(input, model=macro, from=2002:3, to=2005:2)
\end{verbatim}

forecasts a small macro model, assuming 6\% (compound) growth in the (exogenous) money stock and “add factoring” the interest rate equation by .5.
The code above generates X1SIMUL and X2SIMUL as bootstrapped realizations of a model. BOOT combined with the two SET instructions creates a path of residuals selected randomly from the estimation period.

Using the SKIPSAVE Option

When doing multi-step-ahead forecasts in a rolling regression application, you often are interested only in the last forecast step computed on each trip through the loop. For example, suppose you are doing a rolling regression with quarterly data and are only interested in the forecasts for one year beyond the end of the estimation sample. You can’t just do:

```plaintext
do end=1998:1,2007:1
    linreg(define=yequation) y 1980:1 end
    # constant y{1} x1 x2
    forecast(from=end+1,steps=4) 1
    # yeq yfore
end
```

because each new set of four forecasts will overwrite the second, third, and fourth forecasts from the previous trip through the loop. You could add instructions to copy the fourth-quarter forecasts into a separate series, but the easiest solution is to use the SKIPSAVE option:

```plaintext
do end=1998:1,2007:1
    linreg(define=yequation) y 1980:1 end
    # constant y{1} x1 x2
    forecast(skip=3,from=end+1,steps=4) 1
    # yeq yfore
end
```

RATS still computes (and stores internally) all four forecast steps, but only the fourth step is saved into the YFORE series. When the loop is complete, YFORE will contain the one-year-ahead forecasts from each regression.
FREQUENCY: Initializing Frequency Domain Analysis

FREQUENCY creates complex series and sets the default length for complex series.

\[
\text{frequency} \quad \text{cseries} \quad \text{length}
\]

**Parameters**

- **cseries** Number of complex series to create. You *must* supply a value for this parameter. It can be zero, but usually is positive.
- **length** Length of the complex series.

**Description**

You can create complex series using instructions similar to those you use for real series. However, you will probably find it much more convenient to set up the full block of series (using the cseries parameter) and then refer to them with numbers rather than with names.

You can invoke FREQUENCY any number of times during the program. Each use of FREQUENCY eliminates the previous block of complex series. This makes it possible (and desirable) to write self-contained procedures for your analysis.

**Choosing length**

You usually want to choose length to be a convenient number of frequencies for the data analysis. We refer to the extra length beyond the actual number of data points as padding. There are two considerations in choosing length:

- RATS can compute the Fourier transforms much faster for series lengths which have small prime factors (2, 3 and 5) than for lengths which are products of large primes.
- The exact seasonal frequencies are included if the number of frequencies is a multiple of the number of periods per year.

The function \%FREQSIZE(n) will produce a recommended size for n actual data points. This returns the smallest integer of the form 2^m S which is greater than or equal to n. (S is the number of periods per year).
Examples

```
frequency  8  512  8 series with 512 entries
...
compute nords = %freqsize(2006:12)  Let RATS recommend a length
frequency  6  nords  6 series with that length
```

The second `FREQUENCY` eliminates the 8 series created by the first. Its series will again be 1 through 6.

Whether or not you use `%FREQSIZE`, we recommend that, as a habit, you save the number of ordinates in a variable before doing the `FREQUENCY` instruction. If you need to refer to the number of ordinates in any of your instructions, you can use your variable instead. That way, you can easily modify your program to allow for a different number of ordinates. If you’re writing a procedure where you’re working with complex series provided by the user, you can find out the `FREQUENCY` length by using `%FREQEND()` function.

Variables Defined by `FREQUENCY`

- `%Z` is a `RECTANGULAR[COMPLEX]` array with dimensions `length x cseries` which holds the complex series set up by `FREQUENCY`. You access entry `m` of complex series `n` as `%Z(m, n)`.
FRML — Creating Formulas

`FRML` is one of the most important instructions in RATS. It defines a formula which can be used for non-linear estimation, forecasting, and simulation. When you are using the `FRML` for non-linear estimation, you should do your `NONLIN` instruction first, so the free coefficients will be declared.

```
frml( options )  formulaname  = function(T)
frml( options )  formulaname  depvar = function(T)
frml(regressors, other options )  formulaname  depvar
# list of regressors  (in regression format)
```

**Wizard**

You can use the *Equation/FRML* wizard on the *Statistics* menu to define formulas.

**Parameters**

- `formulaname` The symbolic name you are assigning to this formula.
- `depvar` (Optional) Dependent (or left-side) variable for full equations.

Select which form to use based upon the following:

- **MAXIMIZE** requires only the function, and will ignore a dependent variable (`depvar`) if one is specified.
- Simulations and forecasts require a left-side variable (`depvar`).
- Formulas for **NLLS** and **NLSYSTEM** can take either form. For models which require a dependent variable, **NLLS** gives you the option of specifying the dependent variable with the `FRML` or with the **NLLS** instruction itself. With **NLSYSTEM**, however, you must specify any dependent variables in the `FRML` instructions.

**Options**

- `equation=equation to convert`
- `lastreg/[nolastreg]`
- `regressors/[noregressors]`

The (mutually exclusive) options EQUATION and LASTREG can be used to convert the specified equation or the last regression to a formula. Skip the parameters after `formulaname` when you use EQUATION or LASTREG. **You should only use FRML(EQUATION=xxx) after you have estimated the equation.** The REGRESSORS option generates the right hand side of the formula from a list of regressors supplied (in regression format) on a supplementary card. Omit the “=” symbol and function if you use this option.
vector=VECTOR for parameters
names="base for parameter names"
parmset=PARMSET to create or modify
addparms/[noaddparms]
   If you use EQUATION, LASTREG, or REGRESSORS and want to be able to use the
   formula for non-linear estimation, use the VECTOR or NAMES option to tell RATS
to construct the formula using free parameter variables, rather than fixed
numbers, for the coefficients.

   If you use the VECTOR option, the parameters will be the elements of that VEC-
tor. For instance, if you use VECTOR=BB, the parameters will be BB(1), BB(2),
etc. FRML will take care of dimensioning the vector. If you use the NAMES option,
FRML will use parameters created by appending 1, 2, etc. to base name that you
give. For instance, NAME="BB" will give you parameters BB1, BB2, etc. With
either of these options, the parameters will be initialized with the current coeffi-
cients of the regression or equation.

   If you would like the parameters to be added to a parameter set (PARMSET for
short—see NONLIN for details), use the PARMSET or ADDPARMS option, or both:

      • Use PARMSET by itself to create a new PARMSET with the name you supply. If a
        PARMSET with the same name already exists, it will be overwritten.
      • Use PARMSET with ADDPARMS to append the new parameters to an existing
        PARMSET (a new set will be created if necessary).
      • Use ADDPARMS by itself to append the parameters to the default internal
        parameter set.

   You must also use VECTOR or NAMES to specify how the parameters are created.

identity/[noidentity]
   Use the option IDENTITY when the formula you are defining is an identity.

variance=residual variance
   Residual variance for this formula. You only need to supply this if you are going
to use SIMULATE.

residuals=series of residuals
   Series holding the residuals for this formula. Currently, this information is not
used by any RATS instruction.

Description
   FRML defines a function of the form

   \[ f(X_{1t}, X_{2t}, \ldots, X_{kt}, \beta_1, \beta_2, \ldots, \beta_p) \]
or
\[ y_t = f(X_{t1}, X_{t2}, \ldots, X_{tk}, \beta_1, \beta_2, \ldots, \beta_p) \]

where each \( X_t \) is a RATS data series, and each \( \beta \) is a REAL variable or element of a real-valued array. If you are using the FRML for non-linear estimation, you will usually need to do a NONLIN instruction prior to defining the FRML to introduce the parameters. Note that you also can define formulas without any \( \beta \) parameters.

RATS stores the relationship in a variable of type FRML, with the name you supply for formulaname. In general, you write the function in terms of the INTEGER time subscript \( T \) in the same manner as the right side expression is written for SET. As with SET you can use either series or series(\( T \)) to refer to the current value of series and series(lag) for a lagged value.

Once you have defined a formula (and specified values for any \( \beta \) variables, if there are any), you can use it in expressions with

\[ \text{formulaname}(\text{entry}) \]

which will take the value of function(entry). You can do this no matter which form of the FRML command you chose—RATS only uses the function(\( T \)) part in the full equations.

**Examples**

```
frml avgvalue = (gdp{2}+gdp{1}+gdp+gdp{-1}+gdp{-2})/ 5.0
compute avg1909 = avgvalue(1909:1)
compute avg1901 = avgvalue(1901:1)
compute avg1885 = avgvalue(1885:1)
```
evaluates five year moving averages of GDP around specified time periods.

```
frml(identity) gdpid gdp = consmptn+invest+govtexp+netexp
frml(identity) logmlid logml = log(ml)
*  
sur(inst) 3
# conseq
# inveq
# wageeq
*  
frml(equation=conseq) consfrml
frml(equation=inveq) invfrml
frml(equation=wageeq) wagefrml
```
The first FRML instruction sets up GDPID as the standard income accounting identity. The second is a definitional identity required for certain types of models. The last group of instructions estimates a set of three equations (CONSEQ, INVEQ, WAGEEQ) by three stage least squares, then converts the three estimated equations to formulas.
Simplifying Your Life—Using “Sub FRMLs”

Your formulas can include references to other formulas, as long as the “sub” FRML has been defined earlier. This allows you to separate out the different parts of an overall model, into, say, a regression model and a variance model. For instance, the following shows two equivalent code fragments to estimate a regression of $Y$ on $X1$ and $X2$ with first-order autocorrelation correction using common factor restrictions. (Note: these should give almost identical results to $AR1$).

\begin{verbatim}
nonlin rho b1 b2 b3
frml auto1 = rho*y{1} + (1-rho)*b1 + $
           b2*(x1-rho*x1{1}) + b3*(x2-rho*x2{1})
linreg y
# constant x1 x2
compute rho=%rho, b1=%beta(1), b2=%beta(2), b3=%beta(3)
nlls(frml=auto1) y
\end{verbatim}

or:

\begin{verbatim}
nonlin rho
linreg y
# constant x1 x2
frml(lastreg,names="B",addparms) regfrml
frml auto1 = rho*y{1} + regfrml(t) - rho*regfrml(t-1)
compute rho = %rho
nlls(frml=auto1) y
\end{verbatim}

The second example builds $AUTO1$ using a second formula (named $REGFRML$) for the regression part. This can make it simpler to code a complex formula, and does make it simpler to change it. If you want to add an additional explanatory variable to the model the first way, you need to change the NONLIN, FRML, LINREG and COMPUTE instructions. All you need to do in the second is change the supplementary card on LINREG. The first FRML instruction in this second example does all of the following:

- Creates $REGFRML$ with the variables $B1$, $B2$ and $B3$ for the three regression coefficients.
- Initializes $B1$, $B2$ and $B3$ to the values estimated by LINREG.
- Adds $B1$, $B2$ and $B3$ to the list of NONLIN parameters.

If you are creating a formula using a number of “subfrmls,” we would strongly suggest that you explicitly use subfrml(T) (as we’ve done above) rather than subfrml alone when you need to refer to the subformula. This makes it easier to tell formula references from series references.
Reducing Computing Time

You are likely to find subformulas a very handy device in writing your FRMLS. They may, however, obscure how much redundant calculation is done by the overall formula. The instruction

\[
\text{frml truncate} = \%\text{logdensity}(\text{sigmasq}, y - \text{rhsfrml}) - \$
\log(\%\text{cdf}((\text{rhsfrml}(t) - \text{tr})/\text{sqrt}(\text{sigmasq})))
\]

sets up the log likelihood function for a truncated sample, where RHSFRML is a sub-formula representing the model. As written, this will compute RHSFRML twice for each individual on each function evaluation. This can be reworked to evaluate RHSFRML just once:

\[
\text{frml truncate} = (z = \text{rhsfrml}(t)) , \$
\%\text{logdensity}(\text{sigmasq}, y - z) - \$
\log(\%\text{cdf}((z - \text{tr})/\text{sqrt}(\text{sigmasq})))
\]

This calculates RHSFRML once (per individual/per evaluation), saving it into the scalar Z, which is then used in the rest of the formula. This cuts the computing time substantially if RHSFRML is long or complex.

Self-Referencing Formulas

You cannot create a FRML which is defined using an explicit reference to itself. If the value of your formula at \( T \) depends upon previously calculated values of itself (directly or indirectly), you need to write your FRML to store the values into a series as they are computed. The lagged values then come from the series.

An example is the ARCH-M model (User’s Guide, Section 12.1.4). If the square roots of the variance (rather than the variance itself) enters the mean equation, the GARCH instruction won’t handle it. To estimate with MAXIMIZE, this needs one formula for the variance and one for the residual. However, the variance function depends upon the lagged residual, and the regression function depends upon the variance. We cannot write down two simple FRMLS without each referencing itself.

The solution, as shown in that section, is to save the residuals in a series (this also saves the squared residuals):

\[
\text{set } u = 0.0 \\
\text{set } uu = \%\text{seesq} \\
\text{frml archvar} = a0 + a1 \times uu(1) \\
\text{frml regresid} = y - b1 - b2 \times x1 - b3 \times \text{sqrt}(\text{archvar}(t)) \\
\text{frml archlogl} = u(t) = \text{regresid}(t), uu(t) = u(t)**2, \$
\%\text{logdensity}(\text{archvar}(t), u)
\]

We now give a somewhat quicker formulation which also saves the variance in a series. This eliminates redundant calculations of ARCHVAR.
set u = 0.0
set uu = %seesq
set v = %seesq
frml archvar = a0 + a1 * uu{1}
frml regresid = y - b1 - b2 * x1 - b3 * sqrt(v)
frml archlogl = (v(t)=archvar(t)), (u(t)=regresid(t)), $ uu(t)=u(t)**2, %logdensity(v,u)

When you have several layers of formulas, be careful that calculations are done in the correct order. REGRESID depends upon the current value of ARCHVAR, so ARCHVAR has to be calculated first.

VECTORs of FRMLs

RATS allows you to create arrays of FRMLs. For instance,

dec vect[frml] f(3)
nonlin b1 b2
frml f(1) = exp(b1+b2*log(x))
frml f(2) = (b1+b2*sqrt(x))**2
frml f(3) = b1+b2*x
compute b1=b2=0.0
do i=1,3
   nlls(frml=f(i)) y
end do i

will estimate three functional forms for the relationship between Y and X.

You have to be careful, however, if the FRML’s are defined in a loop. Wherever you use the loop index in the formula definition, you must prefix it with the & symbol. For instance, the following sets up formulas to estimate Black’s version of CAPM (Campbell, Lo and MacKinlay (1997), chapter 5) for N assets. The asset returns are in the vector of series S.

dec vector b(n)
dec vect[frml] blackf(n)
nonlin gamma b
do i=1,n
   frml blackf(i) s(i) = (1-b(&i))*gamma+b(&i)*market
end do i

The &i’s are needed in the formula because (1-b(i))*gamma+b(i)*market is a perfectly good formula, which, when it is evaluated by NLSYSTEM or MAXIMIZE later will use the value of i at the time that the estimation instruction is executed. &i tells FRML to use the value of i at the time that the formula is defined.
FRMLs of other Types

You can also use FRML to define formulas which evaluate to data types other than REAL. These are typically used as sub-formulas for a final FRML which does give a real value.

To do this, you must first DECLARE the type of the formula. The formula definition on FRML should then evaluate to the type that you want. As examples:

```
declare frml[vect] ub
frml ub = ||u1{1},u2{1}||
```

```
declare frml[complex] transfer
frml transfer = (1+b*%zlag(t,1))/$
                   ((1-a*%zlag(t,1))**conjg((1-%zlag(t,1))**D))
```

See Also . . .

GROUP
Combines FRMLs to build a forecasting model.

EQUATION
Defines linear relationships.
FUNCTION — User-Defined Functions

The FUNCTION instruction begins the definition of a user-defined function. It specifies the name of the function and the parameter list, if any.

```
function    funcname    parameters
```

Parameters

- **funcname**: The name you want to give this function. `funcname` must be distinct from any other function, procedure, or variable name in the program. By default, the function will return a real value. You can change that by using a TYPE command to define `funcname` as a different type.

- **parameters**: Names of the parameters. These names are local to the function—they will not conflict with variables elsewhere in the program. By default, parameters are INTEGER passed by value. You can change this using TYPE statements.

Description

User-defined functions are very similar to RATS procedures (see Section 16.2 of the User’s Guide). However, procedures are designed to mimic RATS instructions, and thus cannot be called from within an expression. Functions, on the other hand, are designed to be used in expressions, and thus make it possible to embed complex computations in FRMLs and other expressions. As a result, functions make it possible to handle a wide range of optimization problems that would be very difficult or impossible to handle without them.

A function definition begins with a FUNCTION statement and ends with a matching END. The FUNCTION statement itself names the function and lists the formal parameters, if any.

You need to use function instructions in the following order:

```
function    statement
type        statements, if any, to define the type of the function and any parameters
other        instructions
end
```

If possible, you should try to write the function using only parameters, local variables and global variables which RATS itself defines. That way, you don’t have to worry about accidentally changing a global variable elsewhere in the program.
Using SOURCE

As with procedures, you may find it convenient to store functions that you’ve written on separate files, so that they can be used in different applications. If you have a function stored on a separate file, bring it into your current RATS program using the instruction `SOURCE`. The typical instruction is

```
source file with function
```

If you have a collection of functions which you use regularly, you can include them in a “procedure library” that gets brought in right at the start of your program (via the `File-Preferences` operation, or an `ENVIRONMENT` instruction).

Examples

This computes the tri-gamma function (second derivative of log gamma) of its argument. Note that the function checks for an out-of-range argument and returns the missing value in that case. Note also that the function name is used repeatedly in the body of the function for the intermediate calculations. This is permitted; the only thing special about it (compared with any other local variables) is that the final value assigned to it before the return will be the value returned by the function. (Note: the function `%TRIGAMMA` is now built-in).

```r
function TriGamma z
type real TriGamma z
local real zp zpr *

if z<=0 {
    compute TriGamma=%NA
    return
}
compute zp=z
compute TriGamma=0.0
while (zp<30) {
    compute zpr=1/zp
    compute TriGamma=TriGamma+zpr*zpr
    compute zp=zp+1
}
compute zpr=1/zp
compute TriGamma=TriGamma+zpr*(1+zpr*(.5+zpr*(1.0/6.0+zpr**2*$
    (-1.0/30.0+zpr**2/42.0))))
end
```
The example program \texttt{BONDS.PRG}, included with RATS, fits a yield curve to a set of bonds.

\begin{verbatim}
nonlin a0 a1 a2
compute a0=.030,a1=a2=0.0
compute cusp=2.0

function BondPV bond
type real BondPV
type integer bond

local real mdate cdate

compute mdate=maturity(bond)
compute BondPV=100.0 * $
    \exp(-mdate*(a0+mdate*a1+$
        \max(mdate-cusp,0.0)*a2))

* Walk backwards through the coupons.
*
compute cdate=mdate
while (cdate > 0.0) {
    compute BondPV=BondPV + coupon(bond) * $
        \exp(-cdate*(a0+cdate*a1+$
            \max(cdate-cusp,0.0)*a2))
    compute cdate=cdate-1
}

* Adjust for simple interest payable by
* the purchaser for the initial coupon.
* cdate will be -(fraction of period).
*
compute BondPV=BondPV+coupon(bond)*cdate
end
\end{verbatim}

With the evaluating function defined, this set up for using \texttt{NLLS} is quite simple:

\begin{verbatim}
frml BondPrice value = BondPV(t)
*
nlls(robust,trace,frml=BondPrice) value
\end{verbatim}
GARCH — ARCH and GARCH Models

GARCH estimates univariate and multivariate ARCH and GARCH models. You can also estimate ARCH and GARCH models using MAXIMIZE. However, for any model that GARCH supports, you will probably find it more convenient to use GARCH. Also, because the GARCH command is specifically designed for these models, it will usually run much faster than MAXIMIZE—approximately five times faster on average. See Section 12.1 for more information.

```
garch( options ) start end list of series
# explanatory variables for the mean equation (if REGRESSORS)
# extra explanatory variables for the variance equation (if XREGRESSORS)
```

Wizard

The ARCH/GARCH wizard on the Statistics menu provides an easy way to estimate a variety of ARCH and GARCH models. However, the GARCH instruction has several options not included in the Wizard, so be sure to refer to the information below if you need capabilities beyond those available via the wizard.

Parameters

```
start   end       Range to use in estimation. If you have not set a SMPL, this
defaults to the largest common range for all the variables
involved.
```

```
list of series  List of one or more dependent variables
```

ARCH/GARCH Options—Apply to All Models

The following options apply to both univariate and multivariate models:

```
p=number of GARCH (lagged variance) terms [0]
q=number of ARCH (lagged residual squared terms or analogous) [0]
```

These specify the order of the GARCH($p,q$) or ARCH($q$) model.

```
distrib=[normal]/t/ged
shapeparm=input value for shape parameter for t or GED [estimated]
```

The assumed distribution of the error process, Normal, t (Student) or Generalized Error Distribution. GED is only available for univariate models. If using $T$ or GED, you can use SHAPEPARM to provide your own value for the shape parameter. If you don’t, it will be estimated.

```
asymmetric/[noasymmetric]
```

Includes an asymmetry term. For a standard GARCH model, this will give you the GJR (Glosten, Jagannathan, Runkle (1993)) model. This can be used with all of the supported univariate and multivariate models.
**mean/nomean**  [default is MEAN unless you use REGRESSORS]

MEAN includes a constant term in the mean equation(s). Use NOMEAN if you don’t want any mean terms at all. To supply your own mean equation(s), use REGRESSORS or EQUATION for univariate models, or MODEL for multivariate models.

**xregressors/ [noxregressors]**

Use this if you want some exogenous shift variables in the variance equation(s). If you use it, include them on a supplementary card. If you are also using the REGRESSORS option, the XREGRESSORS are on a second card. For multivariate models, the same set of regressors are included in all variance equations.

**presample=SYMMETRIC matrix of pre-sample values**

Use this to supply pre-sample values for the lagged variances and lagged squared residuals. For a univariate model, you can do PRESAMPLE=value.

**condition/ [nocondition]**

If CONDITION, GARCH conditions on the required lagged residuals rather than assigning them the presample values.

**i=nodrift/ drift**

The I option constrains the GARCH coefficients (all a’s and b’s) to sum to one. For the multivariate models, each component is constrained separately. With I=NODRIFT, the constant terms in the variance equations are constrained to zero. With I=DRIFT, those constant terms are estimated. Note: The I option can not be used with MV=BEKK or MV=VECH multivariate models.

**ARCH/GARCH Options—Univariate Models Only**

These options are used only for univariate models.

**regressors/ [noregressors]**

**equation= equation form to use for the mean**

By default, the mean equation for the dependent variable is just a single parameter for a time-invariant mean. If you want any other mean equation, use the REGRESSORS option, and include all the regressors (including the CONSTANT if needed) on a supplementary card. As an alternative to REGRESSORS, you can use the EQUATION option to provide an equation that you’ve already defined.

**exponential/ [noexponential]**

Use EXPONENTIAL if you want to do the E-GARCH model of Nelson (1991) or a generalization of it.

**hseries=SERIES of estimated variances** (univariate)

**resids=SERIES of (non-standardized) residuals** (univariate)

If estimating a univariate model, you can use HSERIES and RESIDS options to save the estimated variances and residuals, respectively, into SERIES.
ARCH/GARCH Options—Multivariate Models Only

These options are used only for multivariate models.

\textbf{mv=[standard]/bekk/diagonal/cc/dcc/vech/ewma}

Chooses the form for a multivariate model. The default is the standard multivariate \textit{GARCH} model. \textit{BEKK} gives the “BEKK” formulation (also known as BEK or EK), which imposes positive-definiteness on the covariance matrix. The other choices are restricted correlation models. \textit{DIAGONAL} estimates separate univariate \textit{GARCH} models for each dependent variable, so that covariances are restricted to zero. \textit{CC} gives the Constant Correlation model, \textit{DCC} implements the Dynamic Conditional Correlations model. \textit{VECH} estimates the “vech” model. \textit{EWMA} implements an exponentially weighted moving average model—an IGARCH model with a single parameter applying to all components. See Section 12.1 for details.

\textbf{variances=[simple]/varma/exponential}

Use the \textit{VARIANCES} option to select the form of the variance terms.

\textbf{model=MODEL (of linear equations) for the mean equations}

Use \textit{MODEL} if you want to supply your own mean equations. You can use \textit{LINREG} or \textit{EQUATION} commands to define the equations, and then use \textit{GROUP} to group them into a model. Otherwise, the \textit{MEAN/NOMEAN} option determines the form of the mean equation. By default (\textit{MEAN}) the mean equations will include only a constant term. Use the \textit{NOMEAN} option if you don’t want the constant term.

\textbf{hmatri}\textit{ces=SERIES[SYMM]} estimated covariance matrices
\textbf{mvhseries=SYMM[SERIES]} estimated variance/covariance series
\textbf{rve\texttildetilde{t}ectors=SERIES[VECT]} of non-standardized residuals (multivariate)

If estimating a multivariate model, you can use \textit{HMATRICES} to save the estimated covariance arrays as a \textit{SERIES} of \textit{SYMMETRIC} arrays, \textit{MVHSERIES} to save the variances and covariances as a \textit{SYMM[SERIES]}, and \textit{RVECTORS} to save the (non-standardized) residuals to a \textit{SERIES} of \textit{VECTORS}. Note that \textit{HMATRICES} and \textit{MVHSERIES} save the same information, just in different types of variables.

\section*{General Options}

The following are common to most of the non-linear estimation commands in RATS.

\textbf{method=bhhh/[bfgs]/simplex/genetic/evaluate}
\textbf{pmethod=bhhh/bfgs/[simplex]/genetic}
\textbf{p\texttildetilde{it}ers=number of PMETHOD iterations to perform [none]}

\texttt{BHHH} is Berndt, Hall, Hall and Hausman; \texttt{BFGS} is Broyden, Fletcher, Goldfarb and Shanno; \texttt{SIMPLEX} is the simplex algorithm; and \texttt{GENETIC} is a genetic search algorithm. See Chapter 7 in the \textit{User’s Guide} for a technical description of these.

\texttt{SIMPLEX} and \texttt{GENETIC} are derivative-free methods which can compute point estimates of the coefficients but not standard errors. They can be helpful in refining initial guess values before applying one of the derivative-based methods.
With `METHOD=EVALUATE`, RATS simply evaluates the model given the initial parameter values, without trying to estimate new coefficient values. You can use options like `HSERIES` and `RESIDS` to save the resulting variance and residual series for evaluation.

Use `PMETHOD` and `PITERS` if you want to use a preliminary estimation method to refine your initial parameter values before switching to one of the other estimation methods. For example, to do 20 simplex iterations before switching to BFGS, use `PMETHOD=SIMPLEX, PITERS=20, and METHOD=BFGS`.

```
[print]/noprint
vcv/[novcv]
```

These control the printing of regression output and the printing of the estimated covariance/correlation matrix of the coefficients (User’s Guide, Section 5.1), respectively.

```
robusterrors/[norobusterrors]
lags=correlated lags [0]
lwindow=neweywest/bartlett/damped/parzen/quadratic/[flat]
damp=value of \( \gamma \) [0.0]
lwform=VECTOR with the window form [not used]
```

These permit calculation of a consistent covariance matrix allowing for heteroscedasticity or (with `LAGS`) serial correlation (you need the `ROBUSTERRORS` option in either case). See Sections 5.3, 5.4, and 5.5 of the User’s Guide and the description of the instruction `MCOV` for more information.

```
iterations=iteration limit [100]
subiterations=subiteration limit [30]
cvcrit=convergence limit [.00001]
trace/[notrace]
```

`ITERATIONS` sets the maximum number of iterations, `SUBITERS` sets the maximum number of subiterations, `CVCRIT` the convergence criterion. `TRACE` prints the intermediate results. See Chapter 7 in the User’s Guide for more details.

```
initial=VECTOR of initial guess values for the parameters
```

`INITIAL` can be used to feed in initial guess values for estimation, or can be used with `METHOD=EVALUATE` for evaluating the likelihood at specific values. See “Order of the Coefficients” below for the order in which you need to place these.

```
hessian=initial estimate for inverse Hessian (for METHOD=BFGS)
```

You can use this with `METHOD=BFGS`. Without it, `GARCH` will start with the identity matrix. See Section 7.2 in the User’s Guide.
Order of the Coefficients
The parameters are added in the following order:

1. Mean model coefficients
2. Constant terms in the GARCH variance model
3. Lagged squared variance (ARCH) terms
4. Lagged variance (GARCH) terms
5. Asymmetry terms in the variance model (if any)
6. Extra explanatory variables in the variance model (if any).

For multivariate models, many of these are matrices. Within each of these categories, the elements are ordered by row. For a matrix which is symmetrical or lower triangular, this would mean (1,1), (2,1), (2,2), (3,1), etc. For instance, with the simple multivariate GARCH, the first will be for the model for the variance of series 1, the second for the covariance of 1 with 2, the third for the variance of 2, etc. If you use more than one lag, the each lag is in a separate block.

Variables Defined by GARCH

%BETA Coefficient vector (VECTOR).
%XX Covariance matrix of coefficients, or $X'X^{-1}$ (SYMMETRIC)
%TSTATS Vector containing the t-stats for the coefficients (VECTOR)
%STDERRS Vector of coefficient standard errors (VECTOR)
%NOBS Number of observations (integer)
%NREG Number of regressors (integer)
%NVAR number of variables (integer)
%RESIDS Series containing the residuals (series)
%FUNCVAL Log likelihood (real)
%LOGL Log likelihood (%FUNCVAL and %LOGL contain the same value for GARCH) (real)
Examples

ARCH(2) on a AR(1) regression model, with the presample variance fixed at the value of %SEESQ from a previous regression.

\[
garch(p=0,q=2,\text{regressors},\text{presample}=|\text{%seesq}|) / \text{ffed}
\]
# constant ffed{1}

ARCH(1) with a \( t \)-distribution with 5 degrees of freedom, saving the residuals in \( AT \) and the variance estimates in \( FVAR \).

\[
garch(p=0,q=1,\text{dist}=t,\text{shape}=5,\text{resids}=at,\text{hseries}=fvar) / \text{rt}
\]

EGARCH(1,1) model with asymmetry on an MA(1) mean model.

\[
garch(p=1,q=1,\text{exp},\text{asymmetric},\text{regressors}) / \text{ibmln}
\]
# constant %mvgavge{1}

GARCH-X(1,1), adding the variable \( U \) to the variance equation.

\[
garch(xreg,p=1,q=1,\text{resids}=at,\text{hseries}=f) / \text{rt}
\]
# u

Bivariate GARCH(1,1) model

\[
garch(p=1,q=1,\text{method}=bfgs) / \text{futures spot}
\]

Trivariate DCC GARCH(1,1) model, with separate mean models for each variable.

\[
group\text{ meanm rspeq rnkeq rhseq}
garch(p=1,q=1,\text{mv}=\text{dcc},\text{model}=\text{meanm},\text{method}=\text{bhhh},\text{iters}=200)
\]
GContour

GCONTOUR — Contour Plots

GCONTOUR produces high-resolution contour plots. It has some similarities to the SCATTER instruction, which produces X vs. Y scatter plots, but differs from other graphics instructions because the data are input using matrices rather than series.

```plaintext
gcontour( options ) hfield vfield
```

Parameters

hfield vfield These, in conjunction with the HFIELDS and VFIELDS options of SPGRAPH, allow you to put multiple graphs on a single page.

Options—Quick Reference

The following is a list of all of the options for GCONTOUR. Many of these are identical to options on SCATTER, while some are unique to GCONTOUR. The options specific to GCONTOUR are described in detail here. See SCATTER for details on the other options.

### GCONTOUR Options

- **x=VECTOR of X grid values**
- **y=VECTOR of Y grid values**
- **f=RECTANGULAR with function values**
- **number=number of contour lines**
- **contours=VECTOR of specific “f” values**

### Options Common to SCATTER and GCONTOUR

- **axis=none/vertical/horizontal/[both]**
- **extend=[none]/vertical/horizontal/both**
- **footer=footer label**
- **frame=[full]/half/none/bottom**
- **header=header string for graph**
- **hgrid=VECTOR of grid values**
- **hlabel=horizontal scale label**
- **hlog=base for a log scale**
- **hmax=value for right boundary**
- **hmin=value for left boundary**
- **hpicture=picture clause for x-axis**
- **hscale=[lower]/upper/both/none**
- **hshade=RECTANGULAR with shading zones**
- **hticks=max number of horizontal ticks**
- **lines=RECTANGULAR with slope/intercept**
- **subhead=subheader string**
- **vgrid=vector of grid line values**
- **vlabel=vertical scale label**
- **vlog=base for a log scale for y-axis**

Function

- Provides X series grid values
- Provides Y series grid values
- Provides the function values
- Sets the number of contour lines
- Provides values for contour lines

Function

- Draw x=0 and/or y=0 axes
- Extend grid lines across graph
- Adds a footer label below graph
- Controls frame around graph
- Adds a header to the graph
- Sets position of grid lines
- Adds a label to the x-axis.
- Selects a log scale for x-axis
- Sets the maximum x-axis value
- Sets the minimum x-axis value
- Format of x-axis scale values
- Placement of x-axis scale
- Shading zones for x-axis
- Number of tick marks on x-axis
- Draw lines given slope/intercept
- Adds a sub-header to the graph
- Sets grid for vertical axis
- Label for vertical axis
- Selects a log scale for y-axis
vmax=value for upper boundary
Sets maximum y-axis value
vmin=value for lower boundary
Sets minimum y-axis value
vpicture=picture clause for y-axis
Formatting of y-axis scale values
vscale=[left]/right/both/none
Placement of vertical scale
vshade=RECTANGULAR with shading zones
Shading zones for y-axis
vticks=max number of vertical ticks
Number of tick marks on y-axis
window=string for window title
Custom title for graph window
xlabels=VECT[STRING] for x-axis labels
Strings for labeling x-axis

GCONTOUR Options in Detail
Here we look at the options that are specific to GCONTOUR. Please see the sections on GRAPH and SCATTER for details on the other options.

x=VECTOR of X grid values [required]
y=VECTOR of Y grid values [required]
f=RECTANGULAR array supplying function values [required]
Use the X option to supply a vector with the grid values for the x-axis, and the Y option to supply a vector with grid values for the y-axis. Use the F option to supply a RECTANGULAR array with dimensions \((\text{dim}(x)) \times (\text{dim}(y))\) containing the function values for each \(x,y\) grid point. For example, \(F(1,2)\) should contain the function value associated with the grid point given by \(X(1),Y(2)\). \(X\) and \(Y\) are usually created using the \%SEQA function or EWISE instruction. The \(F\) array is usually set using EWISE.

number=number of contour lines [30]
contours=VECTOR of specific “f” values
You can use CONTOURS to supply a VECTOR containing the specific function values at which you want contour lines to be plotted. Otherwise, RATS will draw the number of contour lines specified by the NUMBER option, with the contour values chosen to provide a roughly equal distance between each value.

Example
This does contours of the likelihood surface for a GARCH(1,1) model. To avoid an excessive number of contours, the function values are set to NA if too small.

```plaintext
compute [vector] atest=%seqa(.002,.002,100)
compute [vector] btest=%seqa(.500,.005,100)
dec rect ftest(100,100)
do i=1,100
   do j=1,100
      garch(initial=||%beta(1),lrvar*(1-btest(i)-atest(j)),
         atest(j),btest(i)||,p=1,q=1,method=eval) / sp500
      compute ftest(i,j)=%if(%funcval<base-50,%na,%funcval)
   end do j
end do i
gcontour(x=btest,y=atest,f=ftest)
```
Graph

**GRAPH — High-Resolution Time Series Graphs**

*GRAPH* produces high-resolution time series graphs. It has a large number of options, which permits you to control most aspects of the presentation.

<table>
<thead>
<tr>
<th><code>graph( options )</code></th>
<th>number</th>
<th>hfield</th>
<th>vfield</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>num</code></td>
<td><code>start</code></td>
<td><code>end</code></td>
<td><code>representation</code> (one card for each series)</td>
</tr>
</tbody>
</table>

**Wizard**

Select *Graph* from the *Data* menu to access the *Graphics* wizard. Note that in order to keep the Wizard from getting too complicated, some of the *GRAPH* options have been omitted in the Wizard. If you want to further customize your graph, you can edit the *GRAPH* instruction generated by the Wizard by adding the desired options.

You can also display time series plots, histogram plots, and box plots from the *Series List* window. First, select *Show Series Window* from the *Data* menu to display this window. Then select (highlight) one or more series in the window and click on one of the graphing buttons on the toolbar. You can do the same for series stored on a RATS format file, via the *Open RATSDATA* operation on the *Data* menu.

**Parameters**

- **number**
  Number of series to graph. The maximum permitted is twenty.
- **hfield** and **vfield**
  When using *SPGRAPH* to put multiple graphs on a single page, these allow you to put a graph in a specific field. By default, the fields are filled by column, starting at the top left (field 1,1).

**Supplementary Cards**

Use one supplementary card for each series on the graph.

- **series**
  the series to be graphed.
- **start** and **end**
  (Optional) the range to graph. If you have not set a *SMPL*, this defaults to the defined range of *series*. *start* and *end* can be different for each series in the graph.
- **representation**
  (Optional) An integer value (1 through 30) specifying the line, pattern, or symbol style (depending on the graph type) to use for *series*. By default, RATS uses different colors (or patterns, for black and white output) for each series. You can choose from the default styles provided by RATS, or you can use style sheets to customize attributes such as color or gray scale levels, line thickness, fill patterns, and more. See Section 3.5 in the *User’s Guide* for details.
Options

This is a brief alphabetical listing of the options. Detailed descriptions organized by function follow.

[axis]/noaxis
[box]/nobox
[dates]/nodates
extend/[noextend]
footer=footer label
frame=[full]/half/
none/bottom
grid=gridseries
header=string
hlabel=label
[kbox]/nokbox
key=[none]/upleft/upright/
loleft/loright/above/
below/left/right/attach
kheight=value
klabel=vect[strings]
width=value
[ksample]/noksample
log=value
max=value
min=value
number=number
omax=value
omin=value
ovcount=number
overlay=(see style)
[ovkey]/noovkey
ovlabel=label
ovrange=fraction
ovsame/[noovsame]
patterns/[nopatterns]
picture=pict.clause
scale=[left]/right/
both/none
shading=series
smpl=series or frml
style=[line]/polygon/bar/
stackbar/overlap/
vertical/step/symbol/
midpolygon/fan/dots/spike

Draw horizontal axis if Y=0 is within graph bounds
Replacement by the FRAME option
Label entries with dates
Extend horizontal grid lines across graph
Adds a footer label below graph
Controls frame around the graph
Series with non-zeros where you want vertical lines
Adds a header to the top of the graph
Add a label to the horizontal axis
Controls whether a box is drawn around the key
Allows addition of key to the graph
Specifies key box height as fraction of graph height
Used to supply your own labels for the graph key
Specifies key box width as fraction of graph width
Controls whether samples are included in the key
Base for log scale graphs
Value for upper boundary of graph
Value for lower boundary of graph
Starting number for x-axis labels (with NODATES)
Value for upper boundary of overlay scale
Value for lower boundary of overlay scale
Number of series for right-side (overlay) scale
Style for overlay (creates a two-scale graph)
Adds a key for the overlay series, if any
Scale label for the right side of an overlay graph
Controls offset of vertical scales in overlay graph
Use same scale for both axes of an overlay graph.
Use patterns, not colors, to distinguish series
Picture clause for axis label numbers
Placement of vertical scale
Series with non-zeros at entries to be shaded
Series/formula indicating entries to be graphed
Style of the graph
Graph

- **subhead** = *string*  
  Subheader string for graph
- **[ticks]/noticks**  
  Include tick marks, dates/entries on horizontal axis
- **vgrid** = *vector*  
  Values for grid lines across from vertical axis
- **vlabel** = *label*  
  Label for the vertical axis
- **vticks** = *number*  
  Maximum number of vertical ticks
- **window** = *string*  
  Title for graph window
- **xlabel** = *VECT[STRINGS]*  
  For labeling points on the horizontal axis

**General Options**

- **patterns/[nopatterns]**  
  This chooses the way you want **GRAPH** to distinguish among the series. RATS normally uses different colors, and will automatically switch to black and white patterns if you print the graph on a black and white printer. If you want to see on the screen (approximately) how the black and white hard copy will appear, use the option **PATTERNS**—RATS will display the series with patterns rather than colors. Note that you can also switch between patterns and colors after displaying the graph by using the toolbar icon on the graph window.

- **window** = "**Window title**" (in quotes: "...") or **STRING**  
  When working in interactive mode, the **WINDOW** option allows you to set a title for the graph window that will be associated with the graph. By default, graph windows are titled as "Graph.01," "Graph.02," etc.

- **footer** = **STRING for graph footer**  
  This adds a label to the graph. The label will be left-justified in the lower left corner of the graph.

- **frame** = [full]/half/none/bottom  
  This controls the “bounding box” displayed around the outside of the graph. The **HALF** choice displays the frame to the left and below the graph, omitting the top and right sides.

- **[box]/box**  
  **NOBOX** eliminates the “bounding box” line, around the graph. This option has been superseded by the more flexible **FRAME** option described above.

- **smpl** = **SMPL series or formula**  
  You can supply a series or a formula that can be evaluated across entry numbers. Only entries for which the series or formula are non-zero will be graphed.
**Graph**

```
style=[line]/polygonal/bar/stacked/overlapping/
  vertical/step/symbol/midpolygon/fan/dots/spike
```

LINE is a simple line graph. It draws a line from one point to the next.

POLYGONAL draws a line from one point to the next, and paints the region between this line and the X-axis (or bottom of the graph if the minimum is greater than 0). *Recommended only when graphing a single series.*

BAR draws a separate rectangle for each data series at each entry. If you are graphing more than one series, you can’t really use BAR for more than about 100 data points, as the bars get too thin.

STACKED is only useful with a set of non-negative series. With the STACKED option, RATS stacks the bars for all the series at a given time point into a single large rectangle.

OVERLAP is similar to BAR, except that the bars overlap somewhat. This allows it to be used with more data points or series than the simple bar graph. Since this paints the bar for the second series over part of the first series, the third over part of the second, and so on, this style works best when the first series is the largest and the last the smallest.

VERTICAL connects all values at a given time period with a vertical line, with hash marks at all the values. You can use this for high/low/close plots or for plotting confidence intervals.

STEP is similar to LINE except instead of drawing a line directly from one point to the next, it draws horizontally to the new “x” position, then vertically to the new “y” position.

SYMBOLS is similar to LINE except that it draws symbols at regular intervals along the line. This may produce a better printed copy of the graph if you have a number of intertwined series.

MIDPOLYGON is like POLYGON, except that the polygons are centered on tick marks (similar to BAR), rather than centered between tick marks.

FAN creates a fan chart, filling in the gap between series with a set of shaded fill patterns, getting lighter towards the outside. Can be used to fill space between two series, but is most useful with five or more series.

DOTS plots each data point with a large dot.

SPIKE is similar to a bar graph, but uses narrow spikes rather than wide bars.

**Graph Header Options**

```
header="Header string" (in quotes: "...") or STRING) [none]
```

The header is centered above the graph. Although RATS will accept a string of up to 256 characters, it may not be possible to fit such a long string on the page. To get a very long header to fit, you may have to change the header font size (which is drawn in 24 point boldface by default) using GRPARM.
subheader="Subheader string"  (in quotes: "...") or STRING) [none]
The subheader is centered above the graph and below the header. By default, it is presented in 18 point italics.

Time Axis Options

hlabel="Horizontal scale label"  (in quotes: "...") or STRING) [none]
HLABEL is centered below the graph and below the entry labels (if any).

[ticks]/noticks
NOTICKS suppresses the tick marks and entry labeling on the time axis.

[dates]/nodates
By default, GRAPH labels the entries with dates (if possible) on the horizontal axis. How RATS represents the dates depends upon the font sizes (controlled with GRPARM), the number of entries graphed, and the size and shape of the graph. For instance, with a relatively short series of daily data, GRAPH will probably use full month names with a date at each entry. With a long annual series, it may only be able to label one out of five years. With the NODATES option, GRAPH labels graphs with entry numbers (see NUMBER option below).

Note: When RATS prints a year or month label, it is centered under the entries covered by that year or month.

number=labeling number for first entry
You can use NUMBER (with NODATES) when you want to use some sequence other than entry numbers to label the axis. The labeling number is the number assigned to the first entry on the graph. For instance, RATS stores autocorrelations with the 0 lag in entry 1. To have them labelled correctly when graphed, use the options NODATES and NUMBER=0.

grid=series with non-zeros at entries for grid lines
shading=series with non-zeros at entries to be shaded
GRID and SHADING are alternative options for highlighting sets of entries. GRID merely puts a vertical grid line at each entry for which the series has a non-zero value. SHADING is a little more complicated: any block of consecutive non-zero entries will produce a lightly shaded region on the graph covering those entries.

xlabels=VECTOR of STRINGS for labeling points on the time axis
This allows you to override the standard date or entry labels by providing your own labels. These will be spaced equally across the x-axis. If you don’t have one label per data point, be very careful that the labels line up correctly with the data.
Vertical Axis Options

**scale=**[left]/right/both/none

The vertical scale indicates the range of values graphed. You can place it on either or both sides of the graph, or omit it completely.

**vlabel=**"Vertical scale label" (in quotes:"..." or STRING) [none]

This places the Vertical scale label on the SCALE side(s) of the graph. The label will appear to the outside of the tick marks, centered vertically.

**log=**base for a logarithmic scale [not used]

If the LOG option is used, the data are plotted on a log scale. The base really affects only the levels that get labeled, which will always be powers of the base. The values of 10, 2, 4 and 5 are most likely to work best.

**maximum=**value for upper boundary [largest value]

**minimum=**value for lower boundary [smallest value]

Use these (alone or together) to set upper and lower bounds on the vertical axis range. A common use is MINIMUM=0.0 to set 0 as the base value. If data go outside this range their values are clipped: the plot range is not extended to cover them. When this happens, any out-of-range value is replaced by the boundary value.

**picture=**picture clause for values [shortest that works]

The PICTURE option sets the representation for the numeric labels on the vertical axis. For instance, PICTURE="*.##" will show the numbers with two digits right of the decimal. By default, GRAPH chooses the shortest representation that can show all values accurately. See DISPLAY for more on picture clauses.

**[axis]/noaxis**

With NOAXIS, GRAPH will not draw the horizontal axis line even if the zero value lies within the range of values in series.

**extend/[noextend]**

**vgrid=**VECTOR of grid line values for the vertical axis [unused]

Normally RATS marks the vertical axis with small tick marks outside of the graph. The EXTEND option draws dotted lines across the full width of the graph. (RATS ignores EXTEND when you are doing a two-scale graph). VGRID accepts a VECTOR of vertical-axis values at which horizontal lines will be drawn going across the graph.

**vticks=**maximum vertical ticks  [9]

VTICKS sets the maximum number of labeled tick marks on the vertical scale.
Two-Scale (Overlay) Graph Options

Two-scale graphs are created with a single `GRAPH` instruction, using `OVERLAY` and the related options. See Section 3.8 in the User’s Guide for more information.

- `overlay=line/polygonal/bar/stacked/overlapping/vertical/step/symbol/midpolygon/fan/dots/spike`
  OVERLAY makes the graph a two-scale graph, and specifies the style to use for the overlaying series (right-side scale series). See `STYLE` for details on the styles.

- `ovsamescale/[noovsamescale]`
  You can use `OVSAAMESCALE` to force both the regular and the overlay series to share a common scale. They will just be shown in different styles.

- `ovcount=Number of series for right-side scale [1]`
  The last `Number` series listed on the supplementary cards are graphed using the right-side scale. The other series are graphed using the left-side scale.

- `omax=Maximum value for right-side scale [largest value]`
- `omin=Minimum value for right-side scale [smallest value]`
- `ovlabel=Vertical scale label (in quotes: "...") or STRING) [none]`
  `OMAX` and `OMIN` allow you to set the maximum and minimum values, respectively, for the right-side scale. These function like the `MAX` and `MIN` options (which control the left-side scale when doing a two-scale graph). `OVLABEL` allows you to supply a label for the right-side scale.

- `[ovkey]/noovkey`
  You can use `NOOVKEY` to eliminate the key for the overlay series, if the meaning is either obvious, or provided using the labels.

- `ovrange=fraction of scale overlap [1.0]`
  `OVANGE` allows you to offset the left and right scales vertically, so the `STYLE` side and the `OVERLAY` side are each plotted using only a portion of the available vertical space. With the default value of 1.0, both scales will overlap fully (i.e. share the same vertical space). With `OVANGE=0.5`, each scale will use only half of the total space.

Key Options

- `key=[none]/upleft/upright/loleft/loright/above/below/left/right/attached`
  `KEY` controls the placement of the key for the graph. The choices are:
  - NONE: No key
  - UPLEFT: Key in upper left corner, inside the graph box
  - UPRIGHT: upper right corner, inside
  - LOLEFT: lower left corner, inside
**LORIGHT** lower right corner, inside
**ABOVE** centered above the graph (and any HEADER and SUBHEADER).
**BELOW** centered below the graph, and below any X-axis labeling
**LEFT** left side, centered vertically, outside the graph and Y-axis labeling
**RIGHT** right side, centered vertically, outside the graph and Y-axis labeling
**ATTACHED** can be used only with LINES and SYMBOLS styles, puts the labels near the lines, at positions where the association of a line with the labels is as unambiguous as possible.

**klabel=VECTOR of STRINGS for key labels**
By default, RATS labels the KEY with the names of the data series. Use the KLABEL option if you want to supply your own labels. You can create the VECTOR of STRINGS ahead of time, or enter it directly using the ||..|| matrix notation (see Section 4.6 in the User’s Guide). Make sure that the order of the labels in the VECTOR matches the order of the supplementary cards. In the KLABEL strings, you can use \ to produce text which has more than one line.

**[kbox]/nokbox**
This controls whether or not a box (border) is drawn around the key.

**kheight=height of key box (between 0 and 1) [unused]**
**kwidth=width of key box (between 0 and 1) [unused]**
These two options allow you to control the size and proportion of the key box. By default, RATS tries to find the most efficient arrangement for the key, given the number of series in the key, its position, the setting of the GRPARM KEYLABELING parameter, and so on.

You can use these two options to override the default arrangement. KHEIGHT and KWIDTH must be used together—they have no effect if one is used without the other. They allow you to specify the height and width of the key box as a fraction of the overall height and width of the graph.

**[ksample]/nksample**
NOKSAMPLE eliminates the sample line, color or fill pattern from the key, leaving only the labels.

**Notes**
RATS leaves missing values out of the graph. For a line graph, a dotted line connects the points on either side of the missing data point.

If you are graphing a large number of series on a single graph, you may find the LIST and CARDS utility (see the description of LIST) helpful in simplifying the supplementary cards.
Examples

This generates the stacked-bar graph shown below:

```plaintext
cal(q) 1946:1
all 2002:6
open data haversample.rat
data(format=rats) / cd cn cs
labels cd cn cs ; # "Durable" "Non-Durable" "Services"
smpl 1993:1 *
graph(style=stacked,header="Major Components of Consumption", $
   key=below,patterns) 3
   # cd ; # cn ; # cs
```

![Major Components of Consumption](image)

Lines, Fill Patterns, and Symbols

RATS provides seven different line patterns, ten different “fill” patterns (for bar-type graphs), and twelve symbols to choose from. These patterns and symbols are shown on the next page. The line and fill patterns are used when displaying graphs in black and white mode (by using the PATTERNS option or the toolbar icon), or when outputting to a black and white device. The symbol shapes are used for both color and black and white output.

The numbers in the left-hand column are the values for the representation parameter that select the associated pattern or symbol when using the default graph style sheet. Note that the numbering may be different if you are using a customized style sheet, which also allows you to set attributes such as color (or gray scale level), line thickness, whether or not the symbols are filled in, and more (see Section 3.5 in the User’s Guide for details).
Lines, Fill Patterns, and Symbols Samples

1 | ______ |
2 | .......... |
3 | — — — |
4 | — — — |
5 | — — — |
6 | — — — |
7 | · · · · |
8 | " " " " |
9 | " " " " |
10 | " " " " |
11 |
12 |
GROUP — Building General Simultaneous Models

**GROUP** constructs a model for simulation. The model can include any combination of **EQUATIONS** and formulas (**FRMLs**). There is no limit on the number of equations and formulas you can include in a model.

```
GROUP( options )  model  list of formula>>result fields
```

**Parameters**

- **model**
  The name of the model. Use this in the **MODEL** option on **IMPULSE, ERRORS, ESTIMATE, FORECAST, HISTORY, NLSYSTEM, SIMULATE, STEPS, SUR, or THEIL**.

- **formula>>result**
  The name of the formula or equation. The **result** part of each of these fields is optional: it allows you to supply a series into which the forecasts are to be placed.

**Options**

- **cv=SYMMETRIC covariance matrix of residuals**
  You only need this option if you are going to do random simulations of the model and want to take into account the covariances among the residuals. The matrix should have dimensions of \(N\times N\), where \(N\) is the number of structural equations (non-identities) in the model. (The older **VCV** is a synonym for **CV**).

**Setting Up Your Model**

Your model will consist of estimated structural equations and identities. You should list the identities last if you’re going to use **SIMULATE**. No variable should be the dependent variable in more than one formula. Although the identities \(Y=C+I+G\) and \(C=Y–I–G\) are equivalent, you cannot use the second form if there is a structural equation for \(C\) in the system.

When you have non-linearities such as mixed logs and levels, you must also add definitional formulas for these transformations. These formulas usually mimic the **SET** instructions used to define the variables in the first place. For instance, in the model on pages 390-399 of Pindyck and Rubinfeld (1998) (and **User’s Guide** Example 11.1) every equation is linear in its variables. However, the model uses many transformations, so you would need to close the model with the following identities

```
frml(identity)  gnpid gnp = cons+invest+gvt
set ydiff = gnp-gnp{1}
frml(identity)  ydiffid ydiff = gnp-gnp{1}
set rsum = rate+rate{1}
frml(identity)  rsumid rsum = rate+rate{1}
group prsmall conseq>>f_cons  investeq>>f_invest $
  rateeq>>f_rate  gnpid>>f_gnp  rsumid ydiffid
```
GRPARM — Graphics Parameters

GRPARM lets you choose the font, size, and style used for the various graph labels. It is also used to load graph style sheets into memory.

```
grparm( options ) pairs of: labeltype  newsize
```

Parameters

The GRPARM parameters specify the label type or types to which the GRPARM command will apply. You can change one or more of these with a single GRPARM instruction. Separate the items with one or more spaces. You can truncate any of the parameter names to three or more characters, for instance, KEY is sufficient for KEYLABELING. The choices are:

- **header**  Font size for the header. Default = 24 points, bold.
- **subheader**  Font size for the subheader. Default = 18 points, italic.
- **footer**  Font size for the subheader. Default = 18 points.
- **hlabel**  Font size for the HLABEL. Default = 20 points, bold. Applies to HLABEL option on GRAPH, SCATTER, and SPGRAPH.
- **vlabel**  Font size for the VLABEL. Default = 20 points, bold. Applies to VLABEL option on GRAPH, SCATTER, and SPGRAPH.
- **axislabeling**  Font size for value labels on either axis. Default = 18 points.
- **monthlabels**  Font size for month titles and day numbers (AXISLABELING size applies to years). Default = 14 points.
- **keylabeling**  Font size for titles in key. This also determines the size of the key samples. Default = 14 points.
- **matrixlabeling**  Font size for XLABELS and YLABELS on SPGRAPH. Default = 18 points, bold.

If you don’t want to change the size of a label (that is, if you only want to change its font or style of the label), use * for the size parameter. For example:

```
grparm(nobold) vlabel *
```
Options

**italics/[noitalics]**
**bold/[nobold]**
**font="font name"**

BOLD and ITALICS allow you to choose bold and italics styles for the label types listed on the GRParm instruction. The FONT option allows you to specify the font used for the specified labels. See “Using Fonts” in Section 3.6 of the User’s Guide for details.

**portrait/[noportrait]**

The PORTRAIT option lets you rotate the graph by 90 degrees. In combination with the Portrait/Landscape choices in the Page Setup dialog box (Windows/Macintosh versions) or the Portrait and Landscape PostScript exporting choices (all versions), this provides additional choices for orienting the graph. See Chapter 3 in the User’s Guide for details.

**patterns/[nopatterns]**

Use PATTERNS if you want subsequent graphs to be drawn in black and white rather than in color by default. For black and white graphs, RATS uses different patterns to distinguish one series from another.

**recall=VECT[INTEGER] saved by %GRPARM function**

The %GRPARM() function returns a VECTOR containing the current set of font size and style definitions. If you have saved this information into a vector earlier in the session, you can revert back to those settings by using the RECALL option.

For example, if you’ve done:

```plaintext
compute fontstyles = %gr parm()
```

You can revert to those settings by doing:

```plaintext
grp parm(recall=fontstyles)
```

**import=unit from which to input definitions of styles**

This option allows you to load custom definitions for line, fill pattern, and symbol representations from a graph style sheet. See Graph Styles below for details.
Graph Style Sheets

As discussed in Chapter 3, RATS allows you to customize many aspects of the line styles, fill patterns, and symbols used in graphs by editing the definitions in a graph style sheet file. The \texttt{GRParm} instruction is used to load the definitions saved in a style sheet into memory.

You first use an \texttt{OPEN} instruction to open the style sheet, and then use \texttt{GRParm} with the \texttt{IMPORT} option to load the styles. For example, if you have a style sheet called \texttt{GRAPHSTYLES.TXT}, you might do:

\begin{verbatim}
open style graphstyles.txt
grparm(import=style)
\end{verbatim}

See Section 3.5 for more information on style sheets.

Notes on Font Sizing

RATS automatically scales labels to fit the overall graph size, with default sizes based on a full-page (8"×10") graph. If you take a full sized graph and make it smaller, the fonts will be reduced proportionally. If you want a label to be larger or smaller than it appears, set a new size relative to the default size.

For example, suppose you do an \texttt{SPGRAPH} with four graphs on a page, where each graph has its own header. RATS will automatically scale the header down from 18 points to an apparent size of about 8 or 9 points. To select a slightly larger header, you might try \texttt{GRParm HEADER 20} (larger relative to the 18 point default), not \texttt{GRParm HEADER 12}, which would produce smaller headers.

Examples

\begin{verbatim}
grparm(bold) keylabeling 18
grparm monthlabels 12
\end{verbatim}

The first sets the key labels to 18 point bold; the second increases the month and day labels to 12 point.

\begin{verbatim}
grparm axislabel 18 header 30
grparm(italics) subheader 22
graph(extend, $\$\$
header=\"Canadian - US Exchange Rate\",sub=\"Can \$/US \$\") 1
# canusx 1978:1 2002:12
\end{verbatim}

This increases the sizes of the axis, header and subheader labels.
GRText

GRTEXT — Adding Text to Graphs

GRTEXT adds text to a GRAPH, SCATTER, or GCONTOUR plot at a location inside the graph boundary. GRTEXT can only be used in an SPGRAPH block.

\texttt{grtext( options ) \textquotedblright string}\n
Usage
The basic procedure is:

1. Use SPGRAPH to initiate the special graph.
2. Use GRAPH or SCATTER to draw the graph.
3. Use one or more GRTEXT instructions to add text to the graph created in step 2. Use one GRTEXT for each string you want to add.
4. If you are putting multiple graphs on the page (by using the HFIELDS and VFIELDS options on SPGRAPH), repeat steps 2 and 3 as necessary to draw the other graphs and, if desired, add text to those graphs.
5. Issue the SPGRAPH(DONE) instruction to complete the special graph.

Parameters

"string" The string of text you want to add to the graph. This can be a string of text enclosed in quotes, or a STRING or LABEL type variable. A \ can be used to show where line breaks are wanted.

Options

position=upleft/upright/loleft/zoright/rightmargin/bottommargin/leftmargin/topmargin

ENTRY=entry number or date for x-axis position (used with GRAPH)
x=x-axis value for x-axis position (used with SCATTER or GCONTOUR)
y=y-axis value for y-axis position (with GRAPH, SCATTER, or GCONTOUR)

These options control the position of the text. The POSITION option puts the text in one of the corners of the graph box (upper left, upper right, lower left, lower right), or in one of the margins of the graph. To use any of the “margin” options, you need to do the GRTEXT before the graphing instruction.

ENTRY and Y (for GRAPH) or X and Y (for SCATTER or GCONTOUR) allow you to place the text at a specific location to annotate some feature. The ALIGNMENT and VALIGNMENT options described below control placement of the string relative to the location you provide.
If you are doing a **GRAPH**, you use the **ENTRY** option to set the *horizontal* position as a date or entry number, and the **Y** option to set the *vertical* position of the strings, within the Y-axis range. If you are doing a **SCATTER** or **GCONTOUR**, you use the **X** option to specify the *horizontal* position within the X-axis range, and the **Y** option to specify the *vertical* position within the Y-axis range.

alignment=[centered]/right/left  
valignment=[centered]/top/bottom  
ALIGNMENT determines whether the text should be centered at the position, or right- or left-justified at the starting position, and VALIGNMENT determines whether the starting position will give the vertical center, the top or bottom.

font="font name"  
size=relative size of type in points  
These select the typeface and size of the string. The default point size is 14 points, based on a full-page graph. Fonts are automatically scaled for smaller graphs. See “Using Fonts” in Section 3.6 of the *User’s Guide* for details.

bold/[nobold]  
italics/[noitalics]  
These display the string in bold and/or italic type.

box/[nobox]  
Puts a box around the text.

**Examples**

The following computes the histogram of a series, graphs it as a bar graph, overlaying it with the normal density with the sample mean and variance, and displaying the skewness and kurtosis in the upper left corner.

density(maxgrid=25,type=histogram) x / fx dx  
stats(noprint) x  
set nx = 1.0/sqrt(%variance)*%density((fx-%mean)/sqrt(%variance))  
spgraph  
scatter(style=bargraph,overlay=line,ovsamescale) 2  
# fx dx  
# fx nx  
display(store=s) "Skewness" %skewness $  
"\\Excess Kurtosis" %kurtosis  
grtext(position=upleft) s  
spgraph(done)

In the example on the next page, we graph a series and add some text showing the maximum and minimum values of the series. We use the **EXTREMUM** instruction to
calculate the maximum (%MAXIMUM) and minimum (%MINIMUM) values of the series and the entry numbers (%MAXENT and %MINENT) at which these values occur.

We don’t want the text to overwrite any of the graph line, so we’ll put the labels a little above and below the maximum and minimum values. We’ve also chosen to draw the labels left-justified, just to the right of the maximum and minimum points.

cal(q) 1980:1
all 2003:1
set(first=0) x = .05*t + .5*x{1} + %ran(2.0)
extrumum(noprint) x
  * Calculate positions for the labels:
    compute maxentry = %maxent + 1
    compute maxval = %maximum + .3
    compute minentry = %minent + 1
    compute minval = %minimum - .3
  * Construct the strings:
    disp(store=maxstring) "Maximum = " #.## %maximum
    disp(store=minstring) "Minimum = " #.## %minimum
    spgraph
      graph(max=(maxval+1),min=(minval-1)) 1
      # x
      * Add the text at the specified positions:
        grtext(align=left,valign=bottom,entry=maxentry,y=maxval) $ maxstring
        grtext(align=left,valign=top,entry=minentry,y=minval) $ minstring
        spgraph(done)
GSet — Setting Series of Arrays

**GSET** can be used to set entries of **SERIES** of matrices or any other data type. While almost anything that can be done with **GSET** could also be done using **VECTORS** of the desired data types, using the **SERIES** form allows subscripting to match up with that used by the data itself. It also allows you to use lag notation.

```
gset series start end = function(T)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>series</strong></td>
<td>The series of arrays to create or set with this instruction.</td>
</tr>
<tr>
<td><strong>start</strong>, <strong>end</strong></td>
<td>The range of entries to set. If you have not set a <strong>SMPL</strong>, this defaults to the <strong>ALLOCATE</strong> range.</td>
</tr>
<tr>
<td><strong>function(T)</strong></td>
<td>The function of the entry number T, which gives the value for entry T of <strong>series</strong>. This should evaluate to the type of the elements of the series. There should be at least one blank on either side of the = which comes between the other parameters and the function. The function can actually include multiple expressions, separated by commas. The series will be set to the value returned by the final expression in the list.</td>
</tr>
</tbody>
</table>

**Option**

**smpl=SMPL series or formula** *(User’s Guide, Section 5.2)*

You can supply a series or a formula that can be evaluated across entry numbers. Periods for which the series or formula is zero or “false” will be skipped, while entries that are non-zero or “true” will be included in the operation.

**Examples**

```
declare series[sym] uu
gset uu gstart gend = %zeros(2,2)
```

creates **UU** as a set of series of $2 \times 2$ symmetric matrices, all initialized to zeros.

```
dec series[vect] xu
gset xu regstart regend = %eqnxvector(0,t)*u
set ssqtest regstart regend = %qform(xxs,xu)
```

**XXS** is a previously defined **SERIES[Symmetric]**, and **U** is a series (of residuals). This creates **XU** as a **SERIES** of vectors, which are set equal to the $X_i u_i$ from the last regression. (%eqnxvector(0,t) is $X_t$). **SSQTEST** is then the series of values for $(X_i u_i)' XXS_i (X_i u_i)$. 

Halt

HALT — Stopping Execution From Within Compiled Sections

HALT stops execution of the program. You must use HALT, rather than END, if you want to halt execution from within a compiled section.

```
halt message
```

Parameters

- **message** (Optional) You can provide a string of up to 255 characters. RATS will print the message

  Normal Completion. Halt at message

  in the summary at the end of the program output if this HALT is executed. This can be helpful if you use multiple HALT’s in a program.

Example

```
loop
    menu "What Next?"
    choice "Enter Data"
        source indata.src
    choice "Do Forecasts"
        source forecast.src
    choice "Quit"
        halt
    end menu
end loop
```

this loops until the user chooses “Quit” from the menu.

See Also . . .

- **END** Ends a RATS program or an interactive session.
- **BREAK** Breaks control out of a loop.
- **RETURN** Returns control from a procedure or other compiled section.
HISTORY — Historical Decompositions

**HISTORY** decomposes the historical values of a set of time series into a base projection and the accumulated effects of current and past innovations. See the “Technical Description” later in this section for details.

**HISTORY** can only work for linear models as it relies upon linearity properties of moving average representations.

```plaintext
history(options) equations
# equation series newstart column (one for each equation if no MODEL)
```

**Wizard**

The VAR (Forecast/Analyze) wizard on the Statistics menu provides an easy, dialog-driven interface for computing historical decompositions.

**Parameters**

- **equations**
  
  Number of equations in the system. You can use * to skip this parameter if you are using the MODEL option.

  The *period, start* and *VCV Matrix* parameters used in older versions have been deprecated. These are replaced by the *STEPS, FROM, and CV* options, respectively. Version 7 will still recognize these older parameters, but we recommend that you use the new options in any new programs.

**Supplementary Cards (if you don’t use the MODEL option)**

If you are not using the MODEL option, supply one supplementary card for each equation in the system.

- **equation**
  
  The name or number of the equation. If this isn’t an identity, you must either:

  - save its residuals when you estimate the equation, or
  - associate a residuals series with it using **ASSOCIATE**.

- **series**
  
  For a system with *N* estimated equations, this is the first in a block of *N*+1 consecutive series into which **HISTORY** puts the decomposition of the dependent variable for *equation*. The base projection term goes into *series*, and the effects of the *N* components of the residuals go into the next *N* series. These series must already exist, so you will have to use **ALLOCATE**, **SCRATCH** or **DECLARE RECT[SERIES]** to create them.

  *Note: The RESULTS option is an easier way to handle the results.*
History

newstart (Optional) The first entry for the created series. By default, it is the same as start.

column (Optional) The column (row) of the CV matrix which corresponds to equation. This is necessary only if you want to use orthogonalized innovations. The default is the supplementary card position (column 1 corresponds to the first card, and so on).

Options

model=model name
This is an alternative way to specify the system of equations. This cannot include any FRML’s, as it must be fully linear. Typically, this is a model defined for a VAR with SYSTEM(MODEL=...) or one derived from that.

from=starting period of the forecast interval
to=ending period of the forecast interval
steps=number of forecast steps to compute
These determine the periods for which the historical decomposition will be computed. If you have set a SMPL, these default to that range. Otherwise, FROM and TO default to the beginning and end of the most recent estimation range, respectively. If you want something other than the defaults, you can use:

- FROM and TO to set the starting and ending periods for the decomposition, or
- FROM and STEPS to set the starting date and number of steps (periods)

add/ [noadd]
If you use ADD, HISTORY adds the projection series in each block to the other series in that block. This makes it easier to create meaningful graphs.

results=RECTANGULAR[SERIES] for result series
This provides a RECTANGULAR array of SERIES which will be filled with the results. Its dimensions will be $(N + 1) \times N$. Each column of series shows the decomposition of a single series. The first element (that is, the first series) in the column is the base projection. The second is the cumulated effect of the residuals to the first equation, the third shows the effect of the second set of residuals, etc. You do not need to DECLARE or DIMENSION this array before using HISTORY.

cv=SYMmETRIC covariance matrix of residuals [%sigma]
factor=RECTANGULAR decomposition matrix
Use CV when you want orthogonalized innovations using the Choleski factorization (User’s Guide, Section 10.5). If you are using MODEL and omit this option, RATS defaults to using the estimated covariance matrix for the MODEL (stored in %SIGMA). As an alternative, you can use FACTOR to supply your own factorization of the covariance matrix, such as the factor matrix produced by a CVMODEL instruction. (User’s Guide, Section 10.5.4). This option was called DECOMP in earlier versions. DECOMP is still recognized as a synonym for FACTOR.
window="Title of window"

If you use the WINDOW option, a (read-only) spreadsheet window is created with the indicated title and displayed on the screen. This will display $N$ blocks of $N+2$ columns. These columns show the actual value, the forecast and the accumulated effects of each of the series of residuals. You can export information from this window to a file in a variety of formats using the File-Export... operation.

labels=VECTOR[STRINGS] to label shocks

You can use the LABELS option to assign specific labels to the shocks if the standard practice of labeling them with the corresponding dependent variable would be misleading. This matters only if you are using both the WINDOW option and the DECOMP option.

Technical Description

The historical decomposition is based upon the following partition of the moving average representation

$$y_{T+j} = \sum_{s=0}^{j-1} \Psi_s u_{T+j-s} + \left[ X_{T+j}\beta + \sum_{s=j}^{\infty} \Psi_s u_{T+j-s} \right]$$

The first sum represents that part of $y_{T+j}$ due to innovations in periods $T+1$ to $T+j$. The second is the forecast of $y_{T+j}$ based on information available at time $T$.

If $u$ has $N$ components, the historical decomposition of $y_{T+j}$ has $N+1$ parts:

- The forecast of $y_{T+j}$ based upon information at time $T$ (the term in brackets).
- For each of the $N$ components of $u$, the part of the first term that is due to the time path of that component.

This is the order in which the resulting series are organized: if you use the RESULTS option, the first row in each column gives the base forecasts and the remaining rows are the components. If you use supplementary cards, the base forecast is put into the series given by the series field and the effects of the components go into the next $N$ series.

Comments

If you use the actual estimated model and its residuals, the components of the decomposition will add up to the observed data.

Using the ADD option superimposes the innovation components on the base projection: the influential variables will tend to create movements from the base that are much larger than the less important variables.
Examples

```
system(model=canmodel)
variables usargdps canusxsr cancd90d canmls canrgdps cancpinf
lags 1 to 4
det constant
end(system)
*
compute hstart=2000:1
compute hend =2006:4
*
estimate
history(model=canmodel,add,results=history,from=hstart,to=hend)
print / history
```

This computes the historical decomposition from the first quarter of 2000 through the fourth quarter of 2006 for the model CANMODEL. The results are stored into a RECTANGULAR array of SERIES called HISTORY.

```
history(model=canmodel,add,results=history,window="History",$
   labels=||"f1","f2","r1","r2","n1","n2"||,factor=bfactor,}$
   from=hstart,to=hend)
```

This modifies the earlier example by using a non-standard factorization, with relabeled shocks (financial shocks 1 and 2, real 1 and 2 and nominal 1 and 2). In addition to being stored into an array of series, the results are also displayed in a window named “History.”
IF and ELSE — Conditional Execution

The **IF** and **ELSE** instructions provide a means of executing an instruction or a group of instructions only under certain conditions. They operate very much like IF and ELSE statements in standard programming languages. Note well: before you use **IF** and **ELSE**, check out the `%IF` function. Some programs, such as SAS, use IF and ELSE in transformations to do what the `%IF` function does in RATS. See the first example on the next page.

```
if  condition 1
   statement or block of statements (executed if condition 1 is true)
else if  condition n
   statement or block of statements (executed if condition n is true and no earlier conditions in the current IF block were true)
else
   statement or block of statements (executed if no conditions were true)
end if (omit if the IF is nested inside another compiled section—see below)
```

**Parameters**

condition **IF** and **ELSE IF** statements evaluate a condition expression. This can be either an integer- or real-valued expression. The condition is true if it has a non-zero value and false if it has value zero. Usually, you construct these using logical and relational operators.

**Description**

An **IF-ELSE** block has the following structure:

1. An **IF** instruction.
2. (Optionally) One or more **ELSE IF** instructions.
3. (Optionally) An **ELSE** instruction.

RATS handles **IF**, **ELSE IF**, and **ELSE** instructions as follows:

- RATS executes the block of statements following the first **IF** or **ELSE IF** whose condition is true.
- If all of the **IF** or **ELSE IF** conditions are false, and you have an **ELSE**, RATS will execute the block of statements following **ELSE**.
- If all of the conditions are false, and you do not have an **ELSE** statement, RATS drops to the first instruction following the last **ELSE IF** block.
If, Else

After RATS executes the instruction(s) associated with a true **IF** or **ELSE** condition, it will ignore any remaining **ELSE IF** conditions and drop down to the first instruction following the last block of statements in the **IF-ELSE** block.

If the **IF** instruction is the initial instruction in a compiled section, you must include a matching **END** at the end of section, usually after the last clause in the **IF-ELSE** block.

**Blocks of Statements**

You can follow each **IF**, **ELSE IF**, or **ELSE** statement with a single instruction, a group of instructions enclosed in braces (the `{` and `}` symbols), a **DO**, **DOFOR**, **WHILE**, **UNTIL** or **LOOP** loop, or another “nested” **IF-ELSE** block. When using nested **IF**'s, please note the information on the next page regarding ambiguous **ELSE**'s. If you have any doubt about whether you have set up a block properly, just enclose the statements that you want executed in braces (`{` and `}`).

**Examples**

First, here is an example of the use of the `%if` function in a transformation. The series **PATCH** is set equal to **XS** if **XS** isn’t a missing value, or equal to **XV** otherwise. This is **not** done with **IF** and **ELSE** instructions, as the **IF** and **ELSE** would choose one of the two **SETs** to execute for **all** entries.

The right way:  
\[
\text{set patch} = \%if(\%valid(xs),xs,xv)
\]

The wrong way:  
\[
\text{if } \%\text{valid}(xs) \\
\text{set patch} = xs \\
\text{else} \\
\text{set patch} = xv
\]

\[
\text{if } \text{standardize}<>0 \\
\text{diff(standardize)} \text{ depvar}
\]

This executes the **DIFF** instruction only if **STANDARDIZE** is non-zero.

\[
\text{if } \text{ichoice}=1 \\
\text{window(type=flat,width=width) iser 1 nords 5 } [a] \\
\text{else if } \text{ichoice}=2 \\
\text{window(type=tent,width=width) iser 1 nords 5 } [b] \\
\text{else} \\
\text{window(type=flat,mask=imask,width=width) iser 1 nords 5 } [c]
\]

This executes [a] if **ICHOICE** is 1, [b] if **ICHOICE** is 2, and [c] if **ICHOICE** is neither of these values.
nlls(frml=ces) q
if %cvcrit<0.001.and.%cvcrit>0.00001
   nlls(frml=ces) q
end if

This does one set of iterations for non-linear least squares, and checks to see if it has almost converged (true if %CVCRIT is less than 0.001 but is greater than 0.00001). If so, it does a new set of iterations. The IF, NLLS and END IF instructions make up a complete compiled section.

Ambiguous ELSE Statements

You need to be careful when writing nested IF instructions. Consider these two examples:

```plaintext
if condition 1
   if condition 2 ; statement 1
   else ; statement 2

if condition 1
   if condition 2 ; statement 1
else ; statement 2
```

While indenting makes it easier for you to read and interpret code, the leading spaces or tabs are ignored by RATS itself. So, both examples are executed the same way, even though the ELSE instructions are intended to apply to different IF instructions.

RATS resolves the ambiguity by having an ELSE apply to the IF which immediately precedes it; that is, RATS uses the interpretation implied by the first example.

If you want to use the bottom form instead, you must put { and } lines around the second IF; this tells RATS that it should only execute those instructions within this block if condition 1 is true:

```plaintext
if condition 1
   { 
      if condition 2 ; statement 1
   }
else ; statement 2
```

See Also . . .

UG, Section 16.1 The RATS Compiler.
**IMPULSE — Impulse Response Functions**

`IMPULSE` generates the responses of a system of equations to a specified set of shocks. The impulse response functions are the dynamic response of each endogenous variable to a shock to the system. The main use of `IMPULSE` is to generate a moving average representation (MAR) of a vector autoregression.

`IMPULSE` can only work for linear models as it relies upon linearity properties of moving average representations.

```
impulse(options) equations
   # equation response newstart column (one for each equation if no MODEL)
   # first period shocks (only with INPUT option)
   # list of path series (only with PATHS option)
```

**Wizard**

The `VAR (Forecast/Analyze)` wizard on the `Statistics` menu provides an easy, dialog-driven interface for analyzing VAR models, including computing impulse responses.

**Parameters**

- `equations` Number of equations in the system. This can be * if you are using the `MODEL` option.

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. Version 7 will still recognize the older parameters, but we recommend that you use the new options in any new programs.

**Supplementary Cards (if you don’t use the MODEL option)**

If you aren’t using the `MODEL` option, supply one supplementary card for each equation in the system. The order of listing affects the decomposition order. You must list supplementary cards for identities last.

- `equation` The equation name or number.
- `response` (Optional) The series for the response of the dependent variable of `equation`. RATS ignores `response` if you use a single `IMPULSE` instruction to print responses to each of the variables in turn (see `shock to` above). You can get the whole matrix of responses by using the `RESULTS` option.
- `newstart` (Optional) The starting entry for the series `response`. The default is 1.
(Optional) The column of the CV matrix which corresponds to this equation. By default: this is just the supplementary card position: that is, first supplementary card corresponds to column one, second card to column two, etc.

You only need to use the other two types of supplementary cards if you use the INPUT or PATHS options. See below for an explanation of these options.

Options

model=model name
This is an alternative way to specify the system of equations. This cannot include any FRML’s, as it must be fully linear. Typically, this is a model defined for a VAR with SYSTEM(MODEL=...) or one derived from that.

steps=number of steps to compute
This sets the number of steps (periods) for which you wish to compute responses. If you have set a SMPL, this defaults to the number of steps implied by it. Otherwise, you must supply a value, using either the steps parameter or the STEPS option.

column=component to shock [shock all components]
By default, RATS computes a complete set of responses for shocks to each of the equations in the model. Use the COLUMN option if you want to compute responses for a shock to a particular column of the covariance or factor matrix.

cv=SYMMETRIC covariance matrix of residuals [%sigma]

factor=RECTANGULAR decomposition matrix
Use CV when you want orthogonalized innovations using the Choleski factorization (User’s Guide, Section 10.5). If you are using MODEL and omit this option, RATS defaults to using the estimated covariance matrix for the MODEL (stored in %SIGMA).

As an alternative, you can use FACTOR to supply your own factorization of the covariance matrix, such as the factor matrix produced by a CVMODEL instruction. (User’s Guide, Section 10.5.4). This option was called DECOMP in earlier versions. DECOMP is still recognized as a synonym for FACTOR.

results=RECTANGULAR[SERIES] for result series
This provides a RECTANGULAR array of SERIES which will be filled with the results. This will typically be used when you are getting the full moving average representation of a VAR (the default behavior), when it will have dimensions $N \times N$. The responses to the shock in innovation $i$ will be column $i$ in the matrix. If you are requesting responses to a single shock, the matrix will have dimensions $N \times 1$. Each series created will be filled from entries 1 to steps.
[print]/noprint

window="Title of window"

Use NOPRINT to suppress the display of the responses to the output window or file. Use the WINDOW option if you want to display the output in a (read-only) spreadsheet window, which will have the title you supply. The output will be organized as separate sub-tables for each variable shocked. You can export information from this window to a file in a variety of formats using the File–Export... operation.

labels=VECTOR[STRINGS] to label shocks

You can use the LABELS option to assign specific labels to the shocks if the standard practice of labelling them with the corresponding dependent variable would be misleading. This matters only if you are using the DECOMP option.

input/[noinput]

shocks=VECTOR for first period shocks

You can use one of these options to input general first period shocks. With INPUT, you supply the shocks on a supplementary card of the second form; with SHOCKS, the indicated VECTOR provides the shocks. See the next page for details.

matrix=RECTANGULAR array of shock paths

paths/[nopaths]

start=Start Entry for PATHS series [1]

You can use either MATRIX or PATHS to input the paths of shocks over more than one period.

With MATRIX, you set up a RECTANGULAR array to provide the paths of shocks to the equations. The columns of the array should match the order of the equations, that is, the shocks to the first equation should be in the first column. The number of rows does not have to be equal to steps. The shocks will be set to zero for any steps beyond those supplied by the array.

With PATHS, you supply a list of series on a supplementary card. These series provide the paths of the shocks. You must define these series for steps entries beginning with the START option entry. Use * on the supplementary card for any equation whose shocks are to be zero for the entire period.

Notes

IMPULSE can only be applied to linear models (sets of equations). In a non-linear system, the multipliers depend upon the initial point around which you are expanding, so you can’t use IMPULSE for these. Instead, run two FORECAST instructions with changes to the exogenous variables. See User’s Guide, Section 11.3, for more on this.
Technical Details and Choices for Providing Shocks

In the moving average representation,

\[ y_t = X_t\beta + \sum_{s=0}^{\infty} \Psi_s u_{t-s} \]

the response at \( t=k \) to an initial shock \( z \) in the \( u \) process is \( \Psi_k z \). For instance, the response at step \( k \) to a unit shock in equation \( i \) at \( t=0 \) is just the \( i \)th column of the \( \Psi_k \) matrix. \textbf{IMPULSE} allows the shock to the system to take one of several forms:

1. First period shock to a single component of \( u \). All components of \( z \) are zero except the component shocked. It is set equal to the standard error of estimate of the equation. If using supplementary cards to list equations, you can implement this by setting \texttt{COLUMN} to the component you want shocked and omitting the \texttt{CV} option.

If you are using the \texttt{MODEL} option rather than supplementary cards, RATS defaults to method (2) described below (a Choleski factorization of the estimated covariance matrix). To get the type of shock described above when using \texttt{MODEL}, you would need to explicitly use one of the other options (\texttt{FACTOR}, \texttt{INPUT}, or \texttt{SHOCKS}).

2. First period shock which is a unit shock to an orthogonalized innovation of the process. If \( \text{Var}(u) = \Sigma = GG' \), then \( u = Gv \) where \( \text{Var}(v) = I \). A shock of unit size to the \( i \)th component of \( v \) is a \( z \) vector which is the \( i \)th column of \( G \). Implement by setting \texttt{COLUMN} to the component you want shocked.

As noted above, when using the \texttt{MODEL} option, RATS defaults to doing a Choleski factorization using the estimated covariance matrix associated with the model. Use the \texttt{CV} option for a standard Choleski factorization of a user-defined covariance matrix, or the \texttt{FACTOR} option for other factorizations.

3. General first period shocks (\( z \) vector).

Implement using either the \texttt{INPUT} option (including an additional supplementary card with the shocks) or the \texttt{SHOCKS} option.

4. Paths of shocks to one or more equations.

Implement using either the \texttt{PATHS} option (including an additional supplementary card listing the series of shocks) or the \texttt{MATRIX} option.
LIST and CARDS

If you’re using supplementary cards to describe the model rather than a MODEL, these often follow a simple pattern, for instance,

```r
impulse(steps=20,column=1,cv=v) 4
# 1 resp(1) 1 1
# 2 resp(2) 1 2
# 3 resp(3) 1 3
# 4 resp(4) 1 4
```

Each card has the form `# ieqn resp(ieqn) 1 ieqn`.

You may find the LIST and CARDS utility (see the description of LIST) to be very useful for handling batches of supplementary cards like this. You could replace the above example with the following:

```r
list ieqn = 1 2 3 4
impulse(steps=20,column=1,cv=v) 4
cards ieqn resp(ieqn) 1 ieqn
```

Examples

```r
impulse(model=canmodel,steps=20,results=impulses)
```

computes twenty steps of the impulse responses to all the orthogonalized shocks to the equations in CANMODEL. IMPULSES(i,j) is a series defined from entries 1 to 20 which has the response of the i-th dependent variable to a shock in the j-th.

```r
impulse(model=canmodel,steps=24,col=3,window="Shock to Rate")
```

shocks the third orthogonalized component and puts 24 step responses out to a window.

```r
impulse(model=canmodel,steps=20,factor=bfactor,$
  window="Responses",labels=\{|"f1","f2","r1","r2","n1","n2"\}|)
```

puts out to a window a 20 step response to each of the components in an orthogonalized system. The shocks are given labels of f1, f2, r1, r2, n1 and n2.

```r
impulse(steps=20,column=2,cv=v) 3
# 3 impm 1 3
# 1 impg 1 1
# 2 impp 1 2
```

looks at the shock to the orthogonalized innovation to equation 1 (2nd listed), with the decomposition done in the order 3–1–2. The responses are saved in series IMPM, IMPG, IMPP.
impulse(input, steps=20, noprint) 3
# 1 resp1
# 2 resp2
# 3 resp3
# 1.0  -1.0  0.0

computes responses over 20 periods to a shock in the first period of 1.0 to the first equation and –1.0 to the second. This stores the responses in RESP1, RESP2 and RESP3.

set shokm 1998:3 1999:1 = 0.01
set shokm 1999:2 2001:4 = 0.00
impulse(paths, start=1998:3, steps=10, noprint) 3
# 1 impg 1998:3
# 2 impp 1998:3
# 3 impm 1998:3
#  ** shokm

gives shocks of size .01 to the third equation in each of the first three periods.

See Also . . .

*UG*, Section 10.5 Orthogonalization.
*UG*, Section 10.6 Using IMPULSE and ERRORS.
ERRORS Decomposition of Variance.
Include

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 in RATS format files using the mouse. First, open or create a RATS format file by doing File–Open RATSDATA or File–New RATSDATA.

Then, use the Show Series Window wizard on the Data menu to display a list of the series in memory, and drag and drop the desired series from the “Show Series” window onto the RATSDATA 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 name file series already on the file. If you wish to change an existing file series, use STORE or EDIT, or DELETE the series before INCLUDEing.

- **series**: The series (in working memory) which supplies the values for file series. Since you often will be using the same name on the data file, the default for series is file series. 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.

The comments parameter used in older versions has been replaced by the COMMENTS option described below. Version 7 will still recognize this parameter, but you should use the option for new programs.

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 `file series`. RATS takes the data for this from `series entries start to end`. If you only specify the `file series` 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[STRING]` ahead of time, or create it directly as the argument for the option, as shown in the example below.

Example

```plaintext
* 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
save
```
INFOBOX — Informational Dialog Boxes and Progress Bars

Use INFOBOX if you want to inform the user what is happening during a lengthy operation. It brings up a dialog box which sits on top of the other RATS windows. You can display a graphical “progress bar” and up to two lines of text in the box. The dialog box can be updated to display a change in status or continuing progress.

```
infobox(action=define, options)  "messagestring"
infobox(action=modify, options )  "messagestring"
infobox(action=remove)
```

Parameters

"messagestring" Message to display in the dialog box. When used with the ACTION=DEFINE option, this sets the first line of text. *The first line cannot be changed later.* Use this to describe the overall operation. With ACTION=MODIFY, messagestring sets the second line of text, which you can change each time you use ACTION=MODIFY.

You can supply this as a string of text enclosed in quotes, or as a LABEL or STRING type variable.

Options

**action=define/[modify]/remove**

ACTION=DEFINE creates and displays the information box. If you want to display a progress bar in the dialog, you must include the PROGRESS, LOWER, and UPPER options when you do ACTION=DEFINE. The messagestring parameter, if supplied, is displayed as the first line in the dialog box.

ACTION=MODIFY updates the box. Use this along with the CURRENT option to update the progress bar, or with the messagestring parameter to replace the second line of text in the dialog box.

The ACTION=REMOVE option removes the box from the screen.

**progress/[noprogress]**

If you want to display a progress bar, use this option when you do ACTION=DEFINE. You must also use LOWER and UPPER to set the lower and upper bounds for the progress bar.
lower=lower bound for progress bar [1]
upper=upper bound for progress bar

Use these, with ACTION=DEFINE and PROGRESS, to set the lower and upper bounds for the progress of your operation. LOWER and UPPER are integer-valued.
current=current integer value of progress bar

Use this (with ACTION=MODIFY) to indicate how much of the distance from LOWER to UPPER has been completed. For example, if you set LOWER=1 and UPPER=100, INFOBOX(ACTION=MODIFY,CURRENT=50) would display the bar as 50% completed.

Example

If you are doing something like a time-consuming Monte Carlo simulation which loops over a large number of draws, you can use INFOBOX to keep the user (or yourself) informed of its progress.

infobox(action=define,progress,lower=1,upper=ndraws) $
"Monte Carlo Simulation"
do draws=1,ndraws
  *** Monte Carlo simulation instructions here
  .
  .
  .
  infobox(current=draws)
end do draws
infobox(action=remove)

This displays an information box with a progress bar and a single descriptive line. The progress bar is incremented by one each trip through the loop.

See Also . . .

DBOX
Generates custom dialog boxes for user-input. These can include check boxes, text input fields, lists of “radio” buttons, and more.

MESSAGEBOX
Displays simple informational messages in a dialog box. Unlike INFOBOX, a MESSAGEBOX pauses program execution and waits for the user to respond (using OK, OK or Cancel, or Yes, No, or Cancel buttons).
**INITIAL — Solving Yule-Walker Equations**

*INITIAL* computes initial estimates for a simple (non-multiplicative) ARMA equation. *INITIAL* is a holdover from earlier versions of RATS.

```
initial(options) equation start end
```

**Parameters**

- **equation**
  - Equation whose initial estimates are to be computed.

- **start**  **end**
  - Range of entries to use in computing the covariances needed for the calculation. If you have not set a *SMPL*, this defaults to the defined range of the dependent variable of *equation*.

**Options**

- **[print]/noprint**
  - The output from *INITIAL* lists the variables, lags and initial estimates of the coefficients. You can suppress the output with *NOPRINT*.

- **covariances=series of autocovariances**
  - Use the option *COVARIANCES* to solve for the ARMA representation of a process with a specific covariogram. The *series of autocovariances* should start with lag 0 (the variance) in entry 1 and should have at least as many lags as the highest AR lag plus the highest MA lag.

- **transfer=source equation for coefficients**
  - If you use *TRANSFER*, *INITIAL* does no calculations. It simply copies coefficients from *source equation* to *equation*. Any variables in *equation* which do not appear in *source equation* get zero coefficients.

**Description**

*INITIAL* computes initial estimates for the ARMA parameters of *equation* using the autocovariances of the dependent variable, or those provided by the *COVARIANCES* option. *INITIAL* uses the algorithms on pp. 223-224 of Box-Jenkins (1976), using the linearly convergent (Gauss-Seidel) algorithm for the initial estimates of the moving average part. If the equation is an autoregression, this process produces solutions to the Yule-Walker equations.

There may be no solution to the system of equations used for the moving average part. For instance, no MA(1) model is compatible with a first lag correlation greater than .5. This failure usually indicates a poorly specified or overparameterized model. If RATS cannot solve the system of equations, it issues a warning, sets the estimate for the highest MA lag to zero and computes estimates for the remaining coefficients.
INPUT — General Information Input

INPUT reads data into arrays and variables. It is strictly free-format (fields separated by blanks, commas, tabs or end-of-lines) and you can use it for all types of numeric or character variables. By contrast, READ has a FORMAT option and can use free-format, binary or Fortran formatted input. The FIXED instruction is useful for setting arrays of fixed values in PROCEDURES and FUNCTIONS.

\[
\text{input( option )} \quad \text{arrays, variables, array elements}
\]

Parameters

arrays,... These are the objects for which data is to be read. You can use any combination of variables. Any arrays must be dimensioned ahead of time (unless you use the option VARYING).

Options

unit=[input]/data/other unit

INPUT reads the data from the specified I/O unit. The INPUT unit (the default setting) is simply the console if you are working interactively or the current input file if you are working in batch mode or have done a SOURCE or OPEN INPUT instruction.

varying/[novarying]

status=INTEGER variable set to 0,1 status

[singleline]/nosingleline

These are more advanced options. See their description later in this section.

Description

Some general information about INPUT:

- It reads the arrays and variables in the order listed.
- It fills the elements of arrays in the order described below.
- It can read two or more arrays from the same line of data. It can read two or more rows of an array from a single line.
- It reads complex numbers as a pair of real numbers: the real and imaginary parts, respectively.
- A STRING variable is filled with the contents of a complete line.

Before you can INPUT data for a variable, you must first introduce it with DECLARE or some other instruction. You also need to dimension any array prior to using it in an INPUT instruction, unless you use the VARYING option.
Organization of Arrays

These are the three general forms of arrays and their organization:

**One-dimensional arrays (VECTOR data type)**
are written and read as row vectors.

**General two-dimensional arrays (RECTANGULAR data type)**
are written and read by rows (the natural fashion), even though internally they are stored by columns.

**Symmetric two-dimensional arrays (SYMTRIC data type)**
are written and read by rows of the lower triangle of the matrix. Thus, the first row has one column, the second has two, etc.

Example

declare symmetric v(3,3)
declare real y
declare vector[complex] c2(2)
declare integer i
input v y c2 i

1.0  2.0  3.0  4.0  5.0  6.0  7.3
0.0  1.0  1.0  0.0  5

sets all of the following:

\[
\begin{bmatrix}
1.0 \\
2.0 & 3.0 \\
4.0 & 5.0 & 6.0
\end{bmatrix}
\]

\(v = \begin{bmatrix} 1.0 \\ 2.0 & 3.0 \\ 4.0 & 5.0 & 6.0 \end{bmatrix} \)

\(y = 7.3\)

\(c = \begin{bmatrix} 0.0 + 1.0i \\ 1.0 + 0.0i \end{bmatrix} \)

\(i = 5\)

Notes

You can also use COMPUTE for the initialization above:

\[
\text{compute [symmetric] v=|1.0,2.0,3.0|4.0,5.0,6.0|} \\
\text{compute y=7.3} \\
\text{compute [vector[complex]] c2=|%cmplx(0.0,1.0),%cmplx(1.0,0.0)|} \\
\text{compute i=5}
\]

We prefer to use COMPUTE for integer and real scalars and vectors. For arrays with multiple rows, however, INPUT is easier to read and requires fewer extra characters. Compare the initializer for \(v\) above with
dec symm v(3,3)
input v
  1.0
  2.0 3.0
  4.0 5.0 6.0

Of course, **INPUT** is only useful when you are supplying explicit values. If you want to set an array using variable names or other expressions, use **COMPUTE** instead.

### Advanced Options

**varying/[novarying]**

**status=INTEGER** variable set to 0,1 status

**[singleline]/nosingleline**

The **VARYING** and **STATUS** options allow you to work with lists whose size you do not want to set in advance.

You can use **VARYING** to input data for a single **VECTOR** array of any numeric or character type. With **VARYING**, the **VECTOR** is filled with as much data as is available. By default, this is whatever is on a single line of data. With **NOSINGLELINE**, it will read data until the end of the file—though **READ** is preferable for data coming from a file.

If you use the option **STATUS**, **INPUT** does not give you an error if there is not enough data to fill all the variables. Instead, it sets your status variable to 0. If the **INPUT** is successful, it sets the status variable to 1.

### Example

```plaintext
dec vector v
input(varying) v
  1 5 10 25 50 100 250 500 1000
```

**VARYING** is useful if you are too busy to count the entries or want to make quick changes to the list without having to worry about changing the dimensions.

### See Also . . .

- **READ**: An alternative to **INPUT** with a wider set of options. **READ** is designed primarily for reading data from an external file.
- **WRITE**: Writes arrays and variables to output or to an external file.
- **QUERY**: Requests input from the program’s user.
- **MEDIT**: Inputs a matrix from a screen data editor.
- **ENTER**: Inputs data from supplementary cards.
- **FIXED**: Defines fixed-value arrays in user-defined functions and procedures.
Inquire

INQUIRE — Obtaining Information about Series

INQUIRE lets your program obtain information, principally about the ranges of series. This is needed if you are writing general procedures and need to adjust the range to analyze based upon the input series.

```plaintext
inquire( options ) value1<<p1  value2<<p2
#  list of variables in regression format (only with the REGRESSORLIST option)
```

Parameters

- **value1  value2** These are INTEGER variables or INTEGER elements of a VECTOR or RECTANGULAR array which are filled with the information requested. Some options only return one value—you do not need to specify a variable for value2 in such cases.

- **p1  p2** (Optional) Use these in situations where you want your PROCEDURE to mimic the standard “these default to the defined range of series” behavior of RATS instructions. p1 and p2 should be PROCEDURE parameters or options of type INTEGER. value1 and value2 will take the values of p1 and p2, if explicit values for those are provided. Otherwise, they get the INQUIRE values. See the first example below.

Options (Mutually Exclusive)

- **series=**series name or number
  returns the defined range of the indicated series.

- **dseries=RATS data file series name**
  This may only be used with series on a RATS format data file which you have opened with the instructions DEDIT or ENV RATSDATA=file name. It sets value1 and value2 to the defined limits of the indicated series in terms of the current CALENDAR seasonal.

- **regressorlist/[noregressorlist]**
  Use REGRESSORLIST and a supplementary card listing a set of variables in regression format when you want to determine the maximum defined range for a set of variables. value1 and value2 are set to the starting and ending entries of that range. EQUATION is similar, but determines the range based on the variables in the equation you supply. Note: Use the SERIES option for a single variable.
**smpl/[nosmpl]**

`value1` and `value2` are set to the starting and ending entries of the current `SMPL`.

**lastreg/[nolastreg]**

`value1` and `value2` are set to the starting and ending entries of the last regression. You can also get these with the `%REGSTART()` and `%REGEND()` functions.

**valid=** `dummy series to create`

Used in conjunction with `DSERIES`, `LASTREG`, `REGRESSORLIST`, or `SERIES`, this creates a 1/0 dummy series, with 1’s for entries at which the input series or regression is defined, and zeros elsewhere. This can then be used with `SMPL` options on other instructions to select a sample range.

**matrix=** `matrix name`

This returns the dimensions of the indicated matrix. `value1` and `value2` are set to the number of rows and columns, respectively. Note that you can also get the number of rows and columns with `%ROWS(matrix)` and `%COLS(matrix)` functions.

**seasonal/[noseasonal]**

This returns as `value1` the current calendar seasonal: for instance, 4 for a quarterly `CALENDAR`, 12 for a monthly `CALENDAR`.

### Examples

```plaintext
procedure test series start end
type series series
type integer start end
local integer startl endl
inquire(series=series) startl=start endl=end

This is similar to the entry code for many of the procedures which we provide with RATS. Let’s look at three possible command lines to execute `TEST`.

@test gnp82
@test gnp82 1947:1 2002:4
@test gnp82 1955:1 *
```

In the first, `STARTL` and `ENDL` will be the start and end of the series GNP82. In the second, they will be 1947:1 and 2002:4 and in the third, 1955:1 and the end of GNP82.
```
env ratsdata=modeldat.rat

cal(q) 1960:1
inquire(dseries=gdp) * endgdp
inquire(dseries=unemp) * endunemp
compute dataend=%imin(endgdp,endunemp)
allocate dataend
```

This checks the current length of the series GDP and UNEMP on the file MODELDAT.RAT, and sets DATAEND to the minimum of the two. You can use INQUIRE in this fashion to write a forecasting program, for instance, which requires no modification from month to month except updates of the data file.

### See Also...

RATS also offers a number of functions for getting information about arrays, series, and other information:

- `%ROWS(matrix)` Returns the number of rows in a matrix.
- `%COLS(matrix)` Returns the number of columns in a matrix.
- `%REGSTART()` Returns the starting entry of the last regression.
- `%REGEND()` Returns the final entry of the last regression.
- `%EQNSIZE(equation)` Returns the number of explanatory variables in an equation. Use `equation=0` to get information for the last regression in this and the next six functions.
- `%EQNTABLE(equation)` Returns a 2xK INTEGER array of the explanatory variables of an equation. The first row is the series number, the second is the lag.
- `%EQNCOEFFS(equation)` Returns the vector of coefficients from an equation.
- `%EQNDEPVAR(equation)` Returns the dependent variable of an equation.
- `%EQNREGLABELS(equation)` Returns a K vector of STRINGS which gives the regressor labels as they appear on regression output.
- `%MODELSIZE(model)` Returns the number of equations or formulas in a model.
- `%MODELDEPVARS(model)` Returns a VECTOR[INTEGER] which lists the dependent variables of the equations or formulas in a model.
INSTRUMENTS — Setting the Instrument List

INSTRUMENTS creates a list of instrumental variables. The instructions LINREG, AR1, SUR, NLLS and NLSYSTEM use these for instrumental variables estimation, while CMOMENT, SWEEP, and MCOV can use the lists in calculations. You do not need to construct “first-stage regressors” through preliminary regressions. These are done automatically by the estimation instructions which need them.

\[ \text{instruments}(\text{options}) \text{ exogenous variables in regression format} \]

Wizards

The relevant regression and estimation wizards provide fields for including instrumental variables, so you do not need to do an INSTRUMENTS instruction if you will be using a wizard to do the estimation.

Parameters

\text{variables} List the exogenous and predetermined variables in regression format. Note that the CONSTANT isn't included automatically as an instrument. If you need it, put it in the \text{variables} list.

Options

\text{drop/}[\text{nodrop}] With ADD and DROP, you can make small changes to the existing list of exogenous variables. ADD adds the new list to the existing one, while DROP removes any of the listed variables. ADD and DROP are useful when you estimate large models with many potential instruments. If each equation uses a different subset of the instruments, these options can simplify specification of the instrument sets. See the second example below.

\text{print/}[\text{noprint}] Use PRINT to list the current set of instruments.

Example

\text{instruments constant trend govtwage taxes govtexp }$
\text{ capital(1) production(1) profit(1)}$

sets the instruments list for Klein’s Model I.
Notes

The instruction `NLSYSTEM` has a special option `MASK` which allows a different set of instruments to be used for each equation in the system. You have to provide a `RECTANGULAR` array with dimensions “number of instruments” x “number of formulas” which has 1.0 in a cell in the column $j$ if and only if you want instrument $i$ to be used for formula $j$. For instance:

```plaintext
instruments constant csz{1 to 4} pcs{1 to 4} aaz{1 to 4}
dec rect mask(13,2)
compute %do(i,1,13,mask(i,1)=(i<=9))
compute %do(i,1,13,mask(i,2)=(i<=5.or.i>=10))
nlsystem(mask=mask,instruments) / frml frm2
```

would use `CONSTANT`, lags of `CSZ` and lags of `PCS` for the first formula and `CONSTANT`, lags of `CSZ` and lags of `AAZ` in the second.

You need to be careful in using lags as instruments. For instance, in the example of the large simultaneous equations model above, lags 1 to 4 of $Y$ are used as instruments. Since $Y{4}$ isn’t available until $T=5$ (at a minimum), the estimation range can start no earlier than period 5. This can be a major problem in a panel data set, as you lose data points in each cross section. An alternative to using lag notation is to create a separate series for each lag, but with zero values where the lagged data is unavailable. For panel data, use the `%PERIOD` function to get the time period within an individual’s data. For instance, rather than $Y{1 to 4}$, you could do the following:

```plaintext
set y1 = %if(%period(t)<=1,0.0,y{1})
set y2 = %if(%period(t)<=2,0.0,y{2})
set y3 = %if(%period(t)<=3,0.0,y{3})
set y4 = %if(%period(t)<=4,0.0,y{4})
```

and then use $Y1 \ Y2 \ Y3 \ Y4$ on the instrument list.

See Also . . .

- `SWEEP` Projects a set of target variables on a set of other variables.
- `UG`, Section 5.6 Instrumental Variables and Two-Stage Least Squares.
- `UG`, Section 7.8 Method of Moments Estimators (Univariate).
- `UG`, Section 7.9 Non-Linear Systems Estimation.
- `%instlist()` Function returning current instruments as a regressor list
- `%insttable()` Function returning current instruments as a table
- `%instxvector(t)` Function for extracting an $X(t)$ for the current instrument set
KALMAN — Kalman Filtering

**KALMAN** executes one step of the Kalman filter algorithm (*User’s Guide*, Section 10.13) on a system of equations that you have set up with **SYSTEM**. In its simplest use, it does sequential updating of coefficient estimates.

**KALMAN** is used specifically to estimate the coefficients of a system of linear equations over time. The instruction **DLM** is a more general instruction which can apply the Kalman filter in a wider range of settings.

The **KFSET** (Kalman Filter SETup) and **TVARYING** (Time VARYING) subcommands of **SYSTEM** can be used to set information that is used by the Kalman filter to relax the assumptions used for simple coefficient updates.

```
kalman( options )   residual   coeffs   printflag
```

**Parameters**

The parameters 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 **COEFPSS** 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.

**Basic Options**

- **print/[noprint]**
- **ftests/[nofests]**

These control the printing of the regression and $F$ test output. The defaults are **NOPRINT** and **NOFTESTS**. If you want to display output based on the value of a conditional expression, you can use the syntax **OPTION=(condition)**. See “Examples” later in this section for more.

**Advanced Options**

- **startup=startup entry**

When you start the Kalman filter without an **ESTIMATE**, the first **KALMAN** instruction should include this option. **KALMAN** instructions thereafter will recompute coefficients given the next observation.

- **rtype=[all]/current/onestep/recursive**

This option allows you to specify how you want the residuals series filled.

- **ALL** : all entries using the updated coefficient estimates.
- **CURRENT** : just the current entry, using the updated coefficients.
- **ONESTEP** : just the current entry, using the previous coefficients.
- **RECURSIVE** : recursive residual, just the current entry
**backwards**/[noback]

With the **BACKWARDS** option, the filter operates backwards for this update, adding an observation to the beginning of the sample rather than to the end. **KALMAN** replaces the lags in the equation with leads, and vice versa.

**drop**=observation to drop

**add**=observation to add

Use **DROP** and **ADD** either to drop an observation from the sample or add one. You can use **DROP** along with the **TEMP** and **CHANGE** options (below) to see the effect upon the coefficients of dropping a single observation.

**residuals**=VECTOR[SERIES] for residuals

This is the most convenient way to get the residuals from the equations of a VAR or other multiple equation system. The option **RESIDUALS**=**RESVAR** will create series **RESVAR**(1), . . . , **RESVAR**(n) which will have the residuals from the n equations in the system.

**coeffs**=RECTANGULAR for coefficients

For a VAR, this saves the estimated coefficients in a **RECTANGULAR** array. Column i of this will be the coefficients from the i-th equation.

**cohistory**=VECTOR[SERIES] for coefficient history

This can only be used with a single equation model. There will be one series in the **VECTOR**[SERIES] for each coefficient. **KALMAN** will put its estimates into the current period in these series.

**smpl**=**SMPL** series or formula (**User's Guide**, Section 5.2)

You can supply a series or a formula that can be evaluated across entry numbers. Entries for which the series or formula is zero or “false” will be skipped, while entries that are non-zero or “true” will be included in the operation.

**title**="title to identify output"

You can use this to provide a description of the estimation method.

**temp**/[notemp]

**change**=VECTOR of coefficient changes

**TEMP** causes **KALMAN** to compute the new coefficients and (optionally) residuals, then discards the results, leaving the filter in the same state as it was before the **KALMAN** instruction. Use **CHANGE** with or without **TEMP** to save the change in the coefficient vector of a one equation system. You do not need to **DECLARE** or **DIMENSION** the **VECTOR**.
**x** = VECTOR of explanatory variables for dummy observation  
**y** = VECTOR of dependent variables for dummy observation  
**v** = VECTOR of equation variances  
(for a dummy or regular Kalman filter observation)

These are used to implement “dummy observation priors” for a vector autoregression. These take the form \( Y = XB + v \) where \( B \) is the matrix of VAR coefficients. **KALMAN** updates the coefficients, but does not alter the likelihood vector and does not change the entry pointer.

The \( v \) vector gives the variances of \( v \); if you don’t include it, the equation variances provided on the **KFSET** are used. If you do not use \( y \), the Kalman filter error is taken to be zero, so the coefficients don’t change; only the covariance matrix of the coefficients.

Dummy observation priors are described in greater detail in Sims (1993).

### Coefficient Updating

To use the Kalman filter for coefficient updates, define the system using the **SYSTEM** and related instructions, and then use **ESTIMATE** to initialize the Kalman filter, computing estimates over some subset of the data range. Each **KALMAN** instruction thereafter will add a single observation to the end of the data set.

#### Variables Defined

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%NREG</td>
<td>number of regressors in the first equation (INTEGER)</td>
</tr>
<tr>
<td>%XX</td>
<td>the ( X'X^{-1} ) matrix (SYMMETRIC)</td>
</tr>
</tbody>
</table>

#### Examples

```rats
system(model=yrmp)  
variables gdp cpr ipd m1  
lags 1 to 4  
det constant  
end(system)  
estimate 1948:1 2003:4  
do time=2004:1,2006:4  
kalman(print=(time==2006:4))  
end do
```

**ESTIMATE** computes the regressions through 2003:4 and prints them with the \( F \)-tests. The loop then executes **KALMAN** twelve times (quarterly data) so that the last produces estimates using data through 2006:4. This only prints the estimates for this final period because the argument for **PRINT** will be zero until **TIME** is 2006:4.
equation kfeq y
# constant x1 x2 x3
system kfeq
end(system)
estimate(cohistory=cokalman) 1 25
do entry=26,200
  kalman(cohistory=cokalman)
end do entry

This Kalman filters a single equation over the period 26 to 200 and saves time series of the coefficients in series COKALMAN(1),...,COKALMAN(4).

See Also . . .

UG, Section 10.13 Kalman Filter.
DLM General state-space and dynamic linear modeling
SYSTEM Initial instruction in the definition of a system.
KFSET Defines parameters for a Kalman filter.
TVARYING Defines parameters for time-varying coefficients.
ESTIMATE Estimates a system of equations.
KFSET — Kalman Filter Setup

KFSET is a subcommand of SYSTEM which describes the setup of the Kalman filter that you want to apply to the equations. KFSET, KALMAN, and TVARYING are used specifically for estimating the coefficients of a linear model. The instruction DLM is a more general instruction which can apply the Kalman filter more widely, such as for solving and estimating state-space models.

See Section 10.13 of the User’s Guide for more on the use of KFSET and related instructions.

\[
\text{Kfset( options ) list of covariance matrices}
\]

# innovation variances (if VARIANCE=KNOWN, CONSTANT and no VECTOR)
# list of variance series (if VARIANCE=KNOWN and NOCONSTANT)

Parameters

\[
\text{list of matrices}
\]

List of SYMMETRIC arrays which are to hold the covariance matrices of the coefficients (states). You can use the option NAMES as an alternative to listing these explicitly.

There should be one array for each equation in the system, except if the system is a simple OLS vector autoregression. If it is, you only need one array.

If you do not need access to the covariance matrices (either to set them or to examine them), you can omit the list of matrices. RATS will store all information internally.

Technical Information

RATS analyzes the equations in the system individually. The following model is used for each equation:

- The coefficient vector \( \beta_t \) is the vector of “states” at time \( t \).
- The measurement equation is \( y_t = X_\beta_t + u_t \), where the variance of \( u_t \) is \( n_t \).
- The state vector follows the process: \( \beta_t = \beta_{t-1} + v_t \), with \( \text{Var}(v_t) = M_t \).
- \( u_t \) and \( v_t \) are independent.

Let \( \beta_{tit} \) be the estimates of \( \beta_t \) using information through \( t \), and let the covariance matrix of \( \beta_{tit} \) be written \( \Sigma_t \). What KFSET provides are

1. The \( \Sigma_t \) matrices (one for each equation)
2. The \( n_t \). The options VARIANCE, CONSTANT, VECTOR and the supplementary cards are used to allow you to handle both situations where \( n \) is constant over time, and those where it changes.
Options

variance=[concentrated]/known
constant/[noconstant]  (with VARIANCE=KNOWN only)
v=VECTOR of variances  (with VARIANCE=KNOWN, CONSTANT only)

Use these options for setting the variances of the measurement equation errors \( n \). Note that constant variances are provided in a VECTOR. This is done because a SYSTEM can, and often does, have several equations. If you're analyzing a single equation, just use a vector with dimension one, or the notation \| \|variance\| \|. The VARIANCE option replaces the SCALE option used in previous versions.

likelihood=2 x Equations RECTANGULAR array

This allows computation (by KALMAN instructions) of a pseudo-likelihood function for Kalman filter estimates stored in the array. See Doan, Litterman and Sims (1984), page 10 and Section 10.13.2 of the User’s Guide for more information.

names="base for array names"

This is an alternative to using the list of matrices parameter. When you use NAMES, KFSET forms names formed by appending the numbers 1,2,3,etc. to the base label you supply. That is, if you use NAMES="COVMAT", KFSET will create COVMAT1, COVMAT2, etc.; as many as are needed. NAMES is largely obsolete, because you now use a VECTOR of symmetric arrays for the list of matrices.

Description

KFSET uses the list of matrices in two ways:

• If you execute KALMAN without an ESTIMATE, list of matrices supplies the initial estimates of the covariance matrices of the coefficients (states). You have to dimension the array(s) and set them before you can do the KALMAN.

• If you do execute an ESTIMATE, RATS fills the list of matrices with the estimated covariance matrices. For this use, you do not need to dimension the arrays because ESTIMATE does it for you.

If you are using KALMAN without ESTIMATE, the initial coefficients also need to be set. This is usually done with the ASSOCIATE instruction.

If you want time-varying parameters \( M \), non-zero), you need to use TVARYING as well as KFSET.

Notes

If you want to write a single set of code which can handle VARs of various sizes, we recommend that you use a VECTOR [SYMMETRIC] in place of a list. For instance,

dec vector[ symmetric ] kfs(neqn)
kfset kfs

makes KFS(1), KFS(2),..., KFS(NEQN) the list of matrices.
Equation Variances

There are several options which control the behavior of the measurement equation variances. Choose the method which is correct for your assumptions:

**Constant over time and unknown**
Use the option \texttt{VARIANCE=CONCENTRATED} (default). RATS computes new estimates for the variances whenever you request output from \texttt{ESTIMATE} or \texttt{KALMAN}. \textit{If you are using time-varying parameters (M_t is non-zero), you should use one of the VARIANCE=KNOWN alternatives.}

If the system is a vector autoregression without a prior, it is possible to list just one array on \texttt{KFSET} since all equations have the same $X'X$ inverse matrix. See the example below.

**Constant over time and known**
Use \texttt{VARIANCE=KNOWN} with \texttt{CONSTANT}. You can supply the values in one of two ways:

- with the option \texttt{V=VECTOR of variances}
- with a supplementary card

**Changing over time**
Use \texttt{VARIANCE=KNOWN} and \texttt{NOCONSTANT}, and include a supplementary card which lists series whose values are the desired variances.

\textbf{Examples}

```plaintext
system(model=rmpy)
variables cpr m1 ppi ip
lags 1 to 12
kfset xxsys
end(system)
estimate 1948:1 2006:12

\texttt{ESTIMATE} \textit{dimensions and sets XXSYS}
```

uses \texttt{KFSET} to obtain the $X'X^{-1}$ matrix from the estimation of a VAR. Since there is no prior, we only list one array on \texttt{KFSET}.

```plaintext
equation kfeq y
# constant x1 x2
system kfeq
kfset(variances=known,constant) xxmat
# .01
tvarying tvmat
end(system)
dimension xxmat(3,3) tvmat(3,3)
```

Time varying coefficients with $n_t=.01$. You need to set the initial values of \texttt{XXMAT}, \texttt{TVMAT} and the coefficients of \texttt{KFEQ} before you can use \texttt{KALMAN}. 

Labels

LABELS — Setting Output Labels for Series

LABELS attaches an output label to a numbered series. RATS uses these output labels whenever it prints out a series. They are also used when you read data from a labeled file: RATS format, PORTABLE and spreadsheet formats.

labels list of series (usually by number)
  # labels for series (enclosed in single or double quotes)

Parameters

list of series List of series to be given labels.

Supplementary Cards

The labels can be any collection of characters (up to sixteen) enclosed within single or double quotes. You can also use string expressions, LABEL variables, or elements of an array of LABELS.

Notes

The instruction EQV is similar, but goes beyond LABELS to attach a name which can also be used on input, to reference the series. LABELS is usually more appropriate, as labels aren’t subject to the restrictions put on symbolic names—you can use any combination of characters (up to sixteen). Note also, that any number of series can share an output label, while series names (set by EQV) must be unique.

Example

open data setup.dat
declare vector[label] pairs
read(varying) pairs
compute npairs=%rows(pairs)/2
open data ticker.rat
cal(d) 1988:1:5
allocate 2002:12:31
dec vector[series] tickseries(2)
do i=1,npairs
   labels tickseries
      # pairs(2*i-1) pairs(2*i)
      data(format=rats) / tickseries
      ...
end do i

reads a (possibly very long) list of pairs of series names from the file SETUP.DAT. NPAIRS is the number of pairs read. The series are read, two at a time, from the RATS format file TICKER.RAT, and some (unspecified) analysis is performed. You need to use LABELS because DATA(FORMAT=RATS) searches for the data on the basis of the series labels.
LAGS — Listing the Lags for a VAR

**LAGS** is one of the subcommands of **SYSTEM** used to define a vector autoregression (VAR). Use it to list the lags of the endogenous variables in the VAR.

```
lags list of lags
```

**Wizard**

The **VAR (Setup/Estimate)** wizard on the **Statistics** menu provides an easy, dialog-driven interface for defining and estimating VAR models.

**Parameters**

```
list of lags
```

The set of lags of each of the endogenous variables which will go into each created equation. Usually these will be consecutive lags as shown below, but you can skip lags. For example:

```
LAGS 1 2 3 6 12
```

**Example**

```
system(model=canmodel)
variables cangnp canm1 cantbill canunemp canusxr usagnp
lags 1 to 4
det constant
end(system)
```

defines a 6 variable VAR with 4 lags of each variable plus a constant in each equation.

**Notes**

If you use **VARIABLES** and **LAGS** to define your VAR, each equation will include the same list of lags for each variable listed on **VARIABLES**. You cannot, for instance, use a different set of lags for the dependent variable or leave lags of variable $z$ out of the variable $x$ equation. You can get such flexibility by defining each equation separately using the instruction **EQUATION**. However, you cannot use a prior with such models.

If you use the **ECT** instruction to define an error-correction model, use the actual lag length that you want for the original, undifferenced model—RATS will automatically drop one lag when estimating the differenced error correction form. **ECT** models do **not** support non-consecutive lists of lags. If you use **ECT** in your **SYSTEM** definition, RATS will always define the model using all lags from 1 to $L$, where $L$ is the longest lag specified on the **LAGS** instruction. For an error correction model with non-consecutive lags, you will need to create the differenced variables and error correction terms manually and specify the model in error correction form directly, without using **ECT**.
LDV

LDV — Limited Dependent Variable Estimation

The LDV instruction implements limited dependent variable estimation techniques, for models with censored or truncated data.

\[ \text{ldv( options ) depvar start end residuals} \]

# list of explanatory variables in regression format

Wizard

The Limited/Discrete Dependent Variables wizard on the Statistics menu provides dialog-driven access to most of the features of the LDV instruction.

Parameters

- **depvar**: Dependent variable. RATS requires numeric coding for this.
- **start, end**: Estimation range. If you have not set a SMPL, this defaults to the maximum common range of all the variables involved.
- **residuals**: (Optional) Series for the residuals.

Options

- **truncate=[neither]/lower/upper/both**
- **censor=[neither]/lower/upper/both**
- **interval/[nointerval]**

These choose the type of estimation method to be employed. The TRUNCATE options are used when an observation is in the data set only if the dependent variable is in range. CENSOR is used when you can observe the data points which hit the limit. INTERVAL is used when the data actually consist only of the upper and lower limits, that is, all you can observe are bounds above and below. With the interval estimation, you still need the dependent variable, but it is used only to determine which observations to use.

- **upper=SERIES of upper limits**
- **lower=SERIES of lower limits**

Use these options to supply series containing the upper and/or lower bound values as required by your choice of TRUNCATE, CENSOR, or INTERVAL options (note that INTERVAL requires that you supply both UPPER and LOWER series). Use missing value codes for any entries that are to be treated as unlimited.

- **sigma=** input value for the regression [none - estimated]

You can use this to input a value for the standard deviation of the regression equation. If not, it will be estimated.
gresids=SERIES of generalized residuals [unused]
   Use this option if you want to save the generalized residuals to a series.

[print]/[noprint]
vcv/[novcv]
smpl=SMPL series or formula (User’s Guide, Section 5.2)
unravel/[nounravel] (Section 5.11)
equation=equation to estimate
   These are similar to the LINREG options.
title="title for output" [depends upon options]
   This option allows you to supply your own title to label the estimation technique in the output.

iterations=iteration limit [100]
subiterations=subiteration limit [30]
cvcrit=convergence limit [.00001]
trace/[notrace]
initial=VECTOR of initial guesses  [vector of zeros]
   LDV models are estimated using non-linear methods. Because the log likelihood function is well-behaved, the estimation is always done using Newton-Raphson, that is, using analytical second derivatives. ITERATIONS sets the maximum number of iterations, SUBITERS sets the maximum number of subiterations, CVCRIT the convergence criterion. TRACE prints the intermediate results. See the User’s Guide Chapter 7 for details. INITIAL supplies initial estimates for the coefficients. The default values are usually sufficient.

robusterrors/[norobusterrors]
lags=correlated lags [0]
lwindow=neweywest/bartlett/damped/parzen/quadratic/[flat]/panel/white
damp=value of γ for lwindow=damped [0.0]
lwform=VECTOR with the window form [not used]
cluster=SERIES with category values for clustered calculation
   These permit calculation of a consistent covariance matrix allowing for heteroscedasticity (with ROBUSTERRORS) or serial correlation (with ROBUSTERRORS and LAGS), as well as clustered standard errors. See Sections 5.3, 5.4, and 5.5 of the User’s Guide and the description of the instruction MCOV for more information.

Note, however, that the estimates themselves may be inconsistent if the distributional assumptions are incorrect.
Description

The technical information on these models is provided in Section 14.3 of the User’s Guide. All the models are based upon the standard

$$y_i = X_i\beta + u_i ; u_i \sim N(0,\sigma^2) \text{i.i.d.}$$

(1)

They differ upon when and what values can be observed for the dependent variable. Note that while theoretically you can have a data set which is truncated at one end and censored at the other, LDV isn’t designed for it.

Truncated and censored models tend to be fairly easy to set up. The INTERVAL estimator is a bit trickier. It’s used when all that is observed for an individual is a pair of values which bracket the true dependent variable. If you have hard numbers for the upper and lower bounds for all observations, you’re unlikely to get much of an improvement from using LDV versus a linear regression using the interval midpoints for the dependent variable. INTERVAL is most useful when some of the observations are unlimited on either end. With the interval estimator, the “dependent variable” is provided using two series, one indicated with the UPPER option and one with the LOWER option. The dependent variable is used only to determine which observations are valid; LDV can’t look just at the upper and lower series, since a missing value in them is used to show no limit in that direction.

All models are estimated by Newton-Raphson on the model reparameterized as described in Olsen (1978), that is, with \( \{\gamma, h\} \equiv \{\beta/\sigma, 1/\sigma\} \). This assumes that \( \sigma \) is being estimated: if you want to input a specific value, use the SIGMA option. With this parameterization, for instance, the log likelihood for an observation for the interval model is

$$\log L = \begin{cases} 
\log(1 - \Phi(L_i h - X_i \gamma)) & \text{if unbounded above} \\
\log(\Phi(U_i h - X_i \gamma)) & \text{if unbounded below} \\
\log(\Phi(U_i h - X_i \gamma) - \Phi(L_i h - X_i \gamma)) & \text{otherwise}
\end{cases}$$

(2)

The covariance matrix for the natural parameterization is estimated by taking minus the inverse Hessian from the reparameterized model and using the “delta method” (linearization) to recast it in the original terms.
Examples

\texttt{ldv(censor=lower,lower=0.0) hours}
# nwifeinc educ exper expersq age kidslt6 kidsge6 constant

This is a classic “tobit” model, censored below at zero.

\texttt{ldv(truncate=lower,lower=0.0,smpl=affair) y}
# constant z2 z3 z5 z7 z8

This is truncated below at zero, with the sample restricted to those with a non-zero value for \texttt{AFFAIR}.

Variables Defined by LDV

\begin{itemize}
\item \texttt{%BETA} \hspace{1cm} vector of coefficients (VECTOR)
\item \texttt{%SEESQ} \hspace{1cm} value of $\sigma^2$ (use $\text{SQRT}(%\text{SEESQ})$ to get value of \texttt{SIGMA}) (REAL)
\item \texttt{%XX} \hspace{1cm} covariance matrix of coefficients (SYMMETRIC)
\item \texttt{%STDERRS} \hspace{1cm} vector of coefficient standard errors (VECTOR)
\item \texttt{%TSTATS} \hspace{1cm} vector of $t$-statistics of the coefficients (VECTOR)
\item \texttt{%NFREE} \hspace{1cm} number of free parameters (INTEGER)
\item \texttt{%NOBS} \hspace{1cm} number of observations (INTEGER)
\item \texttt{%NREG} \hspace{1cm} number of regressors (INTEGER)
\item \texttt{%LOGL} \hspace{1cm} log likelihood (REAL)
\item \texttt{%CVCRIT} \hspace{1cm} value of the convergence criterion (REAL)
\item \texttt{%ITERS} \hspace{1cm} number of iterations completed (INTEGER)
\end{itemize}

Hypothesis Testing

You can apply the hypothesis testing instructions (\texttt{EXCLUDE}, \texttt{TEST}, \texttt{RESTRICT} and \texttt{MRESTRICT}) to estimates from \texttt{LDV}. They compute the “Wald” test based upon the quadratic approximation to the likelihood function. RATS uses the second set of formulas in Section 6.2 of the \textit{User’s Guide} to compute the statistics. Note that you cannot use the \texttt{CREATE} or \texttt{REPLACE} options on \texttt{RESTRICT} and \texttt{MRESTRICT}. 
**LINREG — Linear Regressions**

**LINREG** computes a single linear regression using least squares, weighted least squares, or instrumental variables. We discuss many of the **LINREG** options in detail in Chapters 1 and 5 of the *User’s Guide*. Note that many of the option descriptions below include references to relevant sections of these two chapters.

```
linreg( options ) depvar start end residuals coeffs
# explanatory variables in regression format
```

**Wizard**

Use the *Regressions* wizard on the *Statistics* menu, and select *OLS, Weighted Least Squares* or *Instrumental Variables* as the technique, depending upon the application.

**Parameters**

- **depvar**: Dependent variable.
- **start end**: Range to use in estimation. If you have not set a **SMPL**, this defaults to the largest common range for all the variables involved. If you use the **INSTR** option, the instruments are included in determining the default.
- **residuals**: (Optional) Series for the residuals. Omit with * if you do not want to save the residuals but want to use **coeffs**.
- **coeffs**: (Optional) Series for the coefficients. It rarely makes sense to use a **coeffs** series rather than simply using the %BETA vector.

**Options**

- **[print]/noprint**
- **vcv/[novcv]**
  These control the printing of regression output and the printing of the estimated covariance/correlation matrix of the coefficients (*User’s Guide*, Section 5.1), respectively.

- **smpl=SMPL series or formula (User’s Guide, Section 5.2)**
  You can supply a series or a formula that can be evaluated across entry numbers. Entries for which the series or formula is zero or “false” will be skipped, while entries that are non-zero or “true” will be included in the operation.

- **spread=Residual variance series (Section 5.4)**
  Use **SPREAD** for weighted least squares. The residual variances are assumed to be proportional to the indicated series.
title="title to identify estimation method"
You can use this to provide a description of the estimation method. Ordinarily this is “Least Squares,” “Instrumental Variables” or “Weighted Least Squares,” depending upon the options chosen. This is mainly used with the CREATE option.

equation=equation to estimate
Use the EQUATION option to estimate a previously defined equation. If you use it, omit the supplementary card.

define=equation to define (User’s Guide, Section 1.1.9)
frml=formula to define (Section 1.9)
These define an equation and formula, respectively, using the results of the estimation. You can use the equation/formula for forecasting or other purposes.

unravel/ [nounravel]
Substitutes for ENCODED variables (User’s Guide, Section 5.11). RATS does not print the intermediate regression (in terms of encoded variables).

cmom/ [nocmom]
Use the CMOM option in conjunction with the CMOMENT instruction (executed prior to the LINREG). LINREG takes the required cross products of the variables from the array created by CMOMENT. This has two uses:

• By computing the cross products just once, you can reduce the computations involved in repetitive regressions.
• By altering the %CMOM array before running the regression, you can implement ridge, mixed and related estimators (User’s Guide, Section 5.12).

To use CMOM, you must include all variables involved in the regression in the list for CMOMENT. RATS will ignore the LINREG start and end parameters, and carry over the range from CMOMENT.

entries=number of supplementary card entries to process [all]
This allows you to control how many of the elements on the supplementary card are processed. This can be useful in repetitive-analysis tasks, where you may want to add additional entries on each trip through a loop, for example. See Section 5.15 in the User’s Guide for details.

create/ [nocreate]
If you use CREATE, RATS does not estimate the regression. Instead, it generates the standard regression output and statistics using information that you provide. See “Options Used with Create” later in this section for details on using CREATE and the related options.
**Linreg**

\(\text{dfc=}\text{Degrees of Freedom Correction (Section 5.15)}\)

Corrects for degrees of freedom lost in the dependent variable in processing prior to the regression.

\[\text{robusterrors/}[\text{noramrobusterrors}]\]

\[\text{lags=}\text{correlated lags [0]}\]

\[\text{lwindow=}\text{neweywest/bartlett/damped/parzen/quadratic/[flat]/panel/white} \]

\[\text{damp=}\text{value of } \gamma \text{ for lwindow=damped [0.0]}\]

\[\text{lwform=}\text{VECTOR with the window form [not used]}\]

\[\text{cluster=}\text{SERIES with category values for clustered calculation} \]

When used without the INSTRUMENTS option, these permit calculation of a consistent covariance matrix allowing for heteroscedasticity (with ROBUSTERRORS) or serial correlation (with LAGS), or clustering based upon some other set of categories. See Sections 5.3, 5.4, and 5.5 of the User’s Guide and the description of the instruction MCOV for more information. Note especially the possible computational problems that may arise when using LAGS.

**Options for Two-Stage Least Squares (GMM)**

\[\text{instruments/}[\text{noinstruments}]\]

Use the INSTRUMENTS option to do two-stage least squares. You must set your instruments list first using the instruction INSTRUMENTS.

\[\text{wmatrix=}\text{weighting matrix } [(Z'Z)^{-1}]\]

\[\text{optimalweights/}[\text{nooptimalweights}]\]

\[\text{update=}\text{none/once/continuous}\]

This controls the updating of the weighting matrix. You can directly input a weight matrix with the WMATRIX option. If you don’t, the weight matrix is initialized as the \( (Z'Z)^{-1} \) matrix which gives two-stage least squares. If you use OPTIMALWEIGHTS or UPDATE=CONTINUOUS (they’re synonyms), the weight matrices are recomputed after every iteration using the choices you have made from the next group of options. Note, however, that the default number of iterations (governed by the ITERATIONS option) is just one. For a linear model, the coefficients can be solved without iterations given a weight matrix, so an iteration here means another calculation of the weight matrix.

\[\text{[zudep]/nozudep}\]

\[\text{lags=}\text{correlated lags [0]}\]

\[\text{lwindow=}\text{neweywest/bartlett/damped/parzen/quadratic/[flat]/panel/white} \]

\[\text{damp=}\text{value of } \gamma \text{ for lwindow=damped [0.0]}\]

\[\text{lwform=}\text{VECTOR with the lag window form [not used]}\]

\[\text{cluster=}\text{SERIES with category values for clustered calculation} \]

Use these to select the type of correlation assumed for the \( Z'u \) process. See the description of MCOV for more information.
**iterations=** iteration limit [1]

**cvcrit=** convergence limit [.00001]

**trace/[notrace]**

ITERATIONS sets the maximum number of iterations, which here means recalculations of weighting matrices. The default is to do just one recalculation.

**zumean=** VECTOR of means of moment conditions [zeros]

This allows you to supply a VECTOR (with dimensions equal to the number of instruments) with a set of known (non-zero) means for $E(Z'u)$. By default, RATS sets these to zero.

**center/[nocenter]**

CENTER adjusts the formula for the weight matrix to subtract off the (sample) means of $Z'u$, which may be non-zero for an overidentified model. See the description of MCOV for more information.

**jrobust=** statistic/[distribution]

You can use this option to adjust the J-statistic specification test when the weighting matrix used is not the optimal one. See Section 7.8 in the User’s Guide for more information.

**Examples**

```plaintext
linreg(define=foodeq) foodcons / resids
# constant dispinc trend
```

regresses FOODCONS on DISPINC and TREND, saves the residuals as RESIDS, and creates equation FOODEQ.

```plaintext
linreg(robusterrors) gdp 1955:1 2002:4
# constant m1{-4 to 8}
```

regresses GDP on 4 leads to 8 lags of M1 and corrects the covariance matrix for possible heteroscedasticity.

```plaintext
instruments constant mdiff{0 1} govt{0 1} $
    invest{1} cons{1 2} gnp{1 2} rate{1 to 5}
linreg(inst,frml=conseq) cons
# constant gnp cons{1}
```

computes two-stage least squares estimates of CONS on GNP and lagged CONS and defines the FRML CONSEQ from the results.
Variables Defined by LINREG

- \%BETA: Coefficient vector (VECTOR)
- \%DURBIN: Durbin-Watson statistic (real)
- \%JSIGNIF: Significance of \%JSTAT (real)
- \%JSTAT: Test statistic for overidentification for instrumental variables (real)
- \%LOGL: Normal log likelihood (real)
- \%MEAN: Mean of dependent variable (real)
- \%NDF: Degrees of freedom (integer)
- \%NFREE: Number of free parameters (integer)
- \%NOBS: Number of observations (integer)
- \%NREG: Number of regressors (integer)
- \%RBARSQ: Adjusted \( R^2 \) (real)
- \%RESIDS: Series containing the residuals (series)
- \%RSQUARED: Centered \( R^2 \) (real)
- \%RSS: Residual sum of squares (real)
- \%SEESQ: Standard error of estimate squared (real)
- \%STDERRS: Vector of coefficient standard errors (VECTOR)
- \%TRSQ: No. of observations times raw \( R^2 \) (real)
- \%TRSQUARED: No. of observations times centered \( R^2 \) (real)
- \%TSTATS: Vector containing the \( t \)-stats for the coefficients (VECTOR)
- \%QSIGNIF: Significance level of \( Q \)-statistic (real)
- \%QSTAT: \( Q \)-statistic (real)
- \%RHO: First lag correlation coefficient (real)
- \%UZWZU: \( \hat{u}'ZW\hat{u} \) for instrumental variables (real)
- \%VARIANCE: Variance of dependent variable (real)
- \%WMATRIX: Final weight matrix for GMM (SYMMETRIC)
- \%XX: Covariance matrix of coefficients, or \( X'X^{-1} \) (SYMMETRIC)

Options Used with CREATE

If you use LINREG with the option CREATE, LINREG does not compute the regression. Instead, it prints standard regression output using the information which you supply. By using LINREG(CREATE), you can get the proper \( t \)-statistics and gain access to the hypothesis testing instructions: EXCLUDE, SUMMARIZE, TEST, RESTRICT and MRESTRICT.

You can use the following LINREG options with CREATE:

- lastreg/[nolastreg]
  With LASTREG, RATS uses the regressors from the preceding regression. You do not need to include a supplementary card.

- coeff=VECTOR of coefficients [default: \%BETA]
- covmat=SYMMETRIC array covariance matrix [default: \%XX]
  The COEFF option supplies the vector of coefficients and the COVMAT option supplies the matrix which will become the \%XX matrix. You may not dimension
%BETA or %XX (the default COEFF and COVMAT arrays) yourself. However, once you have completed a regression, you can use matrix instructions to alter them.

**form=[ftest]/chisquared**

If you change %XX or use COVMAT, and the new matrix is itself an estimate of the covariance matrix, use the option FORM=CHISQUARED to switch the formulas those based upon $F$ and $t$ to those based upon $\chi^2$ and Normal.

**regcorr=number of restrictions**

Use this option if you have computed the coefficients subject to restrictions. This allows LINREG to compute the proper degrees of freedom.

**residuals=input residuals series**

This option allows you to provide your own series of residuals. RATS uses the input residuals series in computing the summary statistics.

**equation=equation to use**

You can use the EQUATION option as an alternative to a supplementary card to input a regression which differs from the preceding one (so you can’t use LASTREG). The equation should be an equation which you have already defined—it supplies the list of explanatory variables and dependent variable. Use the COEFF and COVMAT options to input the coefficients and covariance matrix.

**Example Using CREATE**

```
mcov(matrix=b,lastreg) / f
mcov(matrix=a,lastreg,nosquare) / fprime
compute %xx=%mqform(b,inv(a))
linreg(create,form=chisquared,lastreg)
```

This replaces %XX with a computed covariance matrix and reprints the regression.
List

LIST,CARDS — Shorthand for Sets of Supplementary Cards

Instructions such as GRAPH and SCATTER require sets of supplementary cards whose entries are often linked by a simple formula. In many cases, you can use the CARDS and LIST instructions to replace a large number of supplementary cards with just two lines.

```
list index = list of values
instruction requiring supplementary cards
cards supplementary card fields written in terms of index
```

Parameters

- **index**: The index variable, such as IEQN in the example below. It should be followed by at least one blank, then the =.
- **list of values**: The set of values (numbers or series) the index is to take. You can use a VECTOR of INTEGERS to form part or all of the list.
- **CARDS fields**: The fields from the standard supplementary card for the instruction, written in terms of the index.

Description

CARDS generates a supplementary card for each element in the list of values in turn, by setting index to the value and evaluating the CARDS fields. The easiest way to explain how the procedure works is with an example:

```
declare vector[series] imp(7)
impulse 7 48 1 vcv
# 1 imp(1) 1 1
# 2 imp(2) 1 2
# 3 imp(3) 1 3
# 4 imp(4) 1 4
# 5 imp(5) 1 5
# 6 imp(6) 1 6
# 7 imp(7) 1 7
```

Notice the pattern in the supplementary cards. With CARDS and LIST, you can replace this with

```
declare vector[series] imp(7)
list ieqn = 1 to 7
impulse 7 48 1 vcv
cards ieqn imp(ieqn) 1 ieqn
```

LIST is an actual RATS instruction, while CARDS is a line which replaces the set of supplementary cards.
Comments

Only one LIST is active at a given time. You can set up one LIST and use it for many instructions, for instance,

\begin{verbatim}
list ieqn = 1 to 6
sur 6
cards ieqn
sur 4 / equate 3  # 5 and 6 won't be used since the SUR says 4 equations
cards ieqn
\end{verbatim}

but you can't set up a LIST IEQN = ... and then a LIST ISER = ... and then use IEQN on a CARDS. The second LIST instruction deactivates the first.

Additional Supplementary Cards

If there are any additional supplementary cards, such as the one the one needed with the INPUT option for FORECAST, include them after CARDS. For instance,

\begin{verbatim}
declare vector[series] forecast(8)
list ieqn = 1 to 8
forecast(input,print,steps=24,from=2007:1)  8
cards ieqn forecast(ieqn)
# .05 .03 0.0 0.0 0.0 0.0 0.0 0.0
\end{verbatim}

which replaces

\begin{verbatim}
declare vector[series] forecast(8)
list ieqn = 1 to 8
forecast(input,print,steps=24,from=2007:1)  8
# 1 forecast(1)
# 2 forecast(2)
# 3 forecast(3)
# 4 forecast(4)
# 5 forecast(5)
# 6 forecast(6)
# 7 forecast(7)
# 8 forecast(8)
# .05 .03 0.0 0.0 0.0 0.0 0.0 0.0
\end{verbatim}

Applicability

You can use CARDS for the main set of supplementary cards for the forecasting instructions FORECAST, STEPS, IMPULSE, ERRORS, SIMULATE and HISTORY; and the instructions GRAPH, SCATTER, SUR, and SMODIFY.
LOCAL — Declaring Local Variables

**LOCAL** is used in a **PROCEDURE** or **FUNCTION** to declare variables that will be local to that procedure. It is similar to **DECLARE**, which is used to declare global variables.

```
local type list of names
```

**Parameters**

- **type**: Variable type that you want to assign to these variables. Local variables can be any of the data types supported by RATS. See Section 4.3 of the **User’s Guide**.

- **list of names**: The list of local variables you want to define.

**Description**

Any variable (other than a procedure or function parameter) in a RATS program is a **global** variable unless you state otherwise. You can use a global variable within any part of a program, including **PROCEDURES** and **FUNCTIONS**. Global variable names must be unique—you cannot define two different global variables with the same name (even if they are of different types) within a particular program. Also, within a particular program or session, you cannot redefine an existing global variable as a different type. For instance, you cannot redefine a **VECTOR** as a **REAL**.

Procedures and functions, however, may have **local** variables as well as globals. These are only recognized within the procedure which defines them. RATS keeps global and local variables completely separate, so a local variable can have the same name as a global one. If a name conflict arises within a procedure, the local variable is used.

If you plan to use a single **PROCEDURE** or **FUNCTION** in many different programs, it is a good idea to write the procedure using only parameters, local variables, procedure options, and variables that RATS defines. That way there is no possibility that your procedure will conflict with the names of global variables used in the main program.

You declare a local variable using **LOCAL**. **LOCAL** is quite similar to **DECLARE**.

Note that RATS does not release the memory space for local series and local arrays when you exit the procedure. If you find it necessary to release the space before returning, use the instruction **RELEASE**.

**Examples**

```
local integer  i  j
local series   s1
```
LOG — Taking the Log of a Series

**LOG** creates a series as the (natural) logs of the entries of another series.

```
log  series  start  end  newseries  newstart
```

**Wizard**

You can use the *Transformations* wizard on the *Data* menu.

**Parameters**

- `series`: Series to transform.
- `start` `end`: Range to transform. If you have not set a **SMPL**, this defaults to the defined range of `series`.
- `newseries`: Series for the result. By default, `newseries=series`.
- `newstart`: Starting entry of new series. The default is `newstart=start`.

**Examples**

The first instruction takes the natural log of `GNP` and puts it into the series `LOGGNP`. The second replaces the series `FORECAST` by its log.

```plaintext
log  gnp  /  loggnp
log  forecast
```

Here is how you would do these same operations using **SET**:

```plaintext
set loggnp = log(gnp)
set forecast = log(forecast)
```

You can use a **DOFOR** loop to take logs of multiple series. Here, we use the `%S` and `%L` functions to save the logs into new series by appending the prefix `LOG` to the existing series names, producing `LOGUSGDP`, `LOGJPNGDP`, `LOGCANGDP`, and `LOGGBRGDP`.

```plaintext
dofor  ser = usgdp  jpngdp  cangdp  gbrgdp
    log  ser  /  %s("log”+%l(ser))
end
```

**Missing Values, etc.**

Any missing value or non-positive entries will produce a missing value as the result.

**See Also . . .**

- **EXP**: Takes the exponential (inverse log) of a series.
- **CLN**: Takes the natural log of a **complex** series.
- **LOG (x)**: Returns the log of a single real value.
- **%LOG (x)**: Takes the log of each element of a matrix.
# Loop

## LOOP — Loop Forever

**LOOP** is a more general looping instruction than **WHILE** and **UNTIL**. It loops between the **LOOP** and **END** lines indefinitely unless a **BREAK** instruction is executed.

```
loop (no parameters)
  block of instructions which are executed repeatedly
end loop
```

### Description

**LOOP** repeatedly executes the block of instructions between the **LOOP** and **END LOOP** statements. You decide where and when to execute **BREAK** instructions to terminate the loop. **BREAK** is the only way out of a **LOOP**.

**LOOP** is most valuable when you make the continuation decision in the *middle* of the block, rather than at the beginning or end. It is good programming practice to use **WHILE** or **UNTIL** where possible, since they make it easier to follow the program flow.

### Example

```
loop
  menu "What Next?"
    choice "Specify Model"
      source specify.src
    choice "Estimate Model"
      source estimate.src
    choice "Do Forecasts"
      source forecast.src
    choice "Quit"
      break
  end menu
end loop
```

This repeatedly offers a set of choices, until the person using the program selects Quit. The **BREAK** after the Quit choice breaks control out of the loop.

### See Also . . .

- **BREAK** Breaks control out of a loop.
- **NEXT** Skips to the top of a loop.
- **DO** Looping over an index.
- **DOFOR** Looping over a list of items.
- **WHILE** Conditional looping.
- **UNTIL** Conditional looping.
- *UG*, Section 16.1 The RATS Compiler
LQPROG — Linear and Quadratic Programming

LQPROG solves linear and quadratic programming problems.

```
lqprog( options )  x  c  A  b  Q
```

The linear programs take the form:

minimize  \( c'x \)

subject to:  \( A_c x = b_c, A_i x \leq b_i, A_g x \geq b_g \)

\[ x_i \geq 0 \text{ for all } i=1,...,nvar \]

where, by design, \( b_c \geq 0, b_i \geq 0, b_g \geq 0 \)

\( nvar \) is the number of unknowns. Quadratic programs have the same constraints (except, perhaps, on the \( x \)'s), and, subject to those constraints, solve the problem:

minimize  \( \frac{1}{2} x'Qx + c'x \)

To choose quadratic programming, supply a \( Q \); for linear programming, don't provide a \( Q \). You can supply values for the \( c, A, b, \) and \( Q \) matrices using parameters or options. The newer option form is preferred. If you have a program using the parameters, see the description of the option with the same name below.

Parameters

\( x \) VECTOR into which the solution will be saved. This corresponds to the \( x \) matrix (the matrix of unknowns) in the technical discussions later in this section. You do not need to declare or dimension this array ahead of time.

Options

\( c=\) VECTOR of coefficients for \( c \) matrix [unused]
\( q=\) SYMMETRIC array of quadratic coefficients for \( Q \) matrix [unused]

These input the matrices controlling the objective function.

If there is a \( Q \) matrix, LQPROG does quadratic programming; otherwise it does linear programming.
a=RECTANGULAR array of coefficients for A matrix [unused]
b=VECTOR of constraining values for the b matrix [zeros]

These supply the coefficients on the equality and inequality constraints. The equality constraints should be listed first, followed by ≤ constraints, and finally any ≥ constraints.

equalities=number of equality constraints [0]
ge=number of ≥ constraints [0]

By default, all constraints are assumed to be of the form Ax ≤ b. You can use the EQUALITIES and GE options if you wish to include equality constraints and constraints of the form Ax ≥ b. Any equality constraints should be listed first in the A and b arrays, followed by any Ax ≤ b constraints. List any Ax ≥ b constraints last.

nneg=number of x components which must be non-negative [all]

This can be used only with quadratic programming. You can divide the x vector into a first group which must be non-negative and a second group which are unconstrained.

iterations=number of iterations  [max of 20 or # of constraints]
cvcrit=Convergence criterion  [10^-8]

ITERATIONS sets the maximum number of iterations that will be performed. If the procedure fails to converge in the specified number of iterations, re-execute the instruction with a higher iteration limit. CVCRIT controls the criterion value by which RATS determines whether or not the process has converged to a solution. RATS will continue iterating until either the change in the dot product of the gradient is smaller than the CVCRIT value, or until the iterations limit is reached. Note that the convergence is not measured by the coefficients as it is in most other numerical optimizations.

[trace]/notrace

If you use TRACE, RATS will issue a report on the progress of the estimation after each iteration.

[print]/noprint

This controls the printing of the results.

feasible/[nofeasible]

By default, LQPROG computes the full solution of the problem supplied. If you use the FEASIBLE option, RATS will only compute the initial feasible solution.
Description

**LQPROG** can solve both linear programming problems and quadratic programming problems. Examples of each are shown below. For details on using these instructions, see Section 12.5 in the *User’s Guide*.

Note that, while RATS uses a fairly good general-purpose algorithm for solving these problems, it is not designed to handle very large problems with hundreds of variables or constraints.

Variables Defined

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%FUNCVAL</td>
<td>Final value of the function (REAL)</td>
</tr>
<tr>
<td>%LAGRANGE</td>
<td>VECTOR of Lagrange multipliers on the constraints</td>
</tr>
</tbody>
</table>

Linear Programming Example

minimize $-2x_1 - 4x_2 - x_3 - x_4$

subject to

$x_1 + 3x_2 + x_4 = 4$

$2x_1 + x_2 \leq 3$

$x_2 + 4x_3 + x_4 \leq 3$

$x_i \geq 0$

We have one equality constraint, so we set EQUALITIES to 1.

```
declare rectangular amat(3,4)
declare vector cvec(4) bvec(3)
compute amat = || 1, 3, 0, 1 | 2, 1, 0, 0 | 0, 1, 4, 1 ||
compute bvec = || 4, 3, 3 ||
compute cvec = || -2, -4, -1, -1 ||
lqprog(equalities=1,a=amat,b=bvec,c=cvec) xout
```

The solution is stored into the VECTOR called XOUT.

```
display xout
  1.0000   1.0000   0.5000   0.0000
```
Quadratic Programming Examples

This solves a simple quadratic programming problem:

minimize \[ x_1^2 + x_2^2 + x_3^2 + x_1x_3 - 10x_1 - 4x_3 \]
subject to
\[ 2x_1 + x_2 + x_3 = 2 \]
\[ x_1 + 2x_2 + 3x_3 = 5 \]
\[ 2x_1 + 2x_2 + x_3 \leq 6 \]
\[ x_i \geq 0 \]

```
declare rectangular amat(3,3)
declare symmetric qmat(3,3)
declare vector cvec(3) bvec(3)
compute amat = || 2, 1, 1 | 1, 2, 3 | 2, 2, 1||
compute bvec = || 2, 5, 6 ||
compute cvec = || -10 , 0, -4 ||
compute qmat = || 2 | 0, 2 | 1, 0, 2 ||
lqprog(equalities=2,c=cvec,a=amat,b=bvec,q=qmat) results
```

which produces the output:

QPROG converges at 3 iterations
minimum = -5.480000
solution x =
0.2000    0.0000    1.6000

The example below demonstrates a simple portfolio optimization routine (see Section 12.5 for a more extensive example). This finds the global minimum variance portfolio. \( N \) is the number of returns, and \( \text{ExpRet} \) and \( \text{CovMat} \) are the expected values and covariance matrix of the returns. To modify this for any number of assets, you simply need to change those three variables accordingly.

```
compute N=3
compute [vect] ExpRet = ||.15,.20,.08||
compute [symm] CovMat = ||.20|.05,.30|-.01,.015,.1||
* * Compute the minimum variance portfolio, and its expected return *
compute units=%fill(1,N,1.0)
lqprog(q=CovMat,a=units,b=1.0,equalities=1) x
compute minr=%dot(x,expRet)
compute maxr=%maxvalue(expRet)
```
MAKE — Creating an Array from Data Series

MAKE creates an array from the entries of data series. You can use lags and leads in the series list on the supplementary card.

```
make( options )  array  start  end
# list of variables in regression format  (omit with EQUATION or LASTREG options)
```

Parameters

- **array**: RECTANGULAR array to create. You do not need to declare or dimension it.
- **start**  **end**: Range of entries to use. If you have not set a SMPL, this defaults to the largest range over which all the variables are defined.

Two additional parameters used in previous versions to store the number of observations and the number of variables have been deprecated. MAKE now automatically saves these values in the variables %NUMOBS and %NVAR.

Description

MAKE creates the array from entries start to end of the listed series. It will automatically create and dimension the array. By default, each column represents a different variable and each row represents an observation in the array. This is the standard arrangement for an X array in \( y = X\beta \). You can use the TRANSPOSE option to get \( X' \).

Options

- **transpose/[notranspose]**
  Use TRANSPOSE if you want the transpose of the X array—each column is an entry and each row a variable. If you want to use the instruction OVERLAY to isolate a subset of the entries, you need to use TRANS.

- **equation=**name/number of equation for variables
- **lastreg/[nolastreg]**
- **depvar/[nodepvar]**

  With EQUATION, MAKE uses the explanatory variables from the equation as its list of variables, while LASTREG takes the explanatory variables from the last regression (or similar instruction). With either option, you can use DEPVAR to include the dependent variable as well, as the final column.

- **smpl=**SMPL series or formula (User’s Guide, Section 5.2)

  Includes only entries for which the series or formula returns a non-zero or “true” value. The created matrix is compressed to include only the good rows, so there may no longer be a one-to-one correspondence between matrix rows and entries.
mult=series by which to multiply observations
spread=series by which to divide observations (square root)
scale=value by which to scale matrix

These three options are mutually exclusive.

• With \texttt{MULT}, \texttt{MAKE} multiplies each observation in the matrix by the corresponding entry from the series.
• With \texttt{SPREAD}, it divides each observation in the matrix by the square root of the entry of the spread series. \texttt{SPREAD} has the same basic function as it does for \texttt{LINREG}.
• With \texttt{SCALE}, it multiplies all entries by the single value.

\texttt{panel=RECTANGULAR array of data from panel series}

You can use this option to split one or more panel data series into a $T \times (N \cdot K)$ array, where $T$ is the number of time periods per individual, $N$ is the number of individuals, and $K$ is the number of series.

\textbf{Examples}

\begin{verbatim}
make r 1954:1 2002:4
  # constant x1 x2 x3
creates a 196x4 matrix R.

make(transpose,mult=resid,lastreg) xumat 1947:1 2002:4
creates the matrix with $X_t' u_t$ in each column, where $X$ is the set of regressors from the last regression and $u_t$ is the series \texttt{RESID}.

inquire(regressorlist,valid=common) n1 n2
  # constant x1 x2 x3 y1
make(smpl=common) x n1 n2
  # constant x1 x2 x3
make(smpl=common) y n1 n2
  # y1
compute [vector] b=inv(tr(x)*x)*(tr(x)*y)
does a least squares regression using matrix instructions. We include the \texttt{SMPL} option because each \texttt{MAKE}, on its own, might determine a different default range.
\end{verbatim}

\textbf{Using TRANSPOSE}

\texttt{MAKE} with the option \texttt{TRANSPOSE} creates the matrix $X'$ rather than $X$. This is very useful if you want to treat a subset of the entries of the $X$ matrix as another matrix. The instruction \texttt{OVERLAY} overlays consecutive columns of a matrix, because RATS stores matrices by columns. To isolate entries, each must be in a column, not a row, which is precisely how \texttt{TRANSPOSE} creates the matrix.
In the example below, \( X_{1\text{MAT}} \) is equal to the matrix of the first 99 columns (entries) of \( X_{\text{MAT}} \) and \( Z_{1\text{MAT}} \) equal to the last 99 columns.

```plaintext
make(trans) xmat 1 100
# constant v1 v2
dec rect x1mat z1mat
overlay xmat(1,1) with x1mat(3,99)
overlay xmat(1,2) with z1mat(3,99)
```

**“Unmaking” a Matrix**

Use \texttt{SET} or a sequence of \texttt{SET} instructions to set series equal to the rows (or columns) of matrices. For example,

```plaintext
set x2 1 100 = a(t,1)
```
sets \( X_2 \) equal to the first 100 entries of column 1 of \( A \). Remember, however, that the \( T \) subscript runs over the \texttt{start} to \texttt{end} range on \texttt{SET}, which is not necessarily 1,...,dimension. The following (trivial) example shows how to fix the subscript in the \texttt{SET} to keep the output series aligned with the input. \( XX \) will be the same as \( X \).

```plaintext
make r 1948:1 2002:12
# constant x
set xx 1948:1 2002:12 = r(t-(1948:1-1),2)
```

If the size of the matrix can change from one use to another, you should “unmake” into a \texttt{VECTOR} of \texttt{SERIES}. For instance, the following creates a vector of \texttt{NCOLS} series from the columns of \( R \).

```plaintext
compute ncols=%cols(r)
dec vect[series] columns(ncols)
do i=1,ncols
   set columns(i) 1 %rows(r) = r(t,i)
end do i
```

**Missing Values**

If any series has a missing value at an entry, \texttt{MAKE} drops that entry from the constructed array. If you are using \texttt{MAKE} to create several arrays which must have identical sets of entries, use a common \texttt{SMPL} to deal with missing values.

```plaintext
smpl(reglist)
# constant v1{0 to 3}  
```

\texttt{includes all variables}

**Variables Defined**

- \texttt{%NUMOBS}  Number of observations (\texttt{INTEGER})
- \texttt{%NVAR}  Number of variables (\texttt{INTEGER})
MAXIMIZE — General Maximization

**MAXIMIZE** estimates the parameters of a fairly generalized type of maximization problem. It can be used for maximum likelihood estimation of models for which RATS does not have a direct estimator.

\[
\max_{\beta} G(\beta) = \sum_{i=1}^{T} f(y_i, X_i, \beta)
\]

where \( f \) is a RATS formula (FRML). Before you can use it, you must:

- Set the list of free parameters using **NONLIN**
- Create the explanatory formula \( f \) using **FRML**
- Set initial values for the parameters (usually with **COMPUTE** or **INPUT**)


**Parameters**

- **frml** This is the FRML (created with the FRML instruction) which computes the function \( f \).
- **start** **end** the estimation range. If you have not set a **SMPL**, it defaults to the range over which **frml** can be computed.

Some **frmls** (particularly recursively defined formulas such as GARCH models) require you to specify a range, either with **SMPL** or **start** and **end**. If you get the error message:

```
Missing Values ... Leave No Usable Data Points
```

your range either has to be set explicitly or, if you have already specified one, it must be changed to match the available data.

- **funcval** (Optional) series for the computed values of \( f(y_i, X_i, \beta) \).
Maximize

Options

**parmset=**PARMSET to use [default internal]
This option selects the parameter set to be estimated (User’s Guide, Section 7.6).
RATS maintains a single unnamed parameter set which is the one used for estimation if you don’t provide a named set.

**method=bhhh/[bfgs]/simplex/genetic/evaluate**
**pmethod=bhhh/bfgs/[]/simplex/genetic/evaluate**

**piters=**number of PMETHOD iterations to perform [none]

BHHH is Berndt, Hall, Hall and Hausman; BFGS is Broyden, Fletcher, Goldfarb and Shanno; SIMPLEX is the simplex algorithm; and GENETIC is a genetic search algorithm. See Chapter 7 in the User’s Guide for a technical description of these. BHHH should be used only if the function being maximized is the log likelihood (apart from additive constants). BFGS and BHHH require the formula to be twice continuously differentiable. SIMPLEX and GENETIC are derivative-free methods which can compute point estimates of the coefficients but not standard errors. With better-behaved functions, they can be helpful in refining initial guess values before applying one of the derivative-based methods.

With METHOD=EVALUATE, RATS simply evaluates the model given the initial parameter values, without trying to estimate new coefficient values.

Use PMETHOD and PITERS if you want to use a preliminary estimation method to refine your initial parameter values before switching to one of the other estimation methods. For example, to do 10 simplex iterations before switching to BFGS, you use the options PMETHOD=SIMPLEX, PITERS=10, and METHOD=BFGS.

**hessian=**initial guess for inverse Hessian (METHOD=BFGS only)
You can use this with METHOD=BFGS. Without it, MAXIMIZE will start with a BHHH matrix, but with diagonal elements only. See Section 7.2.

**startup=**FRML evaluated at period “start”
provides an expression which is computed only for the first entry of the estimation range, before the regular formulas are computed. The main advantage of this is to allow flexible initialization for recursive formulas such as in GARCH models. This accepts FRMLs of any type, not just real–valued FRMLs.

[print]/noprint

**vcv/[novcv]**

**smpl=**SMPL series or formula (User’s Guide, Section 5.2)
These are the same as for other regressions. Note that MAXIMIZE has no SPREAD option.

**title=**"title for output" ["MAXIMIZE"]
This option allows you to supply your own title to label the resulting output.
Maximize

\begin{verbatim}
iterations=iteration limit [100]
subiterations=subiteration limit [30]
cvcrit=convergence limit [.00001]
trace/[notrace]
\end{verbatim}

ITERATIONS sets the maximum number of iterations, SUBITER sets the maximum number of subiterations, CVCRIT the convergence criterion. TRACE prints the intermediate results. For METHOD=SIMPLEX, an “iteration” is actually defined as \( K \) vertex changes, where \( K \) is the number of free parameters. This makes the number of calculations per “iteration” similar to the other methods.

\begin{verbatim}
robusterrors/[norobusterrors]
lags=correlated lags [0]
lwindow=neweywest/bartlett/damped/parzen/quadratic/[flat]/panel/white
damp=value of \( \gamma \) for lwindow=damped [0.0]
lwform=VECTOR with the window form [not used]
cluster=SERIES with category values for clustered calculation
\end{verbatim}

These permit calculation of a consistent covariance matrix (Section 5.3) and/or clustered standard errors if you are doing Quasi Maximum likelihood estimation. With MAXIMIZE, the LAGS, LWINDOW, DAMP, and LWFORM are used for dealing with correlation in the gradient elements. See MCOV for details.

\begin{verbatim}
derives=VECTOR[SERIES] for partial derivatives
\end{verbatim}

This stores the series of partial derivatives of the function \( f \). The first series in the VECTOR will be the partials with respect to the first parameter displayed in the MAXIMIZE output, the second series will be the partials with respect to the second parameter, and so on.

\begin{verbatim}
reject=FRML indicating a "rejection" zone for the parameters
\end{verbatim}

If the FRML evaluates to a non-zero (“true”) value, the function is immediately assigned the missing value.

Hypothesis tests

If you have used METHOD=BFGS or METHOD=BHHH, you can test hypotheses on the coefficients using TEST, RESTRICT and MRESTRICT. Coefficients are numbered by their positions in the list set up by NONLIN. You may not use EXCLUDE and SUMMARIZE. See the previous paragraph for information about the validity of these test statistics.

Missing Values

MAXIMIZE drops any entry which cannot be computed because of missing values.
Technical Information

Chapter 7 in the User’s Guide describes the methods and how the instruction NLPAR controls some of the finer adjustments of them. MAXIMIZE uses numerical derivatives in computing the gradient. After the first iteration, the perturbations used in computing these adapt to the current estimates of the dispersion of the parameters.

As discussed in Section 7.5, the validity of the covariance matrix and standard errors produced by MAXIMIZE depend upon the functional form and the options that you choose. In general, standard errors produced using METHOD=BHHH are asymptotically correct only if the function being maximized is the log likelihood apart from additive constants. This is true even if the function you use is a 1–1 transformation of the log likelihood, and thus has an identical maximum. The default standard error calculations for BFGS are correct only under similar circumstances. If you’re maximizing a function other than the log likelihood, you should use the combination of METHOD=BFGS and ROUSTERRORS.

Variables Defined by MAXIMIZE

MAXIMIZE defines the following standard estimation variables (see LINREG)

%NREG, %NOBS  (INTEGER)
%BETA, %TSTATS, %STDERRS  (VECTOR)
%XX  (SYMMETRIC)

and the following other variables

%FUNCV AL  Final value of $G(\beta)$ (REAL)
%CONVERGE D  = 1 or 0. Set to 1 if the process converged, 0 if not.
%CVCRIT  final convergence criterion. This will be equal to zero if the subiterations limit was reached on the last iteration (REAL)
%LAGRANGE  VECTOR of Lagrange multipliers if estimating with constraints.

Examples

This estimates a probit model for a data set where there are repetitions of settings for the explanatory variables ($X_1$ and $X_2$), with REPS and SUCCESS giving the total observations and the number of successes. Note the use of the “sub-formula” to describe the function of the explanatory variables and how the PROBIT formula evaluates this just once and puts it into the variable $Z$. The use of the sub-formula allows quick changes to the basic model.

```
nonlin b0 b1 b2
frml zfrml = b0+b1*x1+b2*x2
frml probit = (z=zfrml(t)) , $success*log(%cdf(z))+(reps-success)*log(1-%cdf(z))
compute b0=b1=b2=0.0
maximize probit
```
Box-Cox models (Box and Cox (1964)) can take several forms. The only one which requires use of \texttt{MAXIMIZE} rather than \texttt{NLLS} has the dependent variable subject to a Box-Cox transformation with unknown parameter. In RATS, you can compute the Box-Cox class of transformations using the function \texttt{%BOXCOX}. The following estimates using maximum likelihood the model

\[
y_i^{(i)} = \beta_0 + \beta_1 X_i^{(i)} + u_i ; u_i \sim N(0, \sigma^2) \text{ i.i.d}
\]

Initial guess values are obtained using a linear regression, which (with an adjustment to the intercept) is a special case for $\lambda = 1$.

```
nonlin b0 b1 lambda sigmasq
frml rhsfrml = b0+b1*%boxcox(x,lambda)
frml boxcox   = (lambda-1)*log(y) + $\%logdensity(sigmasq,%boxcox(y,lambda)-rhsfrml(t))$
linreg y
  # constant x
compute sigmasq=%seesq,b0=%beta(1)+%beta(2)-1
compute b1=%beta(2),lambda=1.0
maximize(method=bfgs,robusterrors,iters=200) boxcox
```

Note: convergence can be extremely slow with these models, which is why the iterations limit is increased.

This estimates a linear regression on a panel data set allowing for heteroscedasticity among the (five) individuals. The variance parameters are in the vector \texttt{SIGSQ}, while the regression parameters come off the vector \texttt{B} generated using \texttt{FRML} with the \texttt{LASTREG} option.

```
linreg iv
  # constant f c
frml(lastreg,vector=b) zfrml
dec vector sigsq(5)
  compute sigsq=%const(1.0)
nonlin b sigsq
frml groupgls = %logdensity(sigsq(%indiv(t)),iv-zfrml)
maximize(iters=200,method=bfgs) groupgls
```

Here $Y$ given $E$ is assumed to have an exponential distribution with mean $B+E$, where $B$ is a parameter to be estimated. This is estimated by maximum likelihood using the BHHH algorithm.

```
data(format=prn,org=columns) 1 20 id y e
nonlin b
frml logl = -log(b+e)-y/(b+e)
maximize(method=bhhh) logl
```
MCOV — Consistent Covariance Matrices

**MCOV** (short for Matrix COVariogram) calculates the key building block for calculations which are robust to heteroscedasticity and autocorrelation. While these are included directly in many RATS estimation instructions, there are situations where the calculations done by **MCOV** are useful by themselves.

```plaintext
mcov( options ) start end list of resids
# list of variables in regression format
```

**Parameters**

- **start**  end  Range of entries to use. If you have not set a **SMPL**, this defaults to:
  - the range of the most recent regression, if you use the option **LASTREG**.
  - the maximum range over which all the regressors are defined, otherwise.

- **list of resids**  This is a list of one or more series of residuals. If you omit this, the u’s are dropped from the formulas later in the section.

**Options**

- **lags=correlated lags  [0]**
  This is the number of lags of autocorrelation (in the form of moving average terms) that you want included. There are certain technical problems which arise when **LAGS** is non-zero; see the discussion later in this section. For the quadratic spectral window (see the **LWINDOW** option), **LAGS** gives the bandwidth, since a full set of lags are always used with that. For other window types, note that RATS counts the number of correlated lags, which is one less than the width that is often used in descriptions of these window. **LAGS** can take non-integer values; while this feature is mainly used with the quadratic window, it can be used with the others.

- **lwindow=neweywest/bartlett/damped/parzen/quadratic/[flat]/panel/white**
  **LWINDOW** chooses the form of the lag window to be used. **NEWEYWEST** and **BARTLETT** are identical to each other, and to **LWINDOW=DAMPED** with **DAMP=1.0**. **QUADRATIC** is the quadratic spectral window. See “Technical Information” later in this section for details. **DAMP** gives the parameter γ of the window if you use **LWINDOW=DAMPED**. None of these matter if **LAGS** is zero, except for **PANEL** which is a special case of clustered calculation; it’s the equivalent of **FLAT** with **LAGS** equal to the number of time periods per individual (minus one).
lwform=VECTOR with the window form [not used]
You can use LWFORM as an alternative to LWINDOW if you want to use a window not covered by the choices in LWINDOW, or if you want to use an asymmetrical window. This must have dimension $2 \times L + 1$, with the first entry giving the lag $L$ value and the last the lead $L$ value. If your window isn’t symmetrical, you need to use the RMATRIX option to get the resulting matrix.

cluster=SERIES with category values for clustered calculation
See the description of LWINDOW=PANEL and CLUSTER later in this section.

model=(linear) MODEL providing variables [not used]
equation=equation provided variables [not used]
lastreg/[nolastreg]
depvar/[nodepvar]
instruments/[noinstruments]
These provide shortcuts for using standard sets of variables. MODEL takes the explanatory variables from a model with linear equations; EQUATION takes these from a single equation, and LASTREG uses the explanatory variables from the last regression (or similar instruction). INSTRUMENTS uses the current set of instruments as the list. If you include any of these options, omit the supplementary card. Use the option DEPVAR with MODEL, EQUATION or LASTREG if you want to include the dependent variable(s) of the equations or regression.

zumean=VECTOR of means of moment conditions [not used]
This allows you to supply a VECTOR with a set of known (non-zero) means for $E(Z'u)$. By default, RATS assumes these are zero.

center/[nocenter]
CENTER adjusts the formula to subtract off the (sample) means of $Z'u$, which may be non-zero for an overidentified model. See the Technical Information.

meanvector=VECTOR for means of moment conditions
This saves the means of the moment conditions into the VECTOR you supply. The VECTOR will be created if it doesn’t already exist.

opgstat=value for OPG statistic [not used]
Use this to get the OPG (outer-product gradient) statistic used in many LM tests. See the Technical Information for details.

[zudependent]/nozudep
ZUDEP is the default for MCOV (but not for NLSYSTEM), as there is little reason to use the MCOV instruction when the instruments and residuals aren’t dependent. See the technical information for the formula used if NOZUDEP.
**matrix=SYMМЕTRIC array for computed matrix [%CMOM]**

**rmatrix=RECTANGULAR array for computed matrix [none]**

Use **MATRIX** to put the computed matrix into a specific **SYMМЕTRIC** array. By default, **MCов** uses the array **%CMOM**. If you’ve put in an asymmetric window using **LWFORM** option, the resulting matrix won’t be a **SYMМЕTRIC**, so you **must** use **RMАTRIX**.

**print/ [noprint]**

Use **PRINT** if you want to print the computed matrix. It is printed in a table similar to the one used for the covariance matrix of regression coefficients (**User’s Guide**, Section 5.1).

**spread=Residual variance series (**User’s Guide**, Section 5.4)**

**smpl=SMPL series or formula (**User’s Guide**, Section 5.2)**

These are the same as for regressions instructions.

**[square]/nosquare**

Use **NOSQUARE** with **v=residuals** if you want to compute **Z'vZ**, rather than **Z'uuZ**. The **LAGS** option is ignored if you use **NOSQUARE**. This can only be used with a single “v” series.

**Technical Information**

As you can probably tell from the options, an **MCов** is embedded within the **ROВUST-ERRORS** options of **LINREG**, **MAXIMIZE** and other estimation instructions. Its value has diminished since these options were added directly to the estimation instructions. However, it still has its uses, as can be seen in **User’s Guide** Sections 5.3, 5.4 and 5.5. If you’re computing a weight matrix for GMM, use the inverse of the output matrix from **MCов**.

For **LAGS=L**, **u_t** = a single series of residuals, **Z** being a list of variables supplied on the supplementary card, and **w_t** a set of window weights, **MCов** produces the matrix

\[
(1) \quad \sum_{l=-L}^{L} \sum_{t} w_t \left( Z'_t u_t u_{t-l} Z_{t-l} \right)
\]

Use the **ZUMEAN** option when the model assumes that the moment conditions **EZ_t'u_t** have a non-zero mean **μ_{Z_t}**. It replaces (1) with

\[
(2) \quad \sum_{l=-L}^{L} \sum_{t} w_t \left( Z_t u_t - \mu_{Z_t} \right)' \left( Z_{t-l} u_{t-l} - \mu_{Z_{t-l}} \right)
\]

The **CENTER** option makes a similar adjustment, but uses the sample mean rather than a hypothesized mean. This is recommended by Hall (2000) for computing the weight matrix to be used in testing overidentifying restrictions.
LWINDOW option

If LAGS is non-zero, and the \( w \)'s are all one (LWINDOW=FLAT), the matrix in (1) may fail to be positive semi-definite, which can produce invalid standard errors. The various window types are provided to avoid this. The formulas for windows are most easily described by defining

\[
(3) \quad z = \frac{|z|}{L+1}
\]

Except for the quadratic window, all the window weights are zero when \(|z|>1\). Note that the phrase \textit{bandwidth} for these windows usually means \(L+1\), not \(L\) itself. BARTLETT and NEWEYWEST are identical (Bartlett is the historical name for the window in spectral analysis). For more information on lag windows, see Hamilton (1994), pages 281-284.

\[
(4) \quad \text{LWINDOW=FLAT} \quad w(z) = 1
\]

\[
(5) \quad \text{LWINDOW=NEWEYWEST} \quad w(z) = 1 - z
\]

\[
(6) \quad \text{LWINDOW=DAMPED} \quad w(z) = (1 - z)^\gamma, \quad \gamma \text{ is the DAMP option.}
\]

\[
(7) \quad \text{LWINDOW=PARZEN} \quad w(z) = \begin{cases} 
1 - 6z^2 + 6z^3 & 0 \leq z \leq \frac{1}{2} \\
2(1 - z)^2 & \frac{1}{2} \leq z \leq 1
\end{cases}
\]

\[
(8) \quad \text{LWINDOW=QUADRATIC} \quad w(z) = \frac{3}{(6\pi z/5)^2} \left[ \frac{\sin(6\pi z/5)}{6\pi z/5} - \cos(6\pi z/5) \right]
\]

LWINDOW=Panel, CLUSTER options

When \(L\) is equal to the number of time series observations (thus allowing for arbitrary patterns of serial correlation), (1) with unit weights is the same as

\[
(9) \quad \left( \sum_{i} Z_i^\prime u_i \right) \left( \sum_{i} Z_i^\prime u_i \right)^\prime
\]

This is positive semi-definite, because it's a matrix times its transpose. For a single time series, however, it's not very useful, because it has rank one. However, if you add these up across a large number of individuals or categories, you get a full rank, consistent estimator (big \(N\), small \(T\)). See, for instance, Wooldridge (2002), section 10.5.4. If this is used in computing a covariance matrix, you get \textit{clustered standard errors}. LWINDOW=Panel will do this calculation (clustering on individuals) for a panel data set. If you want to do the clustering on a variable other than this, use the option CLUSTER=series with clustering categories. This series should have a unique value for each category, and should have at least as many categories as you have variables (regressors) if you want a positive definite result, and should have \textit{many} more if you want consistency.
Multivariate Residuals

If you have more than one series of residuals, the matrix is computed as

\[
(10) \quad \sum_{i=-L}^{L} w_i \left( (u_i \otimes Z_i) \left( u_{i-1} \otimes Z_{i-1} \right) \right)
\]

with the appropriate adjustments for ZUMEAN and CENTER options. This arrangement (blocking the matrix by equation) matches with that expected for a weight matrix by the instructions NLSYSTEM and SUR. Note that this matrix will likely not be full rank if the size of \(Z\) times the size of \(u\) is greater than the number of data points.

The NOZDEP option can only be used with LAGS=0, and only with NOCENTER. It computes the special case of

\[
(11) \quad \Sigma \otimes ZZ
\]

OPGSTAT Option (LM Tests)

The test statistics for many LM tests take the form

\[
(12) \quad (\sum Z'u)\left(\sum Z'u^2Z\right)^{-1}(\sum Z'u)
\]

where \(u\) is the series of residuals or generalized residuals, and \(Z\) the regressors augmented by some other variables. This tests for a zero value of \(\sum Z'u\). You can compute this using MCOV. Just provide the \(u\) and \(Z\) series and use the OPGSTAT option. See the second example below.

Examples

```r
m cov(l window=bartlett, lags=lags) startl+1 endl resids
# constant
```

computes an estimate of the spectral density of RESIDS at frequency zero, using "LAGS" lags. The computed matrix (which here will be 1x1) will be in %CMOM.

```r
m cov(opgstat=sclm) / u
# %reglist() u{1}
cdf(title="LM Test for Serial Correlation") chisqr sclm 1
```

This computes an LM test using the OPGSTAT option as described above.

Variables Defined

In addition to %CMOM, MCOV defines %NCMOM (dimensions of the matrix) and %NOBS (number of observations).
MEDIT opens a spreadsheet-style editing window that allows you to view, edit, or enter data into RECTANGULAR, SYMMETRIC, or VECTOR type arrays. If you list more than one array, MEDIT opens a separate editing window for each array.

```
medit(options) list of arrays
```

**Parameters**

*list of arrays* This is the list of arrays you want to view or edit. These can be any combination of real-valued RECTANGULAR, SYMMETRIC, or VECTOR arrays. You **must** declare and dimension these before using them with MEDIT. You should also initialize the entries of the array(s) to some valid real number (such as 0.0).

MEDIT currently cannot be used with other types of arrays (that is, arrays of integers, arrays of labels, etc.).

**Options**

*edit*/noedit

With *EDIT*, the user can modify the values in the array(s). With *NOEDIT*, the user can view, but not change, the values in the array.

*modal*/[nomodal]

With *MODAL*, an MEDIT editing window functions like a modal dialog box. Execution of any pending instructions is suspended until the user closes the editing window(s) opened by MEDIT. Also, RATS is essentially “locked” into local mode, so no new instructions can be executed while the editing window is open. The user can, however, switch to other windows or select menu operations. This setting is very useful in interactive procedures where you need the user to set the array before continuing with the rest of the program.

With *NOMODAL*, RATS continues executing any pending instructions after opening the editing window(s). The user can also enter new instructions in the input window without first closing the editing window(s).

*window=string for window title*

Use this option to supply a title for the editing window(s); this title will be used for all editing windows when editing more than one array. By default, RATS labels each window with the name of the array associated with the window.
vlabels=VECTOR of STRINGS for row labels
hlabels=VECTOR of STRINGS for column labels
You can use these options to supply your own labels for the rows and columns in the editing window(s). By default, columns and rows are labeled with integer row or column numbers. If you include several arrays on a single instruction, RATS will use the same set of VLABELS and HLABELS for all of the editing windows.

picture=picture clause for data [none]
A picture clause is most helpful when you’re using MEDIT to display data rather than having the user enter or edit it. MEDIT, by default, will use “best” representation that fits in 15 digits. This not only may show many more digits than are statistically reliable, but also will take up more room per number than a smaller format, so fewer values will be visible at one time. For instance, the option PICTURE="*.###", will limit the display to three digits right of the decimal. See DISPLAY for more on picture codes.

prompt="string for window prompt"
This adds the text string you supply to the upper left corner of the MEDIT dialog box. You can use the characters \ to put a line break in the string.

select=VECTOR[INTEGER] of selections
stye=[rows]/one/byrow/bycol
If you use the SELECT option, MEDIT will display the data, and the user can select items from the matrix. When the window is closed, the VECTOR[INTEGER] that you provide on the option will be filled with information regarding the selection, as described below.

You use STYPE to control how selections are made. With the default of STYPE=ROWS, the user can select one or more rows. The array provided using the SELECT option will have dimension equal to the number of selections made, and will list the rows selected using numbers 1,...,n. If STYPE=ONE, it will have dimension 2 with array(1)=row and array(2)=column. If STYPE=BYROW, the user selects one cell per row. The SELECT array will have dimension equal to the number of rows, with array(i) set equal to the column selected in row i. STYPE=BYCOL is similar, but one cell per column is selected.

Note that this type of selection is probably done more easily now using the DBOX instruction with the MATRIX, MSELECT and STYPE options.
**Medit**

**Description**

*MEDIT* can be particularly useful in writing interactive procedures which require the user to enter data into arrays. It provides a more useful and intuitive way to enter data than the alternative methods using **INPUT**, **READ** or **ENTER**.

Note: *MEDIT* currently will *not* work in batch mode, and thus will not work in batch-mode only versions of RATS.

**Examples**

This simple example uses *MEDIT* as a way to type values into a matrix:

```plaintext
declarerectangularr(10,10)
meditr
```

This code below estimates an ARIMA model for various combinations of AR and MA lags, and computes Akaike and Schwarz criterion values for each model. The resulting criteria values are then displayed in an *MEDIT* window.

```plaintext
do mas=0,3
   do ars=0,3
      boxjenk(constant,diffs=1,ma=mas,ar=ars,maxl,$
          iters=100,noprint) logyen 1973:5 1994:12
      @regcrits(noprint)
      compute aic(ars+1,mas+1)=%aic
      compute sic(ars+1,mas+1)=%sbc
   end do ars
end do mas
medit(hlabels=||"0","1","2","3"||,vlabels=||"0","1","2","3"||,$
       picture="*.*##",noedit) aic sic
```

The resulting *MEDIT* window will look something like this:
**MENU,CHOICE — Pop Up Menu Selections**

`MENU` and `CHOICE` help you control a menu and dialog driven procedure. They present to the user a dialog box with mutually exclusive choices. `MENU` and `CHOICE` are useful when you need the user to select one of a (small) set of choices for an operation, or make a single choice of action. However, the `USERMENU` instruction is preferable for setting up the main control of your procedure; `SELECT` is useful when the list of choices is long or variable, or where the user can select more than one item; and `DBOX` offers greater flexibility.

```plaintext
menu  Menu description string
  choice  Choice identifier string
    instruction or block of instructions executed if this choice is made
end menu
```

**Parameters**

- **Menu string** RATS displays the menu string as a title in the dialog box. Use this for a description of what the user is selecting.
- **Choice string** This is the string which identifies the choice.

**Description**

A menu block consists of:

1. a `MENU` instruction
2. two or more `CHOICE` instructions, each followed by the instruction or block of instructions to be executed if the user makes that choice.
3. an `END MENU` to terminate.

You can use either literal strings ("...") or `STRING` variables for the `menu description` and `choice identifier` strings.

**Example**

```plaintext
menu "Which Transformation Do You Want?"
choice  "Log"
  set series = log(series)
choice  "Square Root"
  set series = sqrt(series)
choice  "Percent Change"
  set(scratch) series = log(series/series{1})
choice  "None"
  ;
end menu
```
MESSAGEBOX — Simple Dialog Boxes

MESSAGEBOX displays a message in a dialog box, and waits for the user to respond to
the message. There are four types of dialog boxes from which to choose—they differ
only in the responses available to the user. For more complex dialogs, use the DBOX
or MENU and CHOICE instructions.

On most systems, an audible alert is generated when a MESSAGEBOX is displayed.

$messagebox( \text{options} ) \ "messagestring"

Parameters

"messagestring" This is the string that will be displayed in the dialog box. You
can supply this as a string enclosed in quotes (’ or “), or as a
STRING or LABEL type variable.

For a long message, RATS will automatically wrap lines to keep
the width reasonable. If you want to control breaks yourself,
insert the characters \ \ where you want a line break.

Options

style=[alert]/okcancel/yesno/yncancel
The message box displays the message supplied by messagestring, along with
one, two, or three “buttons.” For all choices except ALERT, you will want to use
the STATUS option to determine which button the user selected:

• STYLE=ALERT displays the message string and an “OK” button. The user
  must select OK to continue.
• STYLE=OKCANCEL displays “OK” and “Cancel” buttons. The user must
  select one of these to continue. The STATUS option returns 1 if the “OK”
  button is selected, and 0 if “Cancel” is selected.
• STYLE=YESNO displays “Yes” and “No” buttons. STATUS returns 1 for
  “Yes” and 0 for “No.” The DEFAULT option determines which is the de-
  fault.
• STYLE=YNCANCEL displays “Yes,” “No,” and “Cancel” buttons. STATUS
  returns 1 for “Yes,” 0 for “No,” and -1 for “Cancel.”

default=default choice for STYLE=YESNO [1=yes]
This controls which button will be the default choice (the button that will be
executed if the user just hits <Enter>). Use DEFAULT=0 if you want the “No”
button to be the default.

status=status code
status code is an INTEGER variable which will be set to the return values
described above.
Example

```r
messagebox(style=yesno,status=yn) "View the residuals?"
if yn==1
{
    graph(header="Regression Residuals") 1
    # %resids
}
```

This displays the following dialog:

![Dialog](image)

If the user clicks on the “Yes” button, RATS will execute the `GRAPH` instruction.

The code uses a `MESSAGEBOX` to ask the user if the current degrees of freedom value (stored in `DGF`) is correct. If the user responds “Yes”, the rest of the code is skipped.

If the user answers “No”, a `QUERY` instruction prompts the user to input a new value. The `VERIFY` option requires the user to supply a valid (greater than –1) value.

```r
if %BFlag == 1
{
    messagebox(sty=yesno,status=Stop) "Calculated "+%string(dgf)+ $ " degrees of freedom. Is this correct?"
    * If "No", use QUERY to have user input a value for DGF:
    if Stop == 0  {
        query(prompt="Input correct degrees of freedom", $ verify=(dgf>-1),error="Value must be larger than -1") dgf
    }
}
```
**MODIFY — Equation Maintenance**

**MODIFY** can be used (with the PRINT option) to display the structure and coefficients of an equation, or as the first of a set of three instructions which permit you to modify the structure of the equation. **MODIFY** only initiates the modification process; the actual work is done with **VREPLACE**, which replaces a variable with a filtered form of another variable, and **VADD**, which adds a new variable to the equation.

```
modify( option )   old equation   new equation
```

**Parameters**

- **old equation** The name or number of the equation you want to modify.
- **new equation** RATS stores the modified equation as `new equation`. By default, this is the same as `old equation`.

**Option**

- **print/[noprint]**
  Use `PRINT` to display the form of `old equation`.

**Example**

```
linreg(inst,define=supply) price
# constant quant sshift1  sshift2
modify supply
vreplace price by quant swap
frml(equation=supply) supplyeq
```

The **LINREG** instruction does an instrumental variables estimation with `PRICE` as the left-side variable, and saves the equation under the name `SUPPLY`. **MODIFY** and **VREPLACE** replace `PRICE` with `QUANT` as the left-side variable. The **FRML** instruction converts the equation to a formula (a **FRML** variable).

**Note**

When you use **MODIFY**, all subsequent **VREPLACE** and **VADD** instructions will apply to `old equation`, unless you do another **MODIFY** with a different equation.

**See Also . . .**

- **EQUATION** Sets up linear equations.
- **VREPLACE** Replace variables in an equation with other variables. It can also renormalize equations.
- **VADD** Adds additional variables to an equation.
MRESTRICT — Testing General Restrictions

MRESTRICT is used to test or impose restrictions on regressions. It is very similar to RESTRICT, except that you enter the restrictions using matrices rather than supplementary cards. This is often much more convenient, especially when there are many restrictions or when each restriction involves many coefficients. The matrix instructions EWISE and FMATRIX frequently prove useful in constructing the required matrices. See RESTRICT and Chapter 6 of the User’s Guide for more information.

```
mrestrict(options) restrictions Rmatrix rvector resids coeffs
```

Wizard

You can use the Regression Tests wizard on the Statistics menu to do restriction tests.

Parameters

See “Description” below for details on the Rmatrix and rvector parameters.

- **restrictions**: Number of restrictions imposed.
- **Rmatrix**: A RECTANGULAR with dimensions restrictions x number of coefficients. This matrix must be set before doing MRESTRICT.
- **rvector**: A VECTOR with dimension equal to restrictions. You can omit this if rvector is the zero vector. Use a * to skip rvector if you want to use either of the next two parameters.

Two additional parameters (resids and coeffs) used in older versions have been deprecated. Version 7 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.

Options

The options are the same as for RESTRICT, so we only include a brief description.

- **create/[nocreate]**
  With CREATE, MRESTRICT does a restricted regression (as opposed to just a test of the restrictions).

- **print/[noprint]** (with NOCREATE)
  With PRINT (and without CREATE), RATS prints the restricted coefficient vector in a table with just the label, lag and coefficient, but without standard errors and t-statistics. If you use NOPRINT explicitly, RATS suppresses all output.

- **title="string for output title"**
  You can use TITLE to include identifying information in the output.
MRestrict

[print]/noprint (with CREATE)
vcv/[novcv]
define=equation to define
frml=formula to define
unravel/[nounravel]
   When used with CREATE, these perform the same task they do for LINREG. See the LINREG section for details.

form=f/chisquared
   This affects only the test statistic, not the restricted regression. See Section 5.13 in the User’s Guide.

replace/[noreplace]
   REPLACE replaces the internal arrays %XX and %BETA by the restricted versions, but does not print output or compute residuals as the CREATE option does.

coeff=VECTOR for restricted coefficients
covmat=SYMMETRIC for restricted covariance matrix
   You can use these to save the restricted coefficient and/or covariance arrays. The REPLACE option is equivalent to using COEFF=%BETA and COVMAT=%XX.

Description
   Represent the linear restrictions as

   \[ R \beta = r \]

   where \( R \) is a \( Q \times K \) matrix and \( r \) a \( Q \)-vector, with \( Q \) as the number of restrictions and \( K \) the number of coefficients. You input \( R \) and \( r \) using the \( Rmatrix \) and \( rvector \) parameter fields.

Example
   This does the polynomial distributed lag from User’s Guide Example 5.6 using MRESTRICT. The restriction is that the 4th difference of the distributed lag coefficients is zero. It estimates the unconstrained distributed lag, then applies the 21 restrictions. The 1 and 2 on FMATRIX keep CONSTANT (regressor 1) out of the restriction by starting the difference operator at the 1,2 element of \( R \).

   linreg longrate 1951:1 2006:4
   # constant shortrate{0 to 24}
declare rect r(21,26)
fmatrix(diffs=4) r 1 2
mrestrict(create) 21 r

Variables Defined
   %BETA Vector of coefficients (VECTOR)
   %RESIDS Residuals series (SERIES)
MVSTATS — Moving Statistics and Fractiles

**MVSTATS** computes the mean, variance, maximum, minimum, median or general fractiles for a moving window of data. This is more efficient than putting a **DO** loop around a **STATISTICS** instruction. **MVSTATS** treats missing values more flexibly than **FILTER** or **SET**.

Note that the **MVFRACTILE** instruction from versions before 7.0 has now been combined into the **MVSTATS** instruction.

```
mvstats( options )   series  start  end  newseries  newstart
```

**Wizard**

You can use the *Moving Window Statistics* wizard on the *Data* menu to generate statistics and fractiles for moving windows of data.

**Parameters**

- **series**: Series to transform.
- **start**  **end**: Range of entries to transform. Unless you choose one of the **EXTEND** options, **start** and **end** need to allow for the span of data required by the window. For instance, a centered window of width 11 requires that **start** and **end** be at least 5 entries in from the beginning and end of **series**. If you have not set a **SMPL**, these default to the maximum usable range of **series**.
- **newseries**: Resulting series, used only when using the **FRACTILE** option to compute a single fractile. Use the **RESULTS** option when computing more than one fractile.
- **newstart**: The starting entry for **newseries**. By default, **newstart=start**.

**Options**

- **span=Width of the moving window [5]**
  The span is the number of entries considered at any one time. If you choose a **CENTERED** window, this should be odd. If it’s even, **MVSTATS** will make it odd by adding one.

- **centered/ [nocentered]**
  If you choose **CENTERED**, the results for an entry use a window centered at that entry. If not (the default), the window consists of the entry and preceding ones.
means=series of sample means [not used]
variances=series of sample variances [not used]
maximums=series of maximums [not used]
minimums=series of minimums [not used]
medians=series of medians [not used]

Use these options to save the moving means, variances, maximum, minimum, median and interquartile range into data series. You can use any combination of these. VARIANCES are calculated using a divisor of N–1.

iqranges=series of interquartile ranges (75%-25%) [not used]

Use these options to save the moving means, variances, maximum, minimum, median and interquartile range into data series. You can use any combination of these.

fractile=other fractile or /|list of other fractiles||
results=VECT[SERIES] of fractiles

Use FRACTILE to compute one more sets of moving fractiles (quantiles). If you just want a single fractile, use the newseries parameter to save the result. If you want to compute multiple fractiles, supply a list of the fractiles in the form FRACTILE=||f1,f2,...,fn|| and use the RESULTS option to save the results into an n–vector of series.

extend=zeros/repeat/rescale (no default)
By default, MVStats won’t compute an output value for observations where the window goes outside the input range. These options allow the output series to cover the same range as the input series. EXTEND=ZEROS extends the input series out-of-sample with zeros. EXTEND=REPEAT extends by repeating the first value (at the lower end) and the final value (at the upper end). EXTEND=RESCALE uses a window truncated to the actual data—for instance, an uncentered window at the left endpoint will use just that one data point.

Algorithm for Computing Fractiles
To compute fractile \( f \), the data in the window are sorted. (This is done by sequential insertions and deletions). If there are \( N \) points in the window, the quantity \( (N−1)f + 1 \) gives the position in the ranking for the desired fractile. If this isn’t an integer, a weighted sum of the values on either side is used. For instance, if \( (N−1)f + 1 \) is 3.75, the result is \(.25X_{(3)}+.75X_{(4)}\).

Missing Values/Global SMPL
Any missing values and entries skipped because of a global SMPL are dropped from the calculation of the statistics. Each will, however, have an output value, computed using the surrounding data.
Examples

\texttt{mvstats(variances=volatile,span=20) price}

sets the series \texttt{VOLATILE} at each period equal to the sample variance of the twenty entries ending at that period. \texttt{VOLATILE} is defined beginning at entry 20.

\texttt{mvstats(max=mhigh,min=mlow,span=13,center,extend=rescale) price}

sets the series \texttt{MHIGH} to the highest value achieved by \texttt{PRICE} in the thirteen periods centered on each entry. Values for entries near each end of the data are computed with increasingly asymmetric windows. For instance, the final entry will have the maximum and minimum of the last seven entries.

\texttt{mvstats(fractiles=||.1,.25,.75,.9||,results=pricefract) price}

computes four different fractiles of \texttt{PRICE}, storing the results in to a \texttt{VECTOR} of \texttt{SERIES} called \texttt{PRICEFRACT}.

\texttt{mvstats(iqrange=iqr,centered,span=21) x}

sets \texttt{IQR} to a moving interquartile range of series \texttt{x}, using current and ten entries on either side.

See Also . . .

\textbf{STATISTICS} \hspace{1cm} With the \texttt{FRACTILES} option, computes a fixed list of fractiles.
\textbf{EXTREMUM} \hspace{1cm} Finds the maximum and minimum values of a series.
\%\texttt{MINVALUE}(x) \hspace{1cm} Returns the minimum value of an array (or series).
\%\texttt{MAXVALUE}(x) \hspace{1cm} Returns the maximum value of an array (or series).
\%\texttt{FRACTILES}(x,f) \hspace{1cm} Computes a collection of fractiles for an array (or series).
NEXT — Loop Control

NEXT causes an immediate end to the current pass through a loop.

next (no parameters)

Description

As noted above, the NEXT instruction immediately ends the current pass through a loop. Execution continues with the start of the next pass through the loop:

- the index update for DO and DOFOR
- the condition check for WHILE and UNTIL
- the top of the loop for LOOP.

Example

dofor i = result1 result2 result3 result4
  statistics(noprint) i
  if %cdstat < 1.96
    next
    (remaining analysis executed only when the condition is false)
  end dofor

This performs a $t$-test on four series. If a series has mean insignificantly different from zero (at 5%), we skip the rest of the analysis for that series and go on to the next.

See Also . . .

* UG, Section 16.1 The RATS compiler
* BREAK Breaks control out of a loop.
* BRANCH Branches to a labeled instruction.
NLLS — Non-Linear Least Squares

NLLS estimates \( \beta \) in the non-linear least squares model

\[
y_i = f(X_i, \beta) + u_i
\]

Before you can use it, you must:

- Set the list of free parameters using NONLIN
- Create the explanatory formula \( f \) using FRML
- Set initial values for the parameters (usually with COMPUTE or INPUT)

NLLS can estimate models either by least squares or by instrumental variables.

NLSYSTEM estimates systems of equations, and MAXIMIZE can estimate models for which least squares isn't an appropriate estimation technique.

\[
nlls(options) \quad \text{depvar} \quad \text{start} \quad \text{end} \quad \text{residuals}
\]

Parameters

- **depvar** the dependent variable. You can use an asterisk (*) for this parameter if the model is \( f(X_i, \beta) = u_i \) so there is no “dependent variable.” If you use *, the regression output will exclude \( R^2 \) and similar summary statistics.
- **start** \( \text{end} \) the estimation range. If you have not set a SMPL, this defaults to the defined range of depvar.
- **residuals** (Optional) Series for the residuals.

General Options

- **frml=** formula name *(This option is required)*
  The name of the formula for the function \( f \).
- **[print]/noprint**
- **vcv/[novcv]**
- **smpl=** SMPL series or formula *(User’s Guide, Section 5.2)*
- **dfc=** Degrees of freedom correction *(Section 5.15)*
  These are the same as for other regressions.
- **title="title for output"** *[“Nonlinear Least Squares” or "Nonlinear Instrumental Variables"]*
  This option allows you to supply your own title to label the resulting output.
General Estimation Options

**parmset=** `PARAMSET` to use [default internal]

This option selects the parameter set to be estimated. See Section 7.6 of the `User’s Guide` for more on `PARAMSET`S. RATS maintains a single unnamed parameter set which is the one used for estimation if you don’t provide a named set.

**startup=** `FRML` evaluated at period “start”

Provides an expression which is computed only for the first entry of the estimation range, before the regular formulas are computed. The main purpose of this is to allow flexible initialization for recursive formulas.

**reject=** `FRML` indicating a “rejection” zone for the parameters

If the `FRML` evaluates to a non-zero (“true”) value, the function is immediately assigned the missing value.

**method=** `[gauss]`/`simplex`/`genetic`/`evaluate`

**pmethod=** `gauss`/`[simplex]`/`genetic`

**piters=** number of `PMETHOD` iterations to perform [none]

`METHOD` sets the estimation method to be used. The three methods are all described in Chapter 7 of the `User’s Guide`. The default is Gauss–Newton, which requires that the formula be twice continuously differentiable.

The `SIMPLEX` and `GENETIC` methods require only continuity, but can’t compute standard errors. They frequently are used to refine initial guesses before using Gauss–Newton. You can do that easily using the `PMETHOD` and `PITERS`, which select a preliminary estimation method. For example, to do 10 simplex iterations before switching to Gauss-Newton, you can use the options `PMETHOD=SIMPLEX`, `PITERS=10`, and `METHOD=GAUSS`.

With `METHOD=EVALUATE`, RATS simply evaluates the model given the initial parameter values, without trying to estimate new coefficient values.

**iterations=** iteration limit [100]

**subiterations=** subiteration limit [30]

**cvcrit=** convergence limit [.00001]

**trace/[notrace]**

`ITERATIONS` sets the maximum number of iterations, `SUBITERS` sets the maximum number of subiterations, `CVCRIT` the convergence criterion. `TRACE` prints the intermediate results. See Chapter 7 in the `User’s Guide` for more details.

**derives=** `VECTOR`[`SERIES`] for partial derivatives

This saves the series of partial derivatives of the residuals. The first series in the `VECTOR` will be the partials with respect to the first parameter displayed in the `NLLS` output, the second series will be the partials with respect to the second parameter, and so on.
Options for Non-Linear Least Squares

- robusterrors/[norobusterrors]
- lags=correlated lags [0]
- lwindow=neweywest/bartlett/damped/parzen/quadratic/[flat]/ panel/white
- damp=value of $\gamma$ for lwindow=damped [0.0]
- lwform=VECTOR with the window form [not used]
- cluster=SERIES with category values for clustered calculation

When used without the INSTRUMENTS option, these permit calculation of a consistent covariance matrix allowing for heteroscedasticity (with ROBUSTERS-RORS) or serial correlation (with LAGS), or clustering based upon some other set of categories. See Sections 5.3, 5.4, and 5.5 of the User’s Guide and the description of the instruction MCOV for more information. Note especially the possible computational problems that may arise when using LAGS.

- spread=Residual variance series/formula
- sv=Residual variance formula

Use one of these for weighted least squares. The difference is that SPREAD is used when the variances don’t depend upon the parameters (so simple weighted least squares can be used) and SV is used when they do (requiring maximum likelihood to account for log variance terms). The residual variances are assumed to be proportional to the given series.

- jacobian=FRML for the Jacobian

This option allows you to specify a FRML for the determinant of the Jacobian, for doing full-information maximum likelihood.

Options for Instrumental Variables (GMM)

- instruments/[noinstruments]

Use the INSTRUMENTS option to do two-stage least squares or GMM. You must set your instruments list first using the instruction INSTRUMENTS.

- wmatrix=weighting matrix $[(Z'Z)^{-1}]$
- optimalweights/[nooptimalweights]
- update=[none]/once/continuous

This controls the updating of the weighting matrix. You can directly input a weight matrix with the WMATRIX option. If you don’t, the weight matrix is initialized as the $(Z'Z)^{-1}$ matrix which gives non-linear two-stage least squares. If you use OPTIMALWEIGHTS or UPDATE=CONTINUOUS (they’re synonyms), the weight matrices are recomputed after every iteration using the choices you have made from the next group of options.
By default, `NLLS` estimates using the Gauss–Newton algorithm with numerical partial derivatives. See *User’s Guide* Sections 7.2 and 7.7 for more on this. The `SIMPLEX` and `GENETIC` methods don’t require derivatives.

**Hypothesis tests**

You can test hypotheses on the coefficients with `TEST`, `RESTRICT` and `MRESTRICT`. RATS numbers the coefficients based on their positions in the list set up by `NONLIN`. You may not use `EXCLUDE` after `NLLS` and can use `SUMMARIZE` only to expand an expression.

The hypothesis tests are based upon a quadratic approximation to the sum of squares surface at the final estimates. This is a “Wald” test.
Missing Values

RATS drops any entries which it can’t compute because of missing values. If you have a recursive formula (that is, the value for entry \( T \) relies upon some quantity computed at \( T - 1 \)), then you have to be careful about the choice of \( \text{start} \) and \( \text{end} \). You can’t let RATS determine the default range because it will simply start at entry one (which generally isn’t computable), and will never find, by itself, the correct place to begin.

Examples

nonlin lgamma delta nu rho
frml ces = lgamma-nu/rho* \\
        log( delta * k**(1-rho) + (1.-delta) * l**(1-rho) )
compute lgamma=1.0, delta=0.4, nu=0.8, rho=0.6
nlls(frml=ces,trace) q

Examples (continued)

nonlin a b g
linreg realcons
# constant realdpi
compute a=%beta(1),b=%beta(2),g=1
frml cfrml realcons = a+b*realdpi**g
instruments constant realcons{1} realdpi{1 2}
nlls(frml=cfrml,inst) realcons 1950:3 *

Variables Defined

NLLS defines the following standard estimation variables (see LINREG)

\%RESIDS \hspace{2cm} \text{(SERIES)}
\%NREG, \%NOBS, \%NDF \hspace{2cm} \text{(INTEGER)}
\%RSS, \%SEESQ, \%RSQUARED, \%RBARSQ, \%TRSQ, \%TRSQUARED, \%LOGL,
\%DURBIN, \%RHO, \%MEAN, \%VARIANCE, \%JSTAT, \%JSIGNIF \hspace{2cm} \text{(REAL)}
\%BETA, \%TSTATS, \%STDERRS \hspace{2cm} \text{(VECTOR)}
\%XX, \%WMATRIX \hspace{2cm} \text{(SYMMETRIC)}

and the following other variables:

\%UZWZU \hspace{2cm} \text{criterion function for instrumental variables (REAL)}
\%FUNCVAL \hspace{2cm} \text{Final value of} \ G(\beta) \hspace{2cm} \text{(REAL)}
\%ITERS \hspace{2cm} \text{Iterations taken}
\%CONVERGED \hspace{2cm} \text{= 1 or 0. Set to 1 if the process converged, 0 if not.}
\%CVCRIT \hspace{2cm} \text{final convergence criterion. This will be equal to zero if the subiterations limit was reached on the last iteration (REAL)}
\%LAGRANGE \hspace{2cm} \text{VECTOR of Lagrange multipliers if estimating with constraints.}
\%JSTAT \hspace{2cm} \text{Test statistic for overidentification for instrumental variables.}
\%JSIGNIF \hspace{2cm} \text{Significance of} \ %JSTAT \hspace{2cm} \text{(REAL)}
NLPAR — Controlling Non-linear Estimation

**NLPAR** allows you to change parameter settings which govern the inner workings of the iterative processes. Changes made using it will apply to any subsequent non-linear estimations (until you change them again with another **NLPAR** or quit RATS).

```
nlpar( options )  (no parameters)
```

### Description

The instructions **BOXJENK** and **ITERATE** (ARMA estimation), **NLLS** and **NLSYSTEM** (Non-Linear Least Squares and systems estimation), **DDV** and **LDV** (discrete and limited dependent variables), **ESMOOTH** (exponential smoothing), **MAXIMIZE, FIND**, **DLM** (dynamic linear models) and **CVMODEL** all involve nonlinear optimization.

Because of this, estimation is an iterative process, and certain models may prove difficult to fit. Most of the time, you can get your model estimated by altering the method used, the initial guess values, and the number of iterations. The options to control these are provided on each estimation instruction.

However, some models can be difficult or impossible to estimate without further adjustment to the estimation process. The options on **NLPAR** provide control over these settings.

In general, there are three main methods of optimization used by RATS: the hill-climbing methods (Section 7.2 of the *User’s Guide*), and the simplex and genetic methods (both in Section 7.3). Of these, the one most likely to benefit from “tuning” is the genetic algorithm. This has a set of four control options. Different settings of these can greatly alter the speed of convergence. This is because the genetic algorithm is the only one designed to handle multiple peaks in a function being maximized. The other methods attempt to climb the “hill” on which you start. If you indeed have a function with multiple peaks, the process of “mutating” vectors can result in a lot of wasted efforts, as combining parameter vectors will tend to give parameter vectors in the regions between the peaks.

### Options

The following is a general description of the **NLPAR** options. Please see Chapter 7 in the *User’s Guide* for technical details on the parameters controlled via **NLPAR**, as well as general information on the algorithms and methods used.

**criterion=[coeffs]/value**

This chooses whether RATS determines convergence by looking at the change from one iteration to the next in the coefficients, or the change in the value of the function being optimized. Convergence of coefficients is the default. Convergence on value should be chosen only if the coefficients themselves are unimportant.
**exactlinesearch/[noexactlinesearch]**

In the “climbing” methods (Section 7.2 of the *User’s Guide*), where a direction is chosen and a search made along that direction, this option controls whether the estimation routine attempts to find the “exact” optimum in that direction, or only looks for a new set of points which meet certain criteria. This choice rarely affects whether you will get convergence—it merely alters how quickly you get there. In general, EXACTLINESEARCH is slower for smaller models, as the extra function evaluations provide little improvement for the effort. It can speed up models with many free parameters, where the “cost” of each new iteration, particularly the need to compute the gradient, is higher.

**subiterations=** subiteration limit [30]

This limits the number of new coefficient vectors examined once a direction has been chosen. The default is 30. Most estimation instructions have their own SUBITERS option. Unless you use the EXACTLINESEARCH, you will rarely see more than three subiterations except on the first few iterations.

**cvcrit=** convergence limit [.00001]

CVCRIT sets the default convergence criterion.

**alpha=** value for the $\alpha$ control parameter [0.0]

**delta=** value for the $\delta$ control parameter [0.1]

These parameters control details of the hill-climbing methods (BHHH and BFGS). There are few situations in which changing these is likely to help.

**mutate=[simple]/shrink/best/random**

**crossover=** probability of taking mutated parameter [.5]

**scalefactor=** scale factor for mutations [.7]

**populate=** scale for population size [6]

These parameters control details of the GENETIC estimation method (*User’s Guide*, Section 7.3)

**marquardt/[nomarquardt]**

For Gauss-Newton estimation, you can use MARQUARDT to select the Levenburg–Marquardt subiteration algorithm (Marquardt, D. 1963. “An Algorithm for Least-Squares Estimation of Nonlinear Parameters.” *Journal of the Society for Industrial and Applied Mathematics*, Vol. 11, pp.431-441). We generally don’t recommend this, as we have generally obtained better results using the default method.

**jiggle=** iterations between perturbations of simplex [30]

When using SIMPLEX, RATS normally perturbs the simplex vertices every 30 iterations. If the TRACE output suggests that the simplex estimation is not moving much, you may find some benefit to using a smaller value for JIGGLE. See Section 7.3 for more information.
NLSYSTEM — Non-linear Systems Estimation

NLSYSTEM estimates a system of non-linear equations using multivariate non-linear least squares or GMM.

```
nlsystem(options)   start   end   list of FRMLS
```

**Parameters**

- **start end**
  Estimation range. If you have not set a SMPL, this defaults to the common defined range of all the dependent variables of the equations.

- **list of FRMLS**
  The list of the formulas you want to estimate. A formula can define the residuals $u_t$ either as $y_t = f(X_t, \beta) + u_t$ or as $f(X_t, \beta) = u_t$. NLSYSTEM assumes the first if you define the formula with a dependent variable and the second if you don’t. Please note that unlike NLLS, you can only indicate the dependent variable of a formula when you define the formula—there is no way to do it on the NLSYSTEM instruction.

Alternatively, you can use the FRMLVECT option to supply the list of formulas in the form of a VECTOR of FRMLs.

**Description**

NLSYSTEM uses one of two techniques:

- Multivariate non-linear least squares (non-linear SURE) when you estimate without instruments.
- Generalized Method of Moments for instrumental variables estimation. Non-linear three-stage least squares is a special case of this.

It does not do Maximum Likelihood estimation (except when these other techniques are equivalent to ML).

Before you can use NLSYSTEM, you must:

- Set the list of free parameters using NONLIN
- Define the FRMLS
- Set initial values for the parameters (usually with COMPUTE or INPUT).

If you have a linear system, the instruction SUR may be a better alternative. It is much faster at any type of model which both instructions can estimate.
Technical Information

(1) \( u_t = (u_{t1}, ..., u_{tn})' \) is the vector of residuals at time \( t \) (\( u \) depends upon \( \beta \)), and

(2) \( \Sigma = E u_t u_t' \)

Multivariate non-linear least squares solves

(3) \( \min_{\beta} \sum_t u_t' \Sigma^{-1} u_t \)

For instrumental variables, further define

(4) \( Z_t = (z_{t1}, ..., z_{tr})' \) is the vector of instruments at \( t \), and

(5) \( G(\beta) = \sum_t u_t \otimes Z_t \)

then Generalized Method of Moments solves

(6) \( \min_{\beta} G(\beta)' [SW] G(\beta) \)

where \( SW \) is the weighting matrix for the orthogonality conditions. Some of the options of \texttt{NLSYSTEM} let you set the form for the matrix \( SW \) in formula (6). By default, it is just \( \Sigma^{-1} \otimes Z'Z^{-1} \), where \( Z \) is the \( T \times r \) matrix of instruments for the entire sample. With this, \texttt{NLSYSTEM} does non-linear three stage least squares.

General Options

\texttt{model=model to be estimated}

\texttt{frmlvect=FRML[VECTOR] to be estimated}

As an alternative to listing the formulas to be estimated as parameters, you can use the \texttt{MODEL} option to estimate an existing \texttt{MODEL}, or use \texttt{FRMLVECT} to provide a single \texttt{FRML} which, at each entry, provides the residuals in a \texttt{VECTOR} form. If you use \texttt{MODEL}, the \texttt{MODEL} must consist only of \texttt{FRML}'s.

\texttt{[print]/noprint}

\texttt{vcv/[novcv]}

\texttt{sigma/[nosigma]}

These control the printing of the regression output, the covariance matrix of the complete coefficient vector, and the final estimate of the residual covariance matrix.

\texttt{smpl=SMPL series or formula (User's Guide, Section 5.2)}

You can supply a series or a formula that can be evaluated across entry numbers. Entries for which the series or formula is zero or “false” will be skipped, while entries that are non-zero or “true” will be included in the operation.
residuals=VECTOR[SERIES] for residuals
This is the most convenient way to get the residuals. RESIDUALS=RESVAR will create series RESVAR(1), ..., RESVAR(n) which will have the residuals from the n equations in the system.

cv=Input Σ matrix (SYMMETRIC)
cvout=Output Σ matrix (SYMMETRIC)
CV allows you to feed in an initial covariance matrix, and CVOUT allows you to store the final estimate of the covariance matrix. For CVOUT, you don’t need to DECLARE or DIMENSION the array. When you use CV, the standard errors and covariance matrix of coefficients will be correct only if the CV matrix incorporates the residual variances or if you use the option ROBUSTERRORS.

These options were called ISIGMA and OUTSIGMA, respectively, in earlier versions.

title="title for output" [depends upon options]
This option allows you to supply your own title to label the resulting output.

derives=RECTANGULAR[SERIES] for partial derivatives
This stores the series of partial derivatives. The columns in the matrix correspond to the different sets of residuals. Thus the series in row one, column one of the array will be the partials with respect to the first parameter displayed in the output for the first residual. The series in row two, column one will the partials with respect to the second parameter for the first residual, and so on.

dfc=Degrees of freedom correction (User’s Guide, Section 5.15)
These are the same as for other regressions.

General Estimation Options

iterations=iteration limit [100]
subiterations=subiteration limit [30]
cvcrit=convergence limit [.00001]
trace/[notrace]
ITERATIONS sets the maximum number of iterations, SUBITERS sets the maximum number of subiterations, CVCRIT the convergence criterion. TRACE prints the intermediate results. See Chapter 7 in the User’s Guide for more details. Note that the “function value” information when you use TRACE is not particularly useful. The objective function actually changes from one iteration to the next as the “nuisance parameters” (the Σ or weight matrix) are re-estimated.

parmset=PARMSET to use [default internal]
This option selects the parameter set to be estimated. See Section 7.6 of the User’s Guide for more on PARMSETs. RATS maintains a single unnamed parameter set which is the one used for estimation if you don’t provide a named set.
\texttt{startup} = \textit{FRML evaluated at period "start"}

provides an expression which is computed only for the first entry of the estimation range, before the regular formulas are computed. The main value of this is to allow flexible initialization for recursive formulas such as Markov switching models (see Section 12.3 of the \textit{User's Guide}).

\texttt{jacobian} = \textit{FRML for the Jacobian}

This option allows you to specify a \textit{FRML} for the determinant of the Jacobian, for doing full-information maximum likelihood.

\texttt{reject} = \textit{FRML indicating a "rejection" zone for the parameters}

If the \textit{FRML} evaluates to a non-zero ("true") value, the function is immediately assigned the missing value.

\textbf{Options for Multivariate Least Squares}

\texttt{robusterrors/ [norobusterrors]}

\texttt{lags=correlated lags [0]}

\texttt{lwindow=neweywest/bartlett/damped/parzen/quadratic/[flat]/panel/white}

\texttt{damp=value of $\gamma$ for lwindow=damped [0.0]}

\texttt{lwform=VECTOR with the window form [not used]}

\texttt{cluster=identifying SERIES for clustered std. errors [not used]}

When you use these \textit{without} the \texttt{INSTRUMENTS} option, they allow you to calculate a consistent covariance matrix allowing for heteroscedasticity (with \texttt{ROBUSTERRORS}), serial correlation (with \texttt{ROBUSTERRORS} and \texttt{LAGS}), or clustered standard errors (with \texttt{ROBUSTERRORS} and \texttt{CLUSTER}). See Sections 5.3, 5.4, 5.5, and 7.5 of the \textit{User's Guide} and the description of the instruction \texttt{MCOV} for more information.

Note that none of these options affect the parameter estimates. Just as with \texttt{LINREG} and \texttt{NLLS}, these options come into play when the covariance matrix of the estimates is computed. These behave differently when you are using \texttt{NLSYSTEM} with the \texttt{INSTRUMENTS} option, as described below.
Options for Instrumental Variables

Given a set of instruments, there is a different Generalized Method of Moments estimator for each choice of the weighting matrix $SW$ (except if the model is just identified—then all choices of $SW$ will give the same estimates). NLSYSTEM treats as a special case those situations where the $SW$ matrix can be written in the form $\Sigma^{-1} \otimes W$. Note that $SW$ is an $nr \times nr$ matrix where $n$ is the number of equations and $r$ the number of instruments.

**instruments/[noinstruments]**

Use the INST option if you want to do instrumental variables estimation. Be sure to construct the instruments list in advance using the INSTRUMENTS instruction.

**zumean=VECTOR of means of moment conditions [all zero]**

This allows you to supply a VECTOR (with dimensions equal to the number of instruments times the number of equations) with a set of known (non-zero) means for $E(u \otimes Z)$. By default, these are zero.

**zudep/[nozudep]**

**wmatrix=SYMMETRIC weighting matrix for instruments [(Z'Z)^{-1}]**

**sw=SYMMETRIC grand weighting matrix [not used]**

**swout=estimated SYMMETRIC grand weighting matrix [not used]**

NOZUDEP (the default) is the special case for the $SW$ matrix. We call this NO-ZUDEP because the most important case is where $u$ is (serially uncorrelated and) independent of the instruments $Z$. More generally, this is Case (i) in Hansen (1982, page 1043).

With NOZUDEP, you can use the WMATRIX option to set the $W$ part of the $\Sigma^{-1} \otimes W$ and the CV option to set $\Sigma$. Otherwise, NLSYSTEM estimates a new $\Sigma$ after each iteration. Note that, if you use the LAGS option, NLSYSTEM will automatically switch to the ZUDEP method of handling the weight matrix.

If ZUDEP, you can use the SW option to set the full $SW$ array. This is an $nr \times nr$ SYMMETRIC array. Otherwise, NLSYSTEM determines a new $SW$ matrix after each iteration by taking the inverse of

$$(7) \quad \frac{1}{T} \sum (u_i \otimes Z_i)(u_i \otimes Z_i)'$$

(or the generalization of this if you use the LAGS option). The SWOUT option allows you to save the estimated $SW$ matrix into the specified array.

**center/[nocenter]**

CENTER adjusts the weight matrix calculation to subtract off the (sample) means of $u \otimes Z$, which may be non-zero for an overidentified model. See the description of MCOV for more information.
**update=none/once/continuous** [default depends on other options]

This controls the updating of the weighting matrix. The default is normally `UPDATE=CONTINUOUS`, which recalculates the weight matrix at each iteration, except in the following cases:

- If you use the `SW` option, the default is `UPDATE=NONE`.
- If you use the `CV` option with `NOZUDEP`, the default is `UPDATE=ONECE`.

This option replaces the `SECONDSTEP` option used in earlier versions.

**mask=RECTANGULAR masking array for instruments** [not used]

Normally, RATS uses the same set of instruments (those listed on the `INSTRUMENTS` instruction) for all equations in the `NLSYSTEM`. `MASK` allows you to use different sets of instruments for some or all of the equations in your system.

You supply a `RECTANGULAR` array with dimensions “number of instruments” by “number of formulas,” so that each column corresponds to a formula (in the order listed on the `NLSYSTEM` instruction), and each row corresponds to one of the instruments (in the order listed on the `INSTRUMENTS` instruction).

To apply a particular instrument to a given formula, set the corresponding element of the `MASK` array to a non-zero value, such as 1. To exclude an instrument from a formula, set the appropriate entry to 0.

**lags=correlated lags [0]**

**lwindow=neweywest/bartlett/damped/parzen/quadratic/[flat]/panel/white**

**damp=value of γ for lwindow=damped [0.0]**

**lwform=VECTOR with the window form** [not used]

**cluster=SERIES for clustered serial correlation** [not used]

Use `LAGS` or `LWINDOW=PANEL` or the `CLUSTER` option to handle serial correlation in the $u \otimes Z$ process. See Section 5.3 of the `User’s Guide` for more information.

Note that, unlike `NLSYSTEM` without `INSTRUMENTS`, these options do affect the estimated coefficients by changing the weight matrix.

**robusterrors/[norobusterrors]**

If you use `ROBUSTERRORS` combined with an input `CV` or `SW` matrix, `NLSYSTEM` will compute the coefficients using the “suboptimal” weighting matrix and then correct the covariance matrix of the coefficients based upon the choices for the `LAGS`, `LWINDOW` and other options immediately above.

**jrobust=statistic/[distribution]**

You can use this option to adjust the $J$-statistic specification test when the weighting matrix used is not the optimal one. See Section 7.8 in the `User’s Guide` for more information.
Variables Defined

The matrices %XX, %BETA, %STDERRS and %TSTATS are all defined for the full set of coefficients and %NREG is the count of the number of regressors in the full system. %NOBS is the number of observations. Other variables defined are:

- %LOGDET: log determinant of the estimate of \( \Sigma \)
- %SIGMA: final covariance matrix of the residuals
- %UZWZU: criterion function for instrumental variables (REAL)
- %LAGRANGE: VECTOR of Lagrange multipliers if estimating with constraints.
- %JSTAT: Test statistic for overidentification for instrumental variables.

If you don’t use ROBUSTERRORS, NLSystem will assume the weight matrix is “optimal” and use the value of (6) as the \( J \)-statistic. With ROBUSTERRORS, the formula in Lemma 4.1 from Hansen (1982) is used.

- %JSIGNIF: Significance of %JSTAT (REAL)

Missing Values

RATS drops any entries which it cannot compute because of missing values.

Output

NLSystem prints a summary of information on the fit for each equation. The parameter estimates are listed in a single table.

Examples

This estimates by GMM a model for the behavior of interest rates which based upon moment conditions for the mean and variance of the residuals.

```plaintext
nonlin alpha beta gamma sigmasq
frml eps = y1{-1}-(1+beta)*y1-alpha
frml variance = eps(t)**2-sigmasq*y1**(2*gamma)
```

Just identified model

```plaintext
instruments constant y1
nlsystem(instruments) / eps variance
```

Overidentified model. With nozudep

```plaintext
instruments constant y1{0 1}
nlsystem(instruments) / eps variance
```
This estimates a system of three equations (for investment, consumption and an interest rate). The first is a simple instrumental variables estimator which will give the same answers as an equivalent \textit{SUR} iterated to convergence.

The second \texttt{NLSYSTEM} uses an alternative investment equation which includes first order autocorre-lated errors. This is done by adding a lagged dependent variable and a lag of the base investment formula to create the formula \texttt{INVESTAR}.

The final estimate returns to the original three equations, but estimates allowing for a two lag serial correlation in $u \otimes Z$. Note that the \texttt{ITERS} option has been greatly increased for this last estimator. Estimating with \texttt{ZDEP} (which is implicit when you use \texttt{LAGS}) tends to cause very slow convergence, as the weights change from iteration to iteration. There are quite a few moment conditions (16 instruments x 3 equations) for the amount of data (144 observations), which creates an even greater problem getting the weights to converge.

\begin{verbatim}
nonlin(parmset=structural) c0 c1 c2 i0 i1 i2 i3 r0 r1 r2 r3 r4
nonlin(parmset=ar) rho

instruments constant cons{1 2} ydiff{1 2} gnp{1} govt{0 1} $
   mdiff{0 1} rate{0 to 5}$

frml investnl invest = i0+i1*ydiff{1}+i2*gnp+i3*rate{4}
frml investar invest = rho*invest{1}+investnl{0}-rho*investnl{1}
frml consnl cons  = c0+c1*gnp+c2*cons{1}
frml ratenl rate  = r0+r1*gnp+r2*ydiff+r3*mdiff+r4*rsum{1}
compute c0=c1=c2=0.0
compute i1=i1=i2=i3=0.0
compute r0=r1=r2=r3=r4=0.0
compute rho=0.0

nlsystem(inst,parmset=structural) 1950:1 1985:4 $
   investnl consnl ratenl

nlsystem(inst,parmset=structural+ar) 1950:1 1985:4 $
   investar consnl ratenl

nlsystem(inst,parmset=structural,lags=2,damp=1.0,$
   iters=400) 1950:1 1985:4 investnl consnl ratenl
\end{verbatim}
NNLEARN — Neural Net Training

**NNLEARN** uses backpropagation techniques to train a new or existing neural network to model the relationship between a set of input series and a set of output series. You can use **NNTEST** to generate output (the fitted values) from a neural network trained using **NNLEARN**. See *User’s Guide* Section 12.6 for more on neural networks.

```
nnlearn( options )  start   end
# list of input series in regression format
# list of output series
```

**Parameters**

`start   end` Range of entries of the output series to use.

**Supplementary cards**

The first supplementary card supplies the list of input series, while the second card supplies the list of output series. The input series (first supplementary card) are analogous to explanatory or independent variables in a regression, while the output series (second card) are analogous to dependent variable(s).

The first card supports regression format, which means that you can include lags or leads on the input list. The output list, however, must consist only of one or more series names (no lags or leads).

**Options**

`save=memory vector` *(required)*

`restart/ [norestart]`

The **SAVE** option saves the estimated weights of neural network model, as well as general information about the model (number of inputs, number of outputs, etc.) in a VECTOR of REALS. The **memory vector** can be used in subsequent **NNLEARN** commands for additional training as described below, or with the **NNTEST** instruction to generate fitted values.

If you re-use an existing **memory vector**, **NNLEARN** will, by default, use the values in the vector as the starting point for further training. This allows you to do further training of an existing network using additional **NNLEARN** commands. Use the **RESTART** option if you want to re-use the same vector name, but want **NNLEARN** to start the estimation from a new set of randomly generated initial values. In either case, after the estimation is completed, the new weights and information are saved into **memory vector**, replacing the earlier values.

`hidden=number of hidden nodes [number of input series]`

This sets the number of hidden nodes in the neural network.
**direct/[nodirect]**

If **DIRECT**, the model will include direct links between the input nodes and the output nodes. If **NODIRECT**, the only connection will be through hidden nodes.

**ymin=** *minimum scale value for outputs*  [ minimum output value ]

**ymax=** *maximum scale value of outputs*  [ maximum output value ]

These options set the upper and lower bounds on the values of the outputs of the neural network. They control how the internal values of the network (which range from 0 to 1 or –1 to 1 depending on the squashing function used) are mapped to the actual output values. By default, these are set to the maximum and minimum values of the original training sample.

**pad=** *fraction to pad*  [ 0 ]

The values of the network outputs run from 0 to 1 or –1 to 1 (depending on the **SQUASH** choice). By default, the outputs are scaled so that this range maps to the smallest and largest values in the training sample output series. If the model is ever used with samples should produce larger or smaller output values than were present in the training sample, the outputs produced by **NNTEST** will be artificially truncated. You can avoid this by using the **PAD** option to provide a value between 0 and 1 which indicates the fraction of “padding” to include when rescaling the output variables.

If, for instance, you choose **PAD= .2**, the smallest output value in the training sample will be mapped to .1 while the largest will be mapped to .9. If the original range of the data were from 7.2 to 8, this would allow the network to produce forecasts up to 8.1 and down to 7.1. See Section 12.6 of the *User’s Guide*.

**mode=[epoch]/example**

This controls how often new weights are computed. With **EPOCH**, RATS does a forward and backward pass through the network for all observations in the sample range before recomputing the weights. With **EXAMPLE**, weights are recomputed after (a forward and backward pass through) each observation in the sample.

**squash=[logistic]/ht1/ht2**

Selects the sigmoidal filter to be used for “squashing” the node outputs:

- **LOGISTIC** \( \frac{1}{1 + e^{-u}} \) (logistic function)
- **HT1** \( \tanh(u) \)
- **HT2** \( \tanh(u/2) \)

where \( u \) is the basic output of a node—a linear function of the input values and the current weights. The actual output of each node is a sigmoidal function of \( u \). The **SQUASH** option allows you to choose which of these three functions is used to generate the output. These serve to scale the outputs so that they fall between 0 and 1 (**LOGISTIC**) or between –1 and 1 (**HT1** and **HT2**).
\texttt{smpl=} \textbf{SMPL series or formula (User’s Guide, Section 5.2)}
This series or formula should return 0 for entries you want to omit from the training, and non-zero values for the other entries.

\texttt{iters=} \textbf{maximum number of iterations [no default limit]}
Sets a limit on the maximum number of iterations that will be performed. Each “iteration” involves less computation than, say, a \texttt{MAXIMIZE} instruction, but also has less of a chance to produce an improvement. You might need an iteration limit in the thousands.

\texttt{cvcrit=} \textbf{convergence criterion [.00001]}
\texttt{rsquared=} \textbf{minimum R-squared level}
These mutually exclusive options provide two ways of specifying convergence criteria for the learning process. Both can produce equivalent fits—they simply offer two ways of thinking about the criteria.

If you use the \texttt{CVCRIT} option, RATS will train the model until the mean square error (the mean of the squared error between the output series and the current output values of the network) is less than the \texttt{CVCRIT} value.

If you use the \texttt{RSQUARED} option, RATS will train the model until the mean square error is less than $\left(1 - R^2\right)\sigma^2$, where $R^2$ is the value specified in the \texttt{RSQUARED} option, and $\sigma^2$ is the smallest of the output series variances.

The default setting is \texttt{CVCRIT=.00001}. If you specify both options, RATS will use the \texttt{CVCRIT} setting.

\texttt{trace/[notrace]}
If you turn on the \texttt{TRACE} option, RATS will periodically display the number of iterations (epochs) evaluated and the current value of the convergence criterion. We recommend that you always use \texttt{TRACE}, particularly when developing new models.

\texttt{theta=} \textbf{theta parameter [0.7]} (must be in the range $0 \leq \theta < 1$)
\texttt{kappa=} \textbf{kappa parameter [0.1]} (must be in the range $0 \leq \kappa < 1$)
\texttt{phi=} \textbf{phi parameter [0.5]} (must be in the range $0 < \phi < 1$)
\texttt{mu=} \textbf{momentum parameter [0.0]} (must be in the range $0 \leq \mu < 1$)
These allow you to control various parameters of the adaptive learning rate algorithm. The \texttt{THETA} value affects how derivatives are averaged over recent derivatives. \texttt{KAPPA} affects how a weight is increased if the current derivative has the same direction as recent derivatives, while \texttt{PHI} affects how the weights change if the derivative changes direction. A non-zero value for the \texttt{MU} option adds “momentum” to the adaptive learning process, which helps prevent temporary changes in direction from adversely affecting the learning process (that is, it limits wild fluctuations in different directions).
Description

The **NNLEARN** instruction fits a neural net model based on the relationship between a set of input series and a set of output series. If the input series are $X_1, X_2, ..., X_n$ and the output series are $Y_1, Y_2, ..., Y_m$, this fits:

$$Y_i \sim F(X_j), \text{ for } i=1,2,...,m; \ j=1,2,...,n$$

where $F$ is the relationship to be modelled.

Examples

As a simple demonstration, we’ll fit a neural network model for the XOR (exclusive OR) function. The XOR function takes two binary values (1 or 0, true or false) as input, and returns a true value when either (but not both) of the inputs are true, and returns a false value otherwise.

```plaintext
all 4
data(unit=input,org=obs) / x1 x2 xor_actual
0 0 0
0 1 1
1 0 1
1 1 0
nnlearn(save=mem,rsquared=.9999,hidden=2)
   # x1 x2
   # xor_actual
nntest / mem
   # x1 x2
   # xor_output
print / x1 x2 xor_actual xor_output
```
Key to Memory Vectors

The `SAVE` option stores information about the structure of the neural network, as well as the estimated network weights in a VECTOR of REALS. If \( I \) is the total number of input nodes, \( O \) is the total number of output nodes, and \( H \) is the number of hidden nodes, this vector will have the following number of elements:

\[
10 + (I + 1) \times H + (H + 1) \times O
\]

(with \textsc{nodi}rect option), or

\[
10 + (I + 1) \times H + (H + 1) \times O + (I \times O)
\]

(with \textsc{di}rect option)

The first 10 elements of the vector contain basic information about the network (number of input nodes, number of hidden nodes, and so on). The remaining elements contain the computed weights of the neural network, as described below:

\[
\text{memory}(11,...,P) = \alpha_{ij}, \text{ for } i=1,...,H; j=0,...,I \quad \text{(i.e. } \alpha_{10}, \alpha_{11}, \alpha_{12},...,\alpha_{20}, \alpha_{21}, \alpha_{22}, \text{ etc.)}
\]

For \( j=0 \), \( \alpha_{ij} \) is the bias weight on hidden node \( i \), otherwise \( \alpha_{ij} \) is the weight on hidden node \( i \) from input node \( j \).

\[
\text{memory}(P+1,...,Q) = \beta_{ij}, \text{ for } i=1,...,H; j=0,...,O
\]

For \( j=0 \), \( \beta_{ij} \) is the bias weight on output node \( i \), otherwise, \( \beta_{ij} \) is the weight on output node \( i \) from hidden node \( j \).

\[
\text{memory}(Q+1,...,R) = \delta_{ij}, \text{ for } i=1,...,O; j=1,...,I \quad \text{(only when using the \textsc{di}rect option)}
\]

If you use \textsc{di}rect, the memory vector will also contain the weights on the direct connections—\( \delta_{ij} \) is the weight on the connection from input node \( j \) on output node \( i \).
NNTEST — Neural Net Testing and Predictions

**NNTEST** generates output from a neural network model. It can be used either to validate a model with in-sample data, or to forecast out-of-sample. See **NNLEARN** and Section 12.6 of the *User’s Guide* for more information.

```
nntest(options) start end memoryvector

# list of input series in regression format
# list of output (or validation) series
```

**Parameters**

- **start**  **end**  The range of entries for which you want to generate output.
- **memoryvector**  (Required) A memory vector containing the neural net model weights (set by the **SAVE** option on **NNLEARN**).

**Options**

- **smpl=SMPL series**  *(User’s Guide, Section 5.2)*
  
  If you use the **SMPL** option, **NNTEST** will only compute output for entries corresponding to non-zero values of the **SMPL** series. No output will be calculated for entries where the **SMPL** series is zero or **NA**.

- **validate/[novalidate]**
  
  If you use the **VALIDATE** option, **NNTEST** compares the output from the network with the actual data in the **output series**. The mean square error is computed and saved in **%FUNCVAL**. You can use this for automated validation of a part of the sample. If you use this, the values of **output series** won’t be affected.

**Description**

Using the neural net model specified by the **memory vector** parameter, **NNTEST** takes the supplied input series and computes the output. If you use **VALIDATE**, it will compare these with the data in the series listed on the supplementary card. If you don’t (by default), it will store the results in the series listed on the supplementary card.

Note that the number of input and output series must match those used on the **NNLEARN** to estimate the model.

**Example**

```
nntest / nnmodel
# x1 x2 x3
# ypreds
```
NONLIN — Setting Free Parameters for Non-Linear Estimation

The instructions \texttt{NLLS}, \texttt{NLSYSTEM}, \texttt{MAXIMIZE}, \texttt{FIND}, \texttt{CVMODEL}, and \texttt{DLM} all perform non-linear estimation over a list of parameters which you set. Before you can use these, you need to create the list of free parameters using \texttt{NONLIN}. \texttt{NONLIN} creates a data type called a \texttt{PARMSET} which allows you to switch between parameter sets quickly. \texttt{PARMSETS} can even be combined, making it easy to adjust parameter sets. \texttt{NONLIN} can also impose constraints on the parameters.

\begin{verbatim}
nonlin( options ) parameterfields (separated by blanks)
\end{verbatim}

Parameters

\texttt{parameterfields} A \texttt{parameterfield} is one of the following:

- a simple \texttt{REAL} variable, such as \texttt{B1}
- a real array (\texttt{VECTOR}, \texttt{RECTANGULAR} or \texttt{SYMmetric})
- an array of arrays, such as a \texttt{VECTOR} of \texttt{VECTORS}.
- a substitution operation: \texttt{B3=B1*B2}
- an equality constraint: \texttt{B3==B1*B2}
- an inequality constraint: \texttt{B3>=0.0}. You cannot use a strict inequality here. For instance, \texttt{B3>0.0} is illegal.

See the notes below on the use of these.

Options

\texttt{parmset=PARMSET} to define [default internal]

Using the \texttt{PARMSET} option, you can define or redefine a parameter set. By using the \texttt{PARMSET} option on your estimation instruction, you can switch easily from one parameter set to another. RATS maintains a single unnamed parameter set, which is the one used for estimation if you don't provide a named set.

\texttt{add/[noadd]}

This allows you to add parameters or constraints to a parameter set without reentering the full set. Changes are added to the default parameter set if you don't use the \texttt{PARMSET} option, or to the \texttt{PARMSET} being defined if you did.

\texttt{ADD} is largely obsolete because you can now separate the parameter set into different parts and “add” them using the \texttt{+} operator when you need to use them for estimation. That is, \texttt{MAXIMIZE(PARMSET=MODELPARMS+GARCH)} will combine the \texttt{MODELPARMS} and \texttt{GARCH} parameter sets to form the working parameter set.
Notes

If you use an array, it must be DECLAREd before the NONLIN. It does not need to be
DIMENSIONed until you are ready to use it. You can include inequality constraints on
the individual elements of an array, or on all of them at once. For instance,

declare vector b
nonlin b b>=0.0

constrains all elements of B to be non-negative.

Use a substitution constraint rather than an equality constraint wherever possible.
RATS handles B3=B1*B2 by setting B3 equal to B1*B2 every time B1 or B2 changes,
thus eliminating one free parameter. With B3==B1*B2, RATS estimates the three
parameters separately, and uses Lagrange multiplier methods to push the estimates
towards the constraint. This is a much slower process. The equality constraint should
be used only when you can’t easily solve out for one parameter in terms of the others.

The FRML instruction can be used to create a formula and matching PARMSET from a
linear equation or regression. This can be very handy when there is a linear model
for the mean whose form you don’t want to fix in advance.

Before you can use any of the estimation instructions, you must set initial values for
the parameters. COMPUTE and INPUT are the two simplest ways to accomplish this—
the two examples below show two ways of setting the same set of initial values:

compute b1=b2=b3=0.0,b4=1.0
input b1 b2 b3 b4
  0 0 0 1

Examples

nonlin b0 b1 b2 b3 b4

nonlin(parmset=base) b0 b1 b2 b3 b4
nonlin(parmset=constraint) b2>=0.0 b3>=0.0
...
maximize(parmset=base+constraint) ...

frml(lastreg,vector=b,addparms,parmset=garchps) meanmodel
dec vector archp(p) garchp(q)
nonlin(parmset=garchps,add) archp garchp

The first example puts five variables into the default parameter set. The second
defines a “base” and a “constraint” parameter set and combines them at estimation
time. The third creates the parameter set GARCHPS from the most recent regression
(using the vector B for the parameters), then adds the vectors ARCHP and GARCHP.
NPREG — Non-Parametric Regressions

NPREG does a non-parametric flexible fit for $Y_i = f(X_i)$ for a single $Y$ and $X$ series. There are three methods which can be chosen: the Nadaraya-Watson kernel estimator, LOWESS (LOcally WEighted Scatterplot Smoother) and Nearest Neighbor.

```
npreg( options )   Y series   X series   start   end   grid   fit
```

Parameters

- **$Y$ series**: The dependent variable.
- **$X$ series**: The explanatory variable.
- **start end**: Range to use in estimating the regression. If you have not set a `SMPL`, this defaults to the maximum range over which both the $Y$ series and $X$ series are defined.
- **grid**: Series of X values at which the fit is computed.
- **fit**: Series of fitted values corresponding to the grid series.

Options

- **method=[nadaraya]/lowess/nn**
  - `METHOD=NADARAYA` does the Nadaraya-Watson kernel estimator,
  - `METHOD=LOWESS` does LOWESS, and `METHOD=NN` does nearest neighbor smoothing. See the Technical Information for descriptions.

- **grid=[automatic]/input**
  - `GRID=AUTOMATIC` has NPREG generate the grid points for the fit. These range from the lowest to the highest values attained by the actual $X$ series, with the number of points being given by the MAXGRID option. To control the points yourself, use `GRID=INPUT`, in which case the grid series should be filled in advance with your settings. Usually, an equally spaced grid is handy if you’re mainly interesting in examining the shape of the $f$ function. If the NPREG is part of a more complex calculation, the grid series will usually be the $X$ series itself.

- **type=[epanechnikov]/triangular/gaussian/logistic/flat/parzen**
  - `TYPE` selects the kernel type for the Nadaraya-Watson estimator.
  - `BANDWIDTH` specifies the bandwidth for the kernel. The default value is

  $$\frac{(0.79 \text{IQR})}{N^{1/5}}$$

  where IQR is the interquartile range (75%ile–25%ile) of the $X$ series and $N$ is the number of data points.
fraction=fraction of data range included in a LOWESS/NN fit [.5]

The larger the value for this option, the “stiffer” is the function.

smpl=SMPL series or formula (User’s Guide, Section 5.2)

You can supply a series or a formula that can be evaluated across entry numbers. Entries for which the series or formula is zero or “false” will be skipped, while entries that are non-zero or “true” will be included in the operation.

Technical Information

The Nadaraya-Watson estimator is:

\[
\hat{f}(x) = \frac{\sum K((x_i - x)/h) y_i}{\sum K((x_i - x)/h)}
\]

where \( K \) is the kernel function and \( h \) is the bandwidth. The kernel types take the following forms:

- **EPANECHNIKOV**
  
  \( K(v) = 0.75(1 - v^2) \) if \(|v| \leq 1\), and 0 otherwise

- **TRIANGULAR**
  
  \( K(v) = (1 - |v|) \) if \(|v| \leq 1\), and 0 otherwise

- **GAUSSIAN**
  
  \( K(v) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{v^2}{2}\right) \)

- **LOGISTIC**
  
  \( K(v) = e^v / (1 + e^v)^2 \)

- **FLAT**
  
  \( K(v) = 0.5 \) if \(|v| \leq 1\), and 0 otherwise

- **PARZEN**
  
  \( K(v) = 4/3 - 8v^2 + 8|v|^3 \) if \(|v| \leq 0.5\), \( 8(1-|v|^3)/3 \) if \( 0.5 \leq v \leq 1\), and 0 otherwise

As you increase the bandwidth, the estimated function becomes smoother, but is less able to detect sharp features. A shorter bandwidth leads to a more ragged estimated function, but sharp features will be more apparent.

LOWESS is computed by doing a weighted least squares regression of \( y \) on a constant and \( x \), for the sample which includes the requested fraction of the data closest to each value of \( x \). The weights are

\[
\left(1 - \left(\frac{|x_i - x|}{D(x)}\right)^3\right)^3
\]

where \( D(x) \) is the range of the sample \( X \)'s used in the fit at \( x \). \( \hat{f}(x) \) is the intercept of this regression.

NN takes the simple average of \( y \) for the requested fraction of the data closest to each value of \( x \).
Examples

This example implements several non-parametric regressions from the NIST Engineering Statistics Handbook. See the file LOWESS.PRG for the complete code and the web link for the NIST document.

The first estimates using LoWess with \texttt{FRAC=.33}. The default \texttt{FRAC=.5} would give a “stiffer” function, and we suspect that .33 was picked (by NIST) because of that. The second does Nadaraya-Watson with the default bandwidth. As this seems to produce a bit too stiff a function as well, this is refit with a shorter bandwidth. (You can look at the \texttt{%EBW} variable to see what bandwidth is used on the first \texttt{NPREG}).

All of the estimated functions are graphed using \texttt{SCATTER}, with the original data done as dots, and the functions done as lines. Note the use of \texttt{OVSAME} on these. If you don’t use it, the function could end up being graphed on a different scale than the data.

```plaintext
allocate 21
data(unit=input,org=columns) / seq x y

See the LOWESS.PRG file for the actual data

npreg(method=lowess,frac=.33) y x / xv yv
scatter(style=dots,overlay=line,ovsame,$
   header="LOWESS Fit, Frac=.33") 2
# x y
# xv yv
*
npreg(method=nadaraya,grid=input) y x / xv yn
scatter(style=dots,overlay=line,ovsame,$
   header="Kernel Fit, Default Bandwidth") 2
# x y
# xv yn
npreg(method=nadaraya,grid=input,bandwidth=2.0) y x / xv yx
scatter(style=dots,overlay=line,ovsame,$
   header="Kernel Fit, Smaller Bandwidth") 2
# x y
# xv yx
```
**OPEN — Opening I/O Units**

**OPEN** opens a new or existing file as a RATS I/O unit. RATS I/O units are used in the UNIT option of a number of RATS instructions. You can also use **OPEN** to create a text window into which you can direct selected output. When working in interactive mode, you can open files using operations on the *File* menu.

```
open( option ) RATS I/O unit filename
```

**Parameters**

- **RATS I/O unit**: The name of the unit you wish to open.
- **filename**: The name of the file or window to associate with the I/O unit.

**Options**

- **window/[nowindow]**
  If you use the WINDOW option, a text window will be opened with the title that you provide in the filename field. Such a window can be used for text output.

- **append/[noappend]**
  If you are writing output to a file, you can use the APPEND option to append any new output to the end of the current file. By default, the contents of the existing file (if any) are destroyed.

- **immediate/[noimmediate]**

- **format=[free]/wks/prn/xls/rats/dbf/dif/portable/"(FORTRAN format)"/binary/html/cdf**

- **write/[nowrite]**

- **status=INTEGER variable returning status of open attempt [unused]**
  Normally, **OPEN** just associates a filename with a unit name. The file isn’t actually opened until you use an instruction like **DATA** or **COPY**. Use the IMMEDIATE option if you want to open the file immediately. The FORMAT option specifies the format of the file (see **COPY** or **DATA** for details on these). **WRITE** opens the file for writing (output), otherwise it will be opened for reading (input). If you use the STATUS option, the variable you supply will be set to 1 if the file was opened successfully, and 0 otherwise.

**RATS I/O Units**

RATS has a number of standard I/O units:

- **INPUT**: Source file (or device) for RATS instructions.
- **OUTPUT**: File (or device) for output from RATS.
- **DATA**: Source file for data read into RATS.
- **COPY**: Output file primarily for data being written from RATS.
**Examples**

```
open data rates.rat
data(format=rats)
```

opens RATES.RAT as a data file and reads from it (DATA is the default I/O unit for the DATA instruction).

```
open copy teststat.fil
do i=1,draws
  ...
  write(unit=copy) tstats
end do i
```

opens the file TESTSTAT.FIL as a “copy” file, and writes output to it from inside a loop. Note that you put the OPEN instruction outside the loop. If you put it inside, each OPEN will erase the old results.

```
open(window) tests "Test Results"
display(unit=tests) "Test Statistic=" %cdstat "P-value" %signif
```

opens the unit TEST as a window titled “Test Results” and puts information from the DISPLAY instruction into it.

**Notes**

You can use the instruction SOURCE to switch the INPUT unit to a different file. Unlike OPEN INPUT, SOURCE returns to the original file when the instructions on the second file have been executed, which is why it is preferred for accessing procedures.

You can use the instruction CHANGE to redefine an I/O unit, which can be particularly useful for redirecting output to different files. See CHANGE for details.

The instruction CLOSE closes its associated file. A text file is usually left “open” after an instruction writes to it. In general, an open file can’t be read from another program. If you’re done with a unit, issuing a CLOSE instruction for it will make it possible to process the information with another application.
OPTION — Defining Options for PROCEDURES

Use **OPTION** to define options for a **PROCEDURE**. When you execute the procedure, these options are very similar to options for standard RATS instructions. All **OPTION** statements should be near the beginning of the procedure. See Section 16.2.2 of the *User’s Guide* for more information.

<table>
<thead>
<tr>
<th>option switch</th>
<th>option name</th>
<th>Default (1 for <strong>ON</strong>, 0 for <strong>OFF</strong>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>option choice</td>
<td>option name</td>
<td>Default number</td>
</tr>
<tr>
<td>option</td>
<td>data type</td>
<td>option name</td>
</tr>
</tbody>
</table>

**Parameters**

**option name**

This is the name which you are assigning to the option. Within the procedure, you must use its full name. However, on the **EXECUTE** instructions which invoke the procedure, only the first three characters are significant, so make sure you don’t use conflicting names (RATS will warn if you do so). Also, do not use a name which begins with the letters **NO**.

The option names are local to the procedure, so you are free to use the same option names in other procedures.

By default, all options are passed by value. **SWITCH** and **CHOICE** options are **always** passed by value. You cannot change the value of such an option within the **PROCEDURE**.

If you want an option to return a value, you need to pass it by address. To do this, use **option name** rather than **option name** alone (*User’s Guide*, Chapter 16).

**Default**

(for **SWITCH**): Use the value 0 if you want the option “off” by default, or the value 1 if you want it to be “on”. If you don’t specify a value, this defaults to 0 (off).

**Default number**

(for **CHOICE**): the listed choice (by position in the **List of Choices**, not by name) you want to be the default. Use a 0 if you want the default to be “not selected.”

**List of Choices**

(for **CHOICE**): the keywords (separated by blanks) that you want to represent the choices. Only the first three letters are significant. If you are using a **PROCEDURE** option to pass a choice for a standard RATS instruction, you should list the choices in exactly the order used in the RATS manual.

**data type**

For value options, this can be any of the RATS supported data types.
**Option**

*Default value* (for value options). Omit this if you want the default for the option to be “not selected.” You can use global variables or procedure parameters in a *Default value* expression. However, all such variables must have been introduced before the **OPTION** instruction. You cannot use local variables. Subject to these restrictions, *Default value* can be

- for *any* option passed by address: a single variable or array element of the proper type.
- for REAL, INTEGER, COMPLEX, LABEL or STRING: any expression of the proper type.
- for SERIES or EQUATIONS: a series or equation name, or an integer expression.
- for arrays, FRMLS, MODELS and PARMSETS: a single variable (global variable or parameter) of the proper type.

**Coding Within the Procedure**

You can use the option names as local variables of the indicated type within the procedure. However, if an option is passed by value, you cannot set it within the procedure. If you need to do something like that (if, for instance, a **SWITCH** has to be **ON** if some other option is used), set a local variable equal to the option, and change the local variable.

- **SWITCH** options are INTEGER with 0 or 1 values
- **CHOICE** variables are INTEGER with values 1,2,...; or 0 if there is no default and the option isn’t used on the **EXECUTE**. Note that the value of a **CHOICE** variable is not equal to the keyword for the choice; it is equal to the position in the list of choices.

**Passing Through to RATS Instructions**

A procedure option can be used to pass choices to the options on the standard RATS instructions by using

```
instruction option name=option name ; e.g. PRINT=PRINTOUT
```

This works even for **SWITCH** and **CHOICE** options. For instance,

```
procedure doreg depvar
type series depvar
option switch printout 1
...
linreg(print=printout, ...)
```

The **PRINT** option on **LINREG** will be “off” if you use **NOPRINTOUT** on the **EXECUTE** and “on” otherwise.
The %DEFINED Function

You can use the function %DEFINED(option or parameter name) to determine whether or not an option or parameter was given a value when the procedure was executed. It returns 1 if it was defined and 0 otherwise. If an option has a default value, it will always be “defined”, since it will, at minimum, get the default value. We use %DEFINED extensively in our procedures to ensure that the user executed the procedure in the proper form. See the second example below.

Examples

```
option  choice  type 1  flat  tent
```

The TYPE option has choices FLAT and TENT with FLAT (choice 1) the default.

```
option  real  variance 1.0
option  symm  vmatrix
```

Option VARIANCE is REAL with a default of 1.0; option VMATRIX is SYMMETRIC with default of “not selected”.

The code below is taken from the TSAYTEST procedure, which does an arranged regression test for threshold autoregression. It is included with RATS on the file TSAYTEST.SRC.

This procedure requires that the user supply a series of threshold values via the THRESHOLD option. The code below checks to make sure that a series has been supplied for this option, as well as for the dependent variable parameter. If the user has failed to supply either item, the procedure generates a message detailing the required syntax, and then exits from the procedure.

```
if .not.%defined(threshold).or..not.%defined(depvar)
{
    display $
    "Syntax: @tsaytest(threshold=threshold series) depvar start end"
    display $
    " # list of regressors"
    return
}
```
ORDER — Sorting Data and Ranking Data

ORDER sorts a series and can reorder a data set based upon the values of one series. Optionally, it can construct a rank ordering of a series.

```
order(options) series start end list of series
```

Parameters

**series**
Series to sort or rank.

**start** and **end**
Range of entries to sort or rank. If you have not set a **SMPL**, this defaults to the defined range of **series**.

**list of series**
(Optional) The listed series are reordered in parallel with **series**. Observations are kept intact across **series** and **list of series** and reordered as a group based upon the values of **series**. To reorder **all** current data series, use the option **ALL**.

Options

**decreasing/[nodecreasing]**
Use **DECREASING** to sort or rank in decreasing order.

**rank=Series for ranks**
Use this to get the rank ordering of **series**. The original **series** is left untouched. **ORDER** assigns ties the average rank for the value.

**all/[noall]**
Use **ALL** to reorder **all** current data series based upon **series**.

**smpl=SMPL series or formula** (Section 5.2 of the User’s Guide)
Limits the sort to the observations for which the series or formula returns a non-zero or true value.

**index=SERIES of INTEGERS for index series [not used]**
This stores the entry numbers which would sort the series, that is:

```
order(index=ix) x
set y = x(ix)
```

would make **Y** a sorted copy of **X**. The original **series** is left untouched.

This is similar to **RANK**, except that **INDEX** returns actual entry numbers, rather than returning a ranking from 1 to **N**.
Examples

order pop / spend tax aid

reorders POP, SPEND, TAX and AID based upon POP.

order(rank=xrank) xseries
order(rank=yrank) yseries
linreg yr ank
# constant xrank

ranks XSERIES and YSERIES and regresses the Y-ranks on the X-ranks. The coefficient and $T$-stat on the XRANK coefficient give Spearman’s rank correlation test.

Missing Values and SMPL Series

Missing values are always put at the end of the sort, regardless of the direction of comparison.

If you use the SMPL option or have set a global SMPL series using the SMPL instruction, ORDER will put all excluded entries at the end of the sort. If you reorder your SMPL series, you need to redo the SMPL instruction.

Technical Information

ORDER uses a “Shell sort.” The sort time varies on the order of $N^{1.5}$, where $N$ is the number of data points. It should provide acceptable performance with virtually any typical RATS data set.

If you use the RANK option, ORDER assigns to all data points involved in a tie the average of the ranks. The smallest (if you are doing the default sort in increasing order) gets a rank value of 1.

On a sort, ORDER breaks ties by keeping data points in their original entry order. Thus, if you do ORDER(ALL) on series A, then ORDER(ALL) on series B, the result will be a data set sorted first on B and then A for tied values of B.

See Also...

The functions %SORT, %SORTC, %SORTCL, and %RANKS can be used to sort or rank data in a vector or matrix. The instruction STATISTICS with the FRACTILES option computes a set group of fractiles for a data series, and the function %FRACTILES computes fractiles for a vector or matrix.
**Panel**

**PANEL — Panel Data Transformations**

**PANEL** computes linear combinations of a panel data series, its time averages or sums and cross-section averages or sums. You can use it to implement regressions on panel data which can't be done by using **PREGRESS**. See Chapter 14 of the *User’s Guide* for more information on panel data.

You can also use the **SPREAD** option to compute an individual by individual variance series, which can be used in a **SPREAD** option on **LINREG** for weighted least squares.

\[
\text{panel}(\text{options}) \quad \text{series start end newseries newstart}
\]

**Parameters**

- **series**
  Original series.

- **start end**
  Range of entries to use. If you have not set a **SMPL**, this defaults to the defined range of **series**.

- **newseries**
  Series for transformed values of **series**. By default, newseries=series.

- **newstart**
  The starting entry for the data in newseries. By default, newstart=start.

**Options**

- **entry=** Weight on value of series [0.0]
- **indiv=** Weight on the individual mean [0.0]
- **time=** Weight on the time mean [0.0]
- **icount=** Weight on the number of individuals [0.0]
- **tcount=** Weight on the number of time periods [0.0]
- **isum=** Weight on the individual sum [0.0]
- **tsum=** Weight on the time sum [0.0]
  These supply the weights on the various components. See the next page for details.

- **smpl=** **SMPL** series or formula (*User’s Guide*, Section 5.2)
  This is the standard **SMPL** option. Any observations for which the series or formula is 0 or “false” will be ignored when **PANEL** computes averages.

- **compress/[nocompress]**
  If the transformation takes individual statistics only, or time statistics only, you can use **COMPRESS**. It eliminates the repetitions and creates a series of length \( N \) for **INDIV** and length \( T \) for **TIME** (as opposed to length \( N \times T \)).
spread=series of individual variances [not used]

With SPREAD, PANEL computes a “SPREAD” series by setting each entry equal to the sample variance of series for the entries of its cross-section. You should do this separately from other transformations. This SPREAD series is in a form directly usable in a SPREAD option for LINREG (Section 5.4). Note that this computes a centered variance.

Description

With \( y_{it} \), \( i=1,\ldots,N \); \( t=1,\ldots,T \) representing series, for entry \( it \):

- **ENTRY** is \( y_{it} \)
- **INDIV** is \( y_{i*} \), the mean of \( y \) for individual \( i \), averaged across \( t \)
- **TIME** is \( y_{*t} \), the mean of \( y \) for time \( t \), averaged across \( i \)
- **ISUM** is the sum across \( t \) of \( y \) for individual \( i \)
- **TSUM** is the sum across \( i \) of \( y \) for time period \( t \)
- **ICOUNT** is the number of individuals across \( t \) of \( y \)
- **TCOUNT** is the number of time periods across \( i \) of \( y \)

For example, you would use the following options to create the indicated series:

- For \( y_{it} - y_{i*} \), use options ENTRY=1.0, INDIV=-1.0
- For \( y_{it} - \theta y_{*t} \), use options ENTRY=1.0, TIME=-THETA

Missing Values

PANEL removes any missing values from any average it calculates.

Example

A fixed effects estimator for \( Y \) on \( X_1 \) and \( X_2 \) with 40 individuals, 5 time periods:

```
panel(entry=1.0,indiv=-1.0) y / ytwid
panel(entry=1.0,indiv=-1.0) x1 / x1twid
panel(entry=1.0,indiv=-1.0) x2 / x2twid
linreg(dfc=40) ytwid
# x1twid x2twid
```

The DFC option (User’s Guide, Section 15.5) corrects for the 40 degrees of freedom lost from subtracting out individual means. This can be done more easily with PREGRESS:

```
pregress(effects=indiv,method=fixedeffects) y
# x1 x2
linreg y / u
# constant x
panel(spread=panels) u
linreg(spread=panels) y / u
# constant x
```

computes weighted least squares estimates, with variances in each cross-section determined empirically.
PForm

PFORM — Forming Panel Data Series

**PFORM** forms a panel data series from either a set of several series (each containing data for one individual), or a single series which contains data for both individuals and time periods, but in a different ordering than the one required by RATS. **PFORM** is usually employed in a short program which rearranges and rewrites the data set. The new data set is then analyzed with other RATS programs.

See Section 14.5 of the *User's Guide* for general information on using panel data. See the “Description” section on the next page for details on using **PFORM**.

```
pform(options) newseries start end # list of input variables
```

**Parameters**

- **newseries** The series to be constructed with correct form for a RATS panel data set.
- **start** **end** The range of the input series to use.

**Supplementary Card**

- **input variables** This can be a list of several series (each containing data for one individual), or a single series that needs to be re-ordered to conform to the panel organization supported by RATS.

**Options**

- **transpose/[notranspose]**
- **block=number of individuals per time block**
  If the input is a single series containing balanced panel data (each individual has data for the same set of time periods), but is blocked by time, rather than by individual, the combination of **TRANSPOSE** plus **BLOCK** will cause the input variable to be transposed into a series blocked by individual.

- **individual=series with identifiers for individuals [not used]**
- **time=series with identifiers for time periods [not used]**
  These allow you to supply index series with numeric codes identifying the individual and the time period for the corresponding entry of *input series*. These are useful if you have an unbalanced panel (different individuals have different numbers of observations), or if the series is not in any regular order.
Description

PFORM constructs newseries as a series with the proper panel form: balanced (same number of time periods for each individual, although the series can contain missing values), and blocked by individuals (entries 1 through \( T \) contain all data for individual 1, entries \( T+1 \) to \( 2T \) contain all data for individual 2, etc.). It constructs one series at a time. PFORM can handle three different forms of input data:

Each individual is in a separate series
List the set of series on the supplementary card. Set \( \text{start} \) and \( \text{end} \) to the range of each of these you want included in the concatenated series. If you leave \( \text{start} \) and \( \text{end} \) blank, they will default to the maximum range covered by the input series, with missing values used to pad individual’s records which are shorter.

Single series, balanced panel, but data blocked by time, not individual
If you have a series blocked by time rather than by individual (first \( N \) observations contain data for time period 1 for all individuals, next \( N \) observations contain data for time period 2, etc.), supply the series on the supplementary card, use the TRANSPOSE options to tell RATS to reorder the data, and the BLOCK option to tell it the number of individuals.

Single series, possibly unbalanced, with a separate index series
If your series is unbalanced, or isn’t in any regular order whatsoever, you can still form a panel series if you have separate index or “tag” series that identify the individuals and time periods. List the single input series on the supplementary card and use the INDIVIDUAL and TIME options to supply your index series. If you use TIME but not INDIVIDUAL, RATS assumes that the first time a value of the TIME series occurs, it is on an observation for the first individual; the second time is for the second individual, etc. Similar assumptions apply if you use INDIV but not TIME.

Regardless of the options, transform each of the series in your data set first. Then set the CALENDAR to describe your data set and either continue your analysis, or save the data to a new data file. The variable \%NOBS is set to the number of observations per individual, which is the proper value for the PANELOBS option on CALENDAR.
Examples

\begin{verbatim}
cal(a) 1935:1
all 1954:1
open data grunfeld.wks
data(format=wks,org=cols) / ige fge cge iwest fwest cwest
pform invest
  # ige iwest
pform firmvalue
  # fge fwest
pform cstock
  # cge cwest

Now change to a panel data CALENDAR and write the data to a RATS format file:
cal(a,panelobs=%nobs) 1935:1
copy(format=rats) / invest firmvalue cstock

concatenates data from two companies to form three output series.

all 30*8
open data county.xls
data(format=xls,org=cols) / unemp expend
cal(a,panelobs=%nobs) 1990:1
pform(indiv=county,time=year) punemp
  # unemp
pform(indiv=county,time=year) pexpend
  # expend
copy(format=xls,org=cols) / punemp pexpend

reads a dataset with thirty counties and eight time periods per individual, blocked by time, and creates a new data set blocked by individual.

Suppose you have the same data set as above, except that data are in random order rather than sorted by time. Assume that you also have two index series, COUNTY and YEAR, identifying the county and year, respectively, of the corresponding entries of UNEMP and EXPEND. For example, the COUNTY series might use numbers 1 through 30 to identify counties, while YEAR might contain the values 1990, 1991, etc.

To form panel series from these, you would use the INDIVIDUAL and TIME options:

all 30*8
open data county.xls
data(format=xls,org=cols) / unemp expend county year
pform(indiv=county,time=year) punemp
  # unemp
pform(indiv=county,time=year) pexpend
  # expend
cal(a,panelobs=%nobs) 1990:1
copy(format=xls,org=cols) / punemp pexpend
\end{verbatim}
**POLAR: Polar Decomposition**

**POLAR** computes the polar decomposition of a complex series. A complex number may be decomposed as

\[ z = |z| \exp(i\theta), \ -\pi < \theta < \pi \]

where \(|z|\) is the *modulus* or absolute value and \(\theta\) is the *argument*.

```
polar(option)  cseries  start  end  modulus  argument  newstart
```

**Parameters**

- **cseries**: Complex series which **POLAR** is to decompose.
- **start end**: Range of entries to process. By default, the defined range of **cseries**.
- **modulus**: (Optional) Complex series for the modulus of each entry. Use * here if you want the *argument* but not *modulus*.
- **argument**: (Optional) Complex series for the argument of each entry. Use * here if you want to use newstart but not argument.
- **newstart**: Starting entry for *modulus* and *argument*. By default, same as *start*.

**Options**

- **periods**/**[noperiods]**
  - If you use the **PERIODS** option, **POLAR** converts the phase lead (*argument*) from radians to periods. At frequency \(\nu\), a phase lead of \(\theta\) is equivalent to a lead of \(\theta/\nu\) periods. *This conversion is meaningful only for frequencies 0 to \(\pi\).*

**Notes**

For individual complex numbers, you can get the polar decomposition using the functions `%CABS(z)` and `%ARG(z)`. These are real-valued functions which return the modulus and the argument of the complex number \(z\).

If you try to do the polar decomposition of an *unsmoothed* cross periodogram, you will find that you get a coherence of 1.0, as, in effect, you are estimating a separate relationship at each frequency. Make sure that you smooth everything first.
Example

**POLAR** can compute coherences and phase leads for cross-spectral analysis. In the following example, series 6, 7 and 8 are the two spectral densities and the cross-spectral density of a pair of series. **POLAR** sets 9 as the series of coherences and 10 as the phase leads. The phase leads are converted to periods. These two series are then sent back to the time domain and graphed using **SCATTER** with a “production-quality” setup, using x-axis labels showing fractions of \( \pi \), and separate scales for each. The coherence is forced onto a scale of 0.0 to 1.0.

```plaintext
fft 1
fft 2
cmult 1 1 / 3
cmult 2 2 / 4
cmult 1 2 / 5
window 3 / 6
window 4 / 7
window 5 / 8
cset 8 = %z(t,8)/%csqrt(%z(t,6)*%z(t,7))
polar(periods) 8 / 9 10
*
ctor 1 nords/2
# 9 10
# coher phase
*
set freq 1 nords/2 = 2.0*(t-1)/nords

Use Symbol font to get \( \pi \)'s for the axis labels. This doesn’t affect the numbers.

grparm(font="Symbol") axislabels 14

These are good labels for monthly data. For quarterly data, labeling at 0, \( \pi/4 \), \( \pi/2 \), \( 3\pi/4 \) and \( \pi \) is more sensible.

compute [vect[strings]] flab=$
  |"0","p/6","p/3","p/2","2p/3","5p/6","p"|
scatter(style=line,vmin=0.0,vmax=1.0,vlabel="Coherence",$
  overlay=line,ovlabel="Phase Lead (periods)",$
  twoscale,xlabels=flab) 2
# freq coher
# freq phase
```

We're not scaling the periodograms as the scale factors will wash out when we define series 8 below.
**PREGRESS — Panel Data Regressions**

**PREGRESS** estimates a regression on a panel data set using one of several specialized techniques. See Sections 14.5 through 14.7 of the *User’s Guide* for more information on the use of panel data in RATS.

```
pregress(options) depvar start end resids
# list of explanatory variables in regression format
```

**Wizard**

The *Panel Data Regressions* wizard on the *Statistics* menu provides dialog-driven access to the **PREGRESS** instruction.

**Parameters**

- **depvar**  
  Dependent variable.

- **start**  end  
  Estimation range. If you have not set a **SMPL**, this defaults to the maximum common range of all the variables involved.

- **resids**  (Optional) Series for the residuals. These will be the transformed residuals.

**Description**

This estimates \( \beta \) in the linear regression

\[
y_{it} = X_{it} \beta + u_{it}, \tag{1}
\]

\[
u_{it} = \epsilon_i + \lambda_t + \eta_{it}, \tag{2}
\]

unless \( \text{METHOD=SU} \). \( \epsilon \) is the individual effect, \( \lambda \) is the time effect and \( \eta \) the purely random effect. If you use the option \( \text{EFFECTS=INDIV} \) or \( \text{METHOD=FD} \), the decomposition only includes the \( \epsilon \) and \( \eta \) components. With \( \text{EFFECTS=TIME} \), it only includes \( \lambda \) and \( \eta \).

If you use \( \text{METHOD=FIXED} \), \( \epsilon_i \) and \( \lambda_t \) are treated as constants and are “swept” out. If you request \( \text{METHOD=RANDOM} \), they are treated as part of the error term and \( \beta \) is estimated by GLS. If \( \text{METHOD=FD} \) (first difference), the data are differenced to eliminate \( \epsilon_i \). \( \text{METHOD=SU} \) assumes that the \( u \)'s are serially uncorrelated, but are correlated across \( i \) at a given \( t \).

For random effects estimation, you can input the variances of the components yourself using the \( \text{VRANDOM} \), \( \text{VINDIV} \) and \( \text{VTIME} \) options, or you can allow **PREGRESS** to estimate them. It does that by running a fixed effects regression and estimating the components from the resulting residuals. This is one of several possible methods for (consistently) estimating the component variances. **PSTATS** applied to the residuals of a simple least squares regression is another, and will give different values.
Options

`effects=[individual]/time/both`
This indicates whether to allow for INDIVIDUAL effects, TIME effects or BOTH.

`method=[fixedeffects]/randomeffects/fd/sur/between`
This chooses the estimation method: fixed or random effects, first difference, cross-section SUR, or the “between” estimator.

`vrandom=variance of the random component [estimated]`
`vindiv=variance of the individual component [estimated]`
`vtime=variance of the time component [estimated]`
With METHOD=RANDOM, these allow you to input variances of the components. If you don’t supply these, they are estimated by first running a fixed effects regression.

`indiv=series of individual effects [not used]`
`time=series of time effects [not used]`
With METHOD=FIXED or METHOD=RANDOM, these allow you to retrieve the coefficients on the individual or time components; whichever ones are estimated based upon your choice for the EFFECTS option. These are produced to match up with the entries on the original data, so, for instance, the output values for the individual effects will be repeated for each time period within each individual's block of entries. If you want to compress out the duplicates, you can use the PANEL instruction with the COMPRESS option.

`[print]/noprint`
`vcv/[novcv]`
`smp=SMPL series or formula (User's Guide, Section 5.2)`
`unravel/[nounravel] (Section 5.11)`
`define=equation to define (Section 1.1.9)`
`frml=formula to define`
`equation=equation to estimate`
`dfc=Degrees of Freedom Correction (Section 5.15)`
`title=title to identify estimation method [depends upon options]`
See LINREG for details. If you use EQUATION, omit the supplementary card.

Variables Defined by PREGRESS

Besides the usual regression variables (%XX, %BETA, %LOGL and %RSS, etc.) PREGRESS defines:

`%VRANDOM` variance of η: the random component
`%VINDIV` variance of ε: the individual component
`%VTIME` variance of λ: the time component
`%SIGMA` covariance matrix for METHOD=SUR
Examples

```
preg(method=between) logp
# constant rnd
preg(method=fixed) logp
# rnd
preg(method=random) logp
# constant rnd
preg(method=random,vindiv=.504,vrandom=.134) logp
# constant rnd
```

This estimates an equation, allowing for individual effects only, using the between estimator, fixed and random effects, then random effects with input variances. The CONSTANT is omitted from the fixed effects estimator, as any time-invariant regressor is zeroed out when the individual means are subtracted. If you include CONSTANT in fixed effects, it will show a zero coefficient with zero standard error.

```
preg vfrall
# beertax
preg(effects=both) vfrall
# beertax
```

This estimates an equation, first allowing for individual effects only, then allowing for both individual and time effects.

Notes

If you estimate using fixed effects, the reported degrees of freedom will be reduced by the number of implied “dummy” variables.

For a balanced sample, the coefficient estimates for both fixed and random effects will be identical to those you would get doing the equivalent regression “by hand,” using PANEL to transform the data. The covariance matrix will be slightly different with random effects because of a different estimate of the variance. In an unbalanced sample, the results will be the same for fixed effects, but only for EFFECTS=TIME or EFFECTS=INDIV. Random effects on an unbalanced data set should only be done using PREGRESS.
PRINT — Printing Data

PRINT is the simplest instruction for displaying the values of data series to the output window or to a file (PRINT has nothing (physically) to do with printers). COPY is a more complex and more flexible instruction for this.

PRINT deals only with data series, not matrices or expressions. Use DISPLAY for those.

\[
\text{print( options ) start end list of series}
\]

**Wizard**

You can also view series by using the *Show Series Window* wizard on the Data menu, selecting the series you want to view, and clicking on the View Data toolbar icon: 

**Parameters**

- **start end** Range of entries to print. If you have not set a SMPL, PRINT uses the smallest range required to show all defined data for the series. Any undefined data are treated as missing values.
- **list of series** The list of series to print. If you omit the list, all current series are printed.

**Options**

- **[dates]/nodates** By default, PRINT labels entries with their dates if you have set a CALENDAR. Use NODATES to get just the entry number instead.
- **number=labeling number for first entry** Use this if you want to label the entries with a sequence of numbers different from the entry numbers. For instance, if you print a sequence of numbers which logically should be labeled –10 to 10, use the option NUMBER=-10.
- **picture=picture clause for data** By default, PRINT finds the “minimal” representation for each data series displayed, using the smallest number of digits to represent exactly all the data in the series. However, for derived series like residuals, this means showing fifteen or more non-zero digits, including many to the right of the decimal.

If you find that distracting, you can use PICTURE to reduce the number of decimals. A picture clause takes a form like "##.###" or "*.#". The first requests two digits left of the decimal and three digits to the right. The second asks for one digit right and as many digits as are needed to the left. See DISPLAY for more on picture codes.
**smpl=SMPL series or formula**
You can supply a series or a formula that can be evaluated across entry numbers. Only entries for which the series or formula are non-zero will be displayed.

**width=field width [15]**
The `WIDTH` option controls how wide a field each series gets. The default of 15 is quite wide—more than enough for most data.

**window="Title of window"**
If you use the `WINDOW` option, the output is displayed in a (read-only) spreadsheet window. The series will be in columns, with labels across the top row. You can export the contents of this window to various file formats using `File–Export`.

**Description**
This displays entries `start to end` of the `list of series`. See the sample output below. If all series will not fit across the page, `PRINT` will put them in blocks of between four and seven. Note, by the way, that `PRINT` determines the default range (`start to end`) separately for each block.

**Missing Values**
Missing values and data outside the range of a series are printed as NA.

**Examples**

```
print 1920:1 1929:1 prod profit capital
```

<table>
<thead>
<tr>
<th>ENTRY</th>
<th>PROD</th>
<th>PROFIT</th>
<th>CAPITAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1920:01</td>
<td>44.9</td>
<td>12.7</td>
<td>182.8</td>
</tr>
<tr>
<td>1921:01</td>
<td>45.6</td>
<td>12.4</td>
<td>182.6</td>
</tr>
<tr>
<td>1922:01</td>
<td>50.1</td>
<td>16.9</td>
<td>184.5</td>
</tr>
<tr>
<td>1923:01</td>
<td>57.2</td>
<td>18.4</td>
<td>189.7</td>
</tr>
<tr>
<td>1924:01</td>
<td>57.1</td>
<td>19.4</td>
<td>192.7</td>
</tr>
<tr>
<td>1925:01</td>
<td>61.0</td>
<td>20.1</td>
<td>197.8</td>
</tr>
<tr>
<td>1926:01</td>
<td>64.0</td>
<td>19.6</td>
<td>203.4</td>
</tr>
<tr>
<td>1927:01</td>
<td>64.4</td>
<td>19.8</td>
<td>207.6</td>
</tr>
<tr>
<td>1928:01</td>
<td>64.5</td>
<td>21.1</td>
<td>210.6</td>
</tr>
<tr>
<td>1929:01</td>
<td>67.0</td>
<td>21.7</td>
<td>215.7</td>
</tr>
</tbody>
</table>

```
print(picture="*.###") 1920:1 1925:1 rcons rinv
```

<table>
<thead>
<tr>
<th>ENTRY</th>
<th>RCONS</th>
<th>RINV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1920:01</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>1921:01</td>
<td>-0.463</td>
<td>-1.320</td>
</tr>
<tr>
<td>1922:01</td>
<td>-0.616</td>
<td>0.257</td>
</tr>
<tr>
<td>1923:01</td>
<td>-1.304</td>
<td>0.860</td>
</tr>
<tr>
<td>1924:01</td>
<td>-0.246</td>
<td>-1.594</td>
</tr>
<tr>
<td>1925:01</td>
<td>0.229</td>
<td>0.259</td>
</tr>
</tbody>
</table>
PRJ — Fitted Values/Normal Distribution Statistics

PRJ (short for PRoJect) computes fitted values and standard errors of projection based upon the most recent regression. You can use it either in or out of sample: in the latter case it computes simple static forecasts.

\[ \text{PRJ}(\text{options}) \ | \ \text{series} \ | \ \text{start} \ | \ \text{end} \]

Wizard

You can use the Single-Equation Forecasts wizard on the Statistics menu to forecast univariate models, including static forecasts as produced by PRJ.

Parameters

- **series**: Series for the fitted values. If you use the options described later under “Distribution Statistics”, these are the normalized fitted values \(z_i\) in the notation there.
- **start** and **end**: Range of entries for which fitted values are to be computed. If you have not used the SMPL instruction to set a range, this defaults to the range of the most recent regression. Note: using the SMPL option on the preceding regression has no effect on the range set by PRJ.

Description

PRJ handles forecasts for only certain types of models and certain situations. Use UFORECAST, FORECAST, or STEPS if you need more flexibility.

You can use PRJ to get fitted values after a LINREG, STWISE, DDV, LDV, AR1, or BOXJENK, although some statistics cannot be computed for AR1 and BOXJENK.

PRJ also has several options for computing distribution statistics from the fitted values. These are important for programming truncated and censored regressions, and for diagnostic tests in probit and related models (see User’s Guide, Section 14.3).

Fitted Values

PRJ takes the coefficients \(\beta\) and the regressors \(x\) from the most recent regression and computes the fitted values \(x_t \beta\) over entries \text{start} to \text{end}. For a logit or probit, this gives the index value for the case.

When you use PRJ outside the regression range, it computes a simple form of forecast called a static forecast: predicting the dependent variable given the values of all the regressors. This is useful only for models with no lagged dependent variables. Note that you must have data available for the right-hand-side variables in order to compute forecasts.
Options (Fitted Values)

Note that these cannot be computed for AR1 and BOXJENK because the covariance matrix for those is from a restricted non-linear model, so these formulas won’t apply.

**stderr=** series for standard errors of projection

This option computes the series of standard errors of projection:

\[
\hat{s} \sqrt{1 + x_i(X'X)^{-1}x_i'}
\]

**xvx=** series for variance of fitted values

This computes the series of (unscaled) variances for the in-sample fitted values. These are useful in various diagnostic tests. The formula is: \(x_i(X'X)^{-1}x_i'\)

Examples

```
smpl 1923:1 1941:1
linreg foodprod
# constant avgprice
prj fitted
scatter(style=symbols) 2
# avgprice foodprod
# avgprice fitted
data(unit=input)   1942:1  1945:1  avgprice
   112.3  112.8  113.9  119.3
prj forecast 1942:1 1945:1
```

The first `PRJ` computes fitted values over 1923:1 to 1941:1. The `SCATTER` instruction does an actual vs. fitted plot. The second `PRJ` forecasts `FOODPROD` over the period 1942:1 to 1945:1 using the four input values for `AVGPRICE`.

```
linreg employ 1947:1 1961:1 resids
# constant year price gnp armed
prj(xvx=px)
set stdresids = resids/sqrt(%seesq*(1-px))
graph(style=symbols,vlabel="Standardized Residual")
# stdresids
```

This uses `PRJ` with the `XVX` option to produce residuals standardized by their individual standard errors.

Distribution Statistics

You can use `PRJ` with the set of options described below to obtain one or more of the following statistics from a series \(z_i\) of (standardized) deviates:

- Density: \(\phi(z_i)\)
- Distribution: \(\Phi(z_i)\)
- Inverse Mills’ ratio: \(\phi(z_i)/\Phi(z_i)\)
- Derivative of the Inverse Mills’ ratio, evaluated at \(z_i\).
If observation $i$ is truncated at the value $T_i$, $z_i$ takes the following values:

$$ z_i = \begin{cases} 
\text{Bottom truncation} & \frac{(X_i\hat{\beta} - T_i)}{\sigma} \\
\text{Top truncation} & \frac{(T_i - X_i\hat{\beta})}{\sigma} 
\end{cases} $$

**Options (Distribution Statistics)**

- **distribution**= [probit]/logit/extreme
  - This selects the distribution to be used

- **density**= Series of densities
- **cdf**= Series of distributions
- **mills**= Series of Inverse Mills’ Ratios
- **dmills**= Series of Derivatives of MILLS

You can use any or all of these four options in a single **PRJ** instruction.

After a **DDV** estimation, the **CDF** option will generate the series of predicted probabilities of the “1” choice. Use the option **DISTRIB=LOGIT** if you want these to be calculated for the logit, as the default is to compute these for the normal (regardless of your choice on the **DDV**).

**Other Options**

- **smpl**= **SMPL** series or formula (User’s Guide, Section 5.2)
  - You can supply a series or a formula that can be evaluated across entry numbers. Entries for which the series or formula is zero or “false” will be skipped, while entries that are non-zero or “true” will be included in the operation.
  
  If the output series already exists, observations of that series not included in the **SMPL** will be completely unaffected by the **PRJ** operation.

- **scale**= the value of $\sigma$ [1]
- **upper**= SERIES or FRML of upper truncation points [unused]
- **lower**= SERIES or FRML of lower truncation points [unused]

  These describe the normalization procedure. The truncation points can differ among observations: for instance, cutoffs may depend on some demographic characteristics. However, you can only do either “top” truncation or “bottom” truncation in a given **PRJ** command—you cannot do both simultaneously. Thus **UPPER** and **LOWER** are mutually exclusive.

  Note that **UPPER** and **LOWER** replace the older **TRUNCATE** and **TOP**/ [NOTOP] options which provided the same functionality.

  Use missing value codes for any entries that are to be treated as unlimited.
atmean/[noatmean]
xvector=the value of \( \sigma \) [unused]

The ATMEAN and XVECTOR options allow you to compute the index, density and predicted probability for a single input set of X’s. The values are returned as the variables \%PRJFIT, \%PRJDENSITY and \%PRJCDF.

The ATMEAN option does the calculation at the mean of the regressors over the estimation range. With XVECTOR, you provide a vector at which you want the values calculated.

Some additional statistics can be obtained using the options DENSITY, MILLS and DMILLS.

Notes
The MILLS option is the only one necessary to compute the correction for truncation. You can use DMILLS with MCOV and matrix operations to compute the covariance matrix of these estimators.

Variables Defined by PRJ

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%MEANV</td>
<td>vector of means of the explanatory variables (Vector)</td>
</tr>
<tr>
<td>%PRJCDF</td>
<td>predicted probability produced by the XVECTOR or ATMEAN options (Real)</td>
</tr>
<tr>
<td>%PRJDENSITY</td>
<td>density produced by the XVECTOR or ATMEAN options (Real)</td>
</tr>
<tr>
<td>%PRJFIT</td>
<td>fitted value produced by the XVECTOR or ATMEAN options (Real)</td>
</tr>
</tbody>
</table>

Example
This is an example of a correction for zero truncation. To iterate on this, enclose everything after the first LINREG in a loop. This uses the standard error of the OLS regression for the estimate of \( \sigma \). It is possible to compute better estimates.

```r
smpl 1 100
linreg y
# constant x1 x2
compute sigma=sqrt(%seesq)
prj(scale=sigma,mills=corr)
set ycorr = y-sigma*corr
linreg ycorr
# constant x1 x2
```
PROCEDURE — User-Defined Procedures

The `PROCEDURE` instruction begins the definition of a RATS procedure. It specifies the name of the procedure and the parameter list, if any.

```
procedure procname parameters
```

**Parameters**

- `procname` The name you want to give this procedure. `procname` must be distinct from any other procedure or variable name in the program.

- `parameters` Names of the formal parameters. These names are `local` to the procedure—they will not conflict with variables elsewhere in the program. When you execute the procedure, RATS passes the values on the `EXECUTE` instruction to the formal parameters.

  By default, parameters are `INTEGER` passed by value. You can change this using `TYPE` statements.

**Description**

The most powerful compiler structure in RATS is the `procedure`. A procedure is similar to subroutines of FORTRAN and Basic and the functions of C. In effect, procedures allow you to define new instructions from a sequence of RATS commands.

A procedure begins with a `PROCEDURE` statement and ends with a matching `END`. The `PROCEDURE` statement itself names the procedure and lists the formal parameters. The usual arrangement of the statements in a procedure is

```
procedure statement
  type, declare, local, option and fixed statements, if any.
  other instructions
end
```

You should try to write the procedure using only parameters, options, local variables and global variables which RATS itself defines. That way you don’t have to worry about accidentally changing a global variable. If you have a global variable that you want to access outside the procedure, use a name starting with `%%` so it won’t conflict with either the user’s names or any names defined by RATS.

An alternative to a procedure is a `function`, which you can create with the `FUNCTION` instruction. Where procedures define new instructions, functions define operations similar to the functions used within RATS expressions.

Procedures are described in much greater detail in Chapter 16 of the *User’s Guide*. You should see also the descriptions of `TYPE`, `LOCAL`, and `OPTION`, used to set parameters types, define local variables, and define procedure options, respectively.
Examples (Partial)

```
procedure cumpdgm series start end
    type series series

    The procedure CUMPDGM has three parameters: SERIES is a type SERIES, START and END are INTEGER.

procedure distrib oldser newser
    type series oldser *newser
    option real rho .9
    option integer factor 3
    option choice model 1 rw1 ar1 rwar1 rw2

    DISTRIB takes two parameters: both are series, but OLDSER is input to the procedure and NEWSER is output by it. The * means that the NEWSER parameter is passed by address, and so its value can be set or changed by the procedure.
```

Using SOURCE

Once you have a procedure working the way you want, it is usually a good idea to save it as a separate file, so you can use it in different applications. A well-designed procedure can be used with a variety of data sets if specific information about the current data is passed to the procedure through parameters and options, rather than being hard-coded into the procedure.

If you have a procedure on a separate file, bring it into your current RATS program using the instruction SOURCE. The typical instruction is

```
source file with procedure
```

If you have a collection of procedures which you use regularly, you can create a procedure library that gets brought in right at the start of your program.

Running a Procedure

Procedures are executed using the EXECUTE command or (more commonly), using the @ sign (a shortcut for EXECUTE). For example:

```
@cumpdgm x 1980:1 2006:12
```

Order of Procedures

For complex tasks, it is very common for a procedure to execute other procedures. Because RATS needs to know the syntax of a procedure before it can even interpret the instruction which will execute it, it must process the sub-procedure before the main one. Thus you either need to place the sub-procedure first in your file, or "SOURCE" it in from a separate file before processing the main procedure.
**PRTDATA — Printing Data File Series**

**PRTDATA** prints to the screen (or exports to a file) series stored on the currently open RATS format data file. Use **PRINT** to display series stored in RATS memory.

```plaintext
prtdata( options )  list of data file series (optional)
```

**Wizard**

If you open the file with **Open RATSDATA** on the **File** menu, you can export series to another file using the **File–Export...** operation.

**Parameters**

- **list**
  - If you list a set of series names, **PRTDATA** will only print the data for those series. If you leave this parameter blank, RATS will print all the series on the RATS format file.

**Options**

- **format=[portable]/rats/xls/wks/dbf/prn/cdf/dif/html/ binary/free/"(FORTRAN format)"**
  - This selects the desired format for the output. The “spreadsheet” formats (XLS, WKS, DBF, PRN, CDF, DIF and HTML) will only work if the listed series have the same frequency. If you are making an archival copy, we would suggest you stick with **PORTABLE**, since it includes all the information present on the RATS format file itself.

- **organization=[rows]/cols (or [variable]/obs)**
  - For formats XLS, WKS, PRN, CDF, DIF and HTML, this selects whether the series will run across the page (or file) in rows (ORG=ROWS) or down the page in columns (ORG=COLS). See Section 1.1.2 in the **User’s Guide**.

- **unit=output/copy/other unit**
  - This sets the output unit. UNIT=OUTPUT is the default if you use FORMAT=PORTABLE. Otherwise PRTDATA defaults to UNIT=COPY. See the **OPEN** instruction for an explanation of I/O units.

- **picture=picture clause for data**
  - This allows you control the formatting of the numeric values in the output in PRN, CDF, DIF, HTML, and FREE formats. A picture clause takes a form like "##.###" or "*.#". The first requests two digits left of the decimal and three digits to the right. The second asks for one digit right and as many digits as are needed to the left. See **DISPLAY** for more on picture codes.
across=number of entries per line [4]
When using FORTRAN format, you can use ACROSS to indicate the number of
data values that will be output on each line. See the COPY command for details.

like=template string for series to export
LIKE allows you to export information only for series that match a template you
supply. You can use the standard wildcard characters “*” and “?”. For example,
LIKE="X*" will list all series whose name begins with X, while LIKE="X?" lists
only those series whose names are two letters long, with the first letter being X.

Description
You can use PRTDATA to:

- check the data on the file.
- make archival copies of the data in a “human-readable” format, since RATS
  format files are machine-readable only.
- transfer data to another program.

You can also export data from a RATS format file by reading the data into memory
with DATA and then writing it to a file with COPY.

Examples

  dedit ukdata.rat
  open copy archive.dat
  prtdata(unit=copy)

This opens the RATS format file UKDATA.RAT and a COPY file called ARCHIVE.DAT.
Then the PRTDATA instruction prints, in PORTABLE format, all series on the
UKDATA.RAT file to ARCHIVE.DAT.

  dedit sales.rat
  open copy sales.wks
  prtdata(format=wks,org=cols) salesx31 salesy45 salesy66

  this constructs the WKS format file SALES.WKS using the data from series SALESX31,
  SALESY45 and SALESY66.

See Also . . .

DEDIT Opens or creates RATS format data files.
OPEN Opens files
COPY Writes information from working data series to a file.
PStats

PSTATS — Analysis of Variance for Panel Data

**PSTATS** computes analysis of variance tests for time or individual effects and decomposes the variance of a series for random effects estimators. See Sections 14.5 and 14.6 of the *User’s Guide* for more information.

```
pstats( options ) series start end
```

**Parameters**

- `series` Series for which you want to compute statistics.
- `start` `end` Range of entries to use. If you have not set a **SMPL**, this defaults to the defined range of `series`.

**Description**

**PSTATS** uses the following decomposition of $u_{it}$ (the `series`):

$$ u_{it} = \varepsilon_i + \lambda_t + \eta_{it} $$

$\varepsilon$ is the individual effect, $\lambda$ is the time effect and $\eta$ the purely random effect. If you use the option **EFFECTS=INDIV**, the decomposition only includes the $\varepsilon$ and $\eta$ components. With **EFFECTS=TIME**, it only includes $\lambda$ and $\eta$. (Note: if a particular effect is weak, it is possible for the estimated variance of the component to be negative.)

**Options**

- **smpl=SMPL series**
  This is the standard **SMPL** option (Section 5.2 of the *User’s Guide*).
- **effects=[individual]/time/both**
  This option indicates whether to allow for **INDIVIDUAL** effects, **TIME** effects or **BOTH**.
- **tests/[notests]**
  **TESTS** requests the calculation of $F$-tests (analysis of variance) for the effects. For **EFFECTS=INDIVIDUAL** or **EFFECTS=TIME**, these are just the one-factor analysis of variance tests. For **EFFECTS=BOTH**, these are two-factor tests with one observation per cell.
- **spread/[nospread]**
  If you use **SPREAD**, **PSTATS** does a likelihood ratio test for equal variances across cross-sections.
Missing Values

RATS drops missing values from the computation.

Example

cal(panel=5)
all 40//5
open data panel.wks
data(format=wks,org=obs) / y z1 z2 x1 x2
linreg y / resids
# constant z1 z2 x1 x2
pstats(tests, effect=indiv) resids
pstats(spread) resids

The first PSTATS tests the residuals from an OLS regression for individual effects, the second tests for equal variances. See the output below.

Output

The output from PSTATS for individual and time effects is an Analysis of Variance table.

<table>
<thead>
<tr>
<th>Source</th>
<th>Sum of Squares</th>
<th>Degrees</th>
<th>Mean Square</th>
<th>F-Statistic</th>
<th>Signif Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDIV</td>
<td>1.5912296947623</td>
<td>39</td>
<td>.0408007614042</td>
<td>1.080</td>
<td>0.35971320</td>
</tr>
<tr>
<td>ERROR</td>
<td>6.0418415088589</td>
<td>160</td>
<td>.0377615094304</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>7.6330712036212</td>
<td>199</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Test of Equal Variances for Series RESIDS
Chi-Squared(39)=97.499868 with Significance Level 0.00000065

Notes

If you use EFFECTS=BOTH, the analysis of variance table will include $F$-tests for individual effects, time effects and a joint test. Note that the individual effects test will not be the same as you would get with EFFECTS=INDIV (and similarly for the time effects test with EFFECTS=TIME), as it is testing for individual effects allowing for time effects, while with EFFECTS=INDIV, it is not conditional on time effects.

Variables Defined by PSTATS

<table>
<thead>
<tr>
<th>%VRANDOM</th>
<th>variance of $\eta$: the random component</th>
</tr>
</thead>
<tbody>
<tr>
<td>%VINDIV</td>
<td>variance of $\epsilon$: the individual component</td>
</tr>
<tr>
<td>%VTIME</td>
<td>variance of $\mu$: the time component</td>
</tr>
</tbody>
</table>
QUERY — Requesting Input from the User

**QUERY** prompts a user to input values for variables. It is handy for writing general RATS applications that change slightly from one run to the next. **QUERY** displays a dialog box with the prompt and a text box for the reply. **DBOX** is a more powerful, but more complicated, instruction for getting user input, as it allows you to create custom dialog boxes.

\[
\text{query( options ) list of variables}
\]

**Parameters**

- **list**
  
  This can be any collection of **INTEGER**, **REAL**, **COMPLEX**, **STRING** or **LABEL** variables or array elements. You **may not** use an array itself. You must introduce any variable prior to using it in a **QUERY** instruction, with **DECLARE** for instance.

**Options**

- **prompt=Prompt String**
  
  This displays a prompt on the screen as a message to the user. It can either be a string enclosed in quotes ("), or a **STRING** variable. If you don’t use **PROMPT**, RATS displays a lone question mark.

- **status=INTEGER variable set to 0,1 status**
  
  If you use the option **STATUS**, **QUERY** does not give you an error if there is not enough data to fill all the variables. Instead, it sets your **STATUS** variable to 0. If the **QUERY** is successful, it sets the status variable to 1.

- **verify=expression returning non-zero value if input is valid [none]**
  
  **VERIFY** can check whether the user has provided a valid response. RATS will check that the information is in the correct form (for instance, it will catch the user inputting a string when a number is needed), but you need to use the **VERIFY** option if you want to test that a value is within the proper range. By using this, you can prevent the user from OK'ing an illegal value.

  The default error message is “Illegal Value”. If you use **VERIFY**, it’s a good idea to include an error message and to make it as informative as possible.

- **initialize/[noinitialize]**
  
  If **INITIALIZE**, the text box is filled with the current value of the variable. By default, the field is blank.
User Responses to a QUERY

In response to QUERY, the user can type either constants or expressions. For example, a user could type in a date field such as 1947:1. If QUERY requests more than one variable, the values can be separated by commas or blanks.

If you request a STRING, QUERY accepts the whole line as the value.

Examples

These show two uses of QUERY in to set up the early part of a RATS session.

```
declare integer ninput
query(prompt= "How many observations") ninput
allocate ninput

declare integer year  period  nfore
query(prompt= "Final Year and Period of Data") year period
query(prompt= "How Many Forecast Steps") nfore
allocate year:period+nfore
```

This is a bit fancier example. It

1. Initializes the input to 1
2. Checks that the input number is in the range 1 to %NREG. If not, it issues the message showing the valid range.

```
declare integer nrestr
compute nrestr=1
compute errmsg="Value must be >1 and <="+\%nreg
query(prompt= "How many restrictions?",initialize,$
   verify=(nrestr>=1.and.nrestr<=\%nreg),errmsg=errmsg) nrestr
```
QUIT — Aborting Data Editing

QUIT terminates editing of a RATS format data file without making any changes to the file. Use SAVE instead of QUIT when you want to save any changes to a file.

```
quit  (no parameters)
```

Because RATS does not make any changes to the file until you give an explicit SAVE instruction, you actually only need to use QUIT if you want to free up the memory space occupied by the file directory.

See Also . . .

DEDIT  Opens or creates a RATS format file.
SAVE  Saves changes to an open RATS format file.
STORE  Adds data series to an open RATS format file. You must do a SAVE to make the changes permanent.
END  Ends a RATS program.
**QZ Instruction — QZ Decomposition**

\[ \text{qz( options ) } A \ B \]

This computes the generalized Schur decomposition of the matrix pair \((A, B)\), which should be square matrices of the same dimensions. This generates a collection of matrices such that \( Q \Delta Z' = A \) and \( Q \Omega Z' = B \), where \( Q \) and \( Z \) are orthogonal matrices \((QQ' = ZZ' = I)\), \( \Omega \) is upper triangular and \( \Lambda \) is block upper triangular, where the blocks on the diagonal are \( 1 \times 1 \) for real generalized eigenvalues and \( 2 \times 2 \) for complex conjugate pairs.

Unlike the standard eigenvalue routines, it isn’t easy to sort generalized eigenvalues. Instead, if it’s necessary to control positioning, they are partitioned into two sets based upon some criterion. Assuming that the criterion never separates a pair of complex conjugate eigenvalues, the \( \Lambda \) and \( \Omega \) matrices will retain a block triangular structure, which is generally all that is needed for further work. The blocking is controlled by the combination of the `block` and `cutoff` options. The `size` option allows you to determine the size of the “upper” block.

**Options**

**block=below/above/real/imag**

Indicates the criterion used for determining which eigenvalues go into the upper block. **BELOW** and **ABOVE** are based upon absolute values, and use the **CUTOFF** value. **BLOCK=BELOW,CUTOFF=1.0** will put all eigenvalues with absolute value less than 1.0 in the upper block. **REAL** and **IMAG** partition the eigenvalues into real and complex, moving the indicated group into the upper block.

**cutoff=value or formula giving the cutoff value [1.0 by default]**

This supplies the cutoff value for **BLOCK=BELOW** or **BLOCK=ABOVE**.

The following are used to provide names for the variables computed by `QZ`:

- **q=Q matrix**
- **z=Z matrix**
- **lambda=\( \Lambda \) matrix**
- **omega=\( \Omega \) matrix**
- **cvalues=VECTOR[COMPLEX] of generalized eigenvalues**
- **evalues=VECTOR of (the real parts of) the generalized eigenvalues**
- **size=size of the upper block**

**Example**

\[ \text{qz(q=q, z=z, lambda=lambda, omega=omega, block=below, cutoff=1.0/beta) g0 g1} \]

does a QZ decomposition of the pair \( g0, g1 \) with blocking to put all generalized eigenvalues less than \( 1.0/beta \) in absolute value at the top.
RATIO — Tests with Multiple Equation Systems

**RATIO** computes a likelihood ratio statistic based upon the log determinants of two sets of series of residuals. Although you can use it in other situations, the primary purpose of **RATIO** is testing hypotheses in vector autoregressions: either block exogeneity restrictions or lag length restrictions. See *User's Guide* Section 10.4.

```
ratio(degrees=test degrees of freedom, other options) start end
  # first list of residual series
  # second list of residual series
```

**Parameters**

`start   end` Range over which the covariance matrices are computed. If you have not set a **SMPL**, this defaults to the maximum common range of all the series in the two sets.

**Supplementary Cards**

On the two supplementary cards, list the two sets of series that you use in the test. **RATIO** will compare the covariance matrices of these two sets of series. Both sets should have the same number of series. You don’t need to worry about which of the two is from restricted estimates and which is from the unrestricted because **RATIO** takes the absolute value of the computed statistic.

**Options**

**degrees=test degrees of freedom (Required)**

This is the degrees of freedom for the chi-squared statistic, that is, the number of restrictions. You must use this option.

**smpl=SMPL series or formula (User’s Guide, Section 5.2)**

You can supply a series or a formula that can be evaluated across entry numbers. Entries for which the series or formula is zero or “false” will be skipped, while entries that are non-zero or “true” will be included in the calculation.

**spread=series of residual variances (Section 5.4)**

The residuals which are generated by the regression commands are “unweighted.” Consequently, if you used the **SPREAD** option to compute residuals, you should use the same **SPREAD** series on **RATIO**.

```
[print]/noprint
  Use **NOPRINT** to suppress the printing of the test results.
```

**title="string for output title"**

Use the **TITLE** option to provide your own title for the output.
The multiplier correction is the $c$ in the formula below. Sims (1980, p.17) suggests a correction equal to the number of variables in each unrestricted equation in a vector autoregression. The use of the proper correction improves the small sample properties of the test.

**Description**

**RATIO** takes two lists of residual series, computes the two covariance matrices ($\Sigma_1$ and $\Sigma_2$) and generates the chi-squared statistic:

$$(T - c) \log |\Sigma_1| - \log |\Sigma_2|$$

where $T$ is the number of observations and $c$ is given by the **MCORR** option. Note that **RATIO** does not compute centered covariance matrices, that is, it does not subtract means from the input series.

The null hypothesis is that the two log determinants are equal. Small test statistics and significance levels close to 1.0 suggest the hypothesis can be accepted. Larger statistics and significance levels close to 0.0 suggest the hypothesis is rejected.

**Example**

```plaintext
ratio(degrees=27,mcorr=10)
# ures1 ures2 ures3
# rres1 rres2 rres3
tests the difference between the covariance matrix of series ures1, ures2 and ures3 and that of series rres1, rres2 and rres3. The test statistic is compared with a $\chi^2$ distribution with 27 degrees of freedom.
```

Output such as:

```
Log Determinants are -46.441108 -44.054300
Chi-Squared(100)= 90.698721 with Significance Level 0.73622035
```

would suggest that the null hypothesis can be accepted.

**Variables Defined by RATIO**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%CDSTAT</td>
<td>the computed test statistic (real)</td>
</tr>
<tr>
<td>%SIGNIF</td>
<td>the marginal significance level (real)</td>
</tr>
<tr>
<td>%NOBS</td>
<td>number of observations (integer)</td>
</tr>
<tr>
<td>%NVAR</td>
<td>number of variables (integer)</td>
</tr>
</tbody>
</table>

**See Also . . .**

- **VCV**
  - Computes a covariance matrix of a single set of series.
- **CDF**
  - Computes the significance level for a test statistic.
- **%DET (A)**
  - Function returns the determinant of a matrix.
READ — General Information Input

`READ` reads data into arrays and other variables. It supports free-format, Fortran-format, and binary-format data. By contrast, `INPUT` is strictly for free-format. `READ` will not work with data series—use the `DATA` instruction to read data into series.

```
read( options )   arrays, variables, array elements
```

Parameters

`arrays,...` These are the objects for which data is to be read. You can use any combination of variables. Any arrays must be dimensioned ahead of time (unless you use the option `VARYING`).

Options

`unit=input/[data]/other unit`

`READ` reads the data from the specified I/O unit. By default, this is the `DATA` unit.

`format=[free]/binary/"( FORTRAN format )"

This tells RATS how to read in the data. See below for information on how the format option affects the way that RATS fills arrays.

`varying/[novarying]`

`status=INTEGER variable set to 0,1 status`

`singleline/[nosingleline]`

These are more advanced options. See their description later in this section.

Description

The `READ` instruction reads information into the arrays and variables in the order listed. The manner in which arrays are read depends upon the `FORMAT` option.

- With `FORMAT=FREE`, `READ` is identical to `INPUT` except for the different default setting for the `UNIT` and `SINGLELINE` options. It reads RECTANGULAR arrays by rows and SYMMETRIC arrays by the rows of the lower triangle. It can read more than one row from a single line.
- With a FORTRAN format (User’s Guide, Section 2.9), `READ` requires that each array and each row of an array begin on a new line. The format should be the format required to read a single row, not the whole array. As with `FORMAT=FREE`, it reads RECTANGULAR arrays by rows and SYMMETRIC arrays by the rows of the lower triangle. A VECTOR is treated like a single row.
- With `FORMAT=BINARY`, `READ` processes arrays in their internal order, RECTANGULAR arrays are read by columns, SYMMETRIC by the rows of the lower triangle.
Notes

You should use FORMAT=BINARY only to read data written out of RATS using WRITE FORMAT=BINARY. If the binary data are generated in some other fashion, it is possible that the byte streams won’t match and you will end up with gibberish.

Examples

```plaintext
declare symmetric d(3,3)
read(unit=input,format="(3f6.2)") d
  10.00
  -5.20  12.20
  1.30   5.40  14.60
read(unit=input,format=free) d
  10.0  -5.2  12.2  1.3  5.4  14.6
```

The two READ instructions put the same set of numbers into the SYMMETRIC array D. The first is an example of a formatted read: each row of data for the array appears on a separate line. The second creates the same matrix using the free format option.

```plaintext
declare rectangular a(5,5)
open data arrayin.dat
read(format=free) a
```

This reads the first twenty five numbers from the file ARRAYIN.DAT into the RECTANGULAR array A, row by row.

Advanced Options

- **varying/[novarying]**
  - **status=INTEGER variable set to 0,1 status**
  - **singleline/[nosingleline]**

  The VARYING and STATUS options allow you to work with lists whose size you do not want to set in advance.

You can use VARYING to input data for a single VECTOR array of any numeric or character type. With VARYING, the VECTOR is filled with as much data as is available. By default, this is the entire contents of the data file. With SINGLELINE, it will read data only from a single line. (SINGLELINE will be ignored if you use FORMAT=BINARY).

If you use the option STATUS, READ does not give you an error if there is not enough data to fill all the variables. Instead it sets your STATUS variable to 0. If the READ is successful, it sets the status variable to 1.
Examples

dec vector[label] tickers
open data tickers.lst
read(varying) tickers
do  i=1,%rows(tickers)
    ....
end do i

This reads labels from the file TICKERS.LST, then loops over the number of elements processed.

See Also . . .

INPUT  An alternative to READ, capable of reading only free-format data.
It is designed for reading data from the input unit.

ENTER  Reads data for arrays and variables from supplementary cards.

MEDIT  Obtains data for an array from the user through a spreadsheet window.

WRITE  Writes arrays and variables to output or to an external file.
RELEASE — Releasing Segments of Memory

You can use RELEASE to free up the space in memory which previously has been set aside for the listed arrays and series. By using the options, you can release several other large memory blocks.

```
release(options) list of arrays or series
```

Parameters

`list...` The arrays or series whose memory blocks you want to release.

Options

The options allow the release of space set aside for certain other purposes. Each of these is a switch option, which is off by default:

- **FREQUENCY** releases the frequency domain series block set up by a FREQUENCY instruction.
- **REGRESS** releases the block of regression information: %XX, %BETA and other things.
- **CMOMENT** releases the array %CMOM and other information set up by CMOMENT.

Notes

You may find it necessary to use this instruction if you have severe constraints on available memory, or if you are running a large program. However, it makes little sense to release a series or array unless you are truly finished with it. If you set it or dimension it later, you simply have borrowed the space temporarily.

RELEASE does not actually remove the arrays and series themselves from the table of variables, so you can re-dimension them later without using another DECLARE.

If you use LOCAL arrays or series in a PROCEDURE, note that RATS does not automatically release the space allocated to them when it finishes executing the procedure. If you want to free up the space, you will have to do an explicit RELEASE instruction.
**Rename**

**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  comments
< text lines >  comment lines if comments = 1 or 2
```

**Wizard**

If you open the file using `Open RATSData` on the `File` menu, you can rename a series using the `Rename` toolbar icon:

**Parameters**

- `old name` Current name of the series.
- `new name` New name for the series.

The `comments` parameter used in older versions has been replaced by the `COMMENTS` option described below. Version 7 will still recognize this older parameter, but you should use the option for new programs.

**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**

```plaintext
dedit mydata.rat
rename fzml 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`.

```plaintext
dedit prices.rat
open data prices.xls
store(convert=xls,org=cols)
rename 1234 a1234
save
```

converts an XLS file to RATS format, then fixes an illegal name (RATS variable names can’t start with a digit).
**REPORT — Report Generation**

**REPORT** is a flexible report generator, which allows you to insert information into a table, format it and display it, either as text, or as a spreadsheet-style window which can be copied into other applications or exported in a variety of formats. You use a set of at least three **REPORT** instructions to create the table: **REPORT** with **ACTION=DEFINE** initiates the report, **ACTION=MODIFY** adds information (you'll usually have more than one of these) and **ACTION=SHOW** displays it. **ACTION FORMAT** can be used to format the numbers and add other formatting information.

```
report (action=define, other options)
report (action=modify, regressors, other options)
report (action=modify, other options) variables/expressions
report (action=format, other options)
report (action=sort, bycol=column to sort on)
report (action=show, other options)
```

**Parameters**

*variables/expressions*  
With **ACTION=MODIFY**, this list of variables and expressions provides the information which is to be inserted into the report.

**Options**

*action=define/[modify]/format/show/sort*  
**REPORT** with **ACTION=DEFINE** option initiates the report. **ACTION=MODIFY** (which is the default, and so can be omitted) adds content to the report. **ACTION=FORMAT**, in conjunction with **PICTURE** and **WIDTH**, allows you to adjust the formatting. **ACTION=SORT** sorts the report on the values of one the columns. Finally, use **REPORT** with **ACTION=SHOW** to display the report.

The **ACTION** option is used on every **REPORT** instruction. The remaining options are described below, grouped by the **ACTION** choice with which you use them.

**Options Used With ACTION=DEFINE**

*hlables=VECTOR of STRINGS to provide the column (header) labels*
*vlables=VECTOR of STRINGS to provide row labels*

These provide labels for the columns and rows in the displayed report. Note: the **HLABELS** are outside the main body of the report, while the **VLABELS** are considered to be in column one.
Options Used With ACTION=MODIFY

regress/[noregress]
extra=[stderrs]/tstats/both/neither
arrange=[name]/position

The REGRESS option adds the results of the most recent regression to the report. The coefficients are always included and, by default, the standard errors are reported as well. You can use the EXTRA option to select $t$-statistics instead of, or in addition to, the standard errors, or to include neither.

When adding results from multiple regressions to the same report, by default, the regressors will be arranged by name—if a regressor in the current regression also appeared in one of the previous regressions, the results for that regressor will be placed in the same row (in a new column) as the regressor with the matching name in the earlier regressor. If you’d rather have the table arranged so that the regressors in the same position within the regression are to be considered “parallel”, use the option ARRANGE=POSITION. If you do this, you should also use the ATROW option to fix the row for the first regressor.

row=[input]/new/current/find
col=[input]/new/current
string= search string

These determine the position at which new data is to be inserted. The defaults are ROW=INPUT and COL=INPUT. With those, the ATROW and ATCOL options give the position. NEW means that a new row or column is created. CURRENT means that the last row or column used will be used again. ROW=FIND, combined with the STRING option, looks for a match in the first column. If a match isn’t found, a new row is opened up.

atrow=row number or starting row number [row 1]
atcol=column number or starting column number [column 1]

ATROW and ATCOL indicate an exact location to begin inserting the new data.

fillby=[rows]/columns

With the default choice FILLBY=ROWS, VECTORS and lists of scalars are added to the report going across in a row. Use FILLBY=COLUMNS if you instead want the data to be inserted running down a column. This option has no effect on data in RECTANGULAR or SYMMETRIC arrays, which are added across both rows and columns exactly as the data are stored in the array.

align=[left]/center/right/decimal

Sets the alignment of a string or label. RIGHT and DECIMAL are the only ones you can use for numbers.
special=[none]/onestar/twostars/parens/brackets
You can use SPECIAL to enclose cells in parentheses (), brackets [] or tag them with one star (*) or two stars (**).

span/[nospan]
tocol=span up to and including this column [unused]
Use SPAN if you want the cell being added to span multiple columns. Include TOCOL if you want the cell to span across to a specific column. This is normally used when inserting a text string that serves as a description for the next set of rows.

With NOSPAN, the width of the column being used will be expanded as needed to fit the supplied information.

Options Used With ACTION=FORMAT
picture=picture code for formatting data
width=maximum width for numerical display
These are used with ACTION=FORMAT. If you provide a picture code (see DISPLAY for a description of these), you set the format yourself. If you use width, you set the maximum width in character positions, and REPORT determines a common format, within that width, which will display all the data covered by the request.

tag=maximum/minimum
special=[none]/onestar/twostars/parens/brackets
TAG identifies the maximum or minimum value in the range specified by the ATROW, TOROW, ATCOL, and TOCOL options. The SPECIAL option determines how the cell will be identified: enclosed in parentheses or brackets, or tagged with one star (*) or two stars (**). For example, the following would “star” the maximum value in the first column:

    report(act=format,atcol=1,tocol=1,tag=max,special=onestar)

atrow=starting row number [1]
atcol=starting column number [1]
torow=ending row number [last row]
tocol=ending column number [last column]
These set the range over which the PICTURE and WIDTH formatting will be applied, or the range over which TAG will search to identify a maximum or minimum. The range includes all rows from starting row number to ending row number, and all columns from starting column number to ending column number.

align=[left]/center/right/decimal
Sets the alignment of a string or label. RIGHT and DECIMAL are the only ones you can use for numbers.
Options Used With ACTION=SORT

`bycol=column number`
This sorts the report by the values in the selected column.

Options Used With ACTION=SHOW

`window="title for window"`
By default, the output will be displayed as text in the output window. Use the `WINDOW` option if you instead want to display the report in a new spreadsheet style “report window”, with the title you specify. From a report window, you can use *File–Export* to export the results to an external file in a variety of formats.

`unit=copy/[output]/other unit`
`format=html/prn/wks/cdf/dif/dbf/xls [formatted text]`
You can use the `UNIT` option to redirect the report output to another unit (usually an external file previously opened with an `OPEN` instruction). If you use `UNIT` without using `FORMAT`, RATS will generate a text file, with the output formatted just as it is when displayed in the output window. If you prefer, you can use the `FORMAT` option to select one of the other file formats, as shown above (`FORMAT` only applies when used in conjunction with `UNIT`).

Examples

This performs example 12.4 from Greene (2003):

```plaintext
calendar(q) 1950:1
all 200004
open data greenetablef511.rat
data(format=rats) 1950:1 2000:4
set logm1 = log(m1)
set loggdp = log(realgdp)
set logcpi = log(cpi_u)

linreg logm1 / resids
# constant loggdp logcpi
compute olsstd=%stderrs
linreg(robust, lags=5, lwindow=newey) logm1
# constant loggdp logcpi
```

*Use REPORT to build Table 12.1. We saved the standard errors from OLS in the vector OLSSTD.*

```plaintext
report(action=define,hlabels="|Variable","OLS Estimate","OLS SE","Corrected SE"|)
report(atrow=1,atcol=1) "Constant" %beta(1) olsstd(1) %stderrs(1)
report(atrow=2,atcol=1) "ln Output" %beta(2) olsstd(2) %stderrs(2)
report(atrow=3,atcol=1) "ln CPI" %beta(3) olsstd(3) %stderrs(3)
report(action=format,picture="*.####")
report(action=show,window="Table 12.1 Robust Covariance Matrix")
```
The program below implements the example from pages 140-142 of Diebold (2004). It uses the SPECIAL option to flag the significant correlations. The resulting Report Window is shown below, as it appears in the Windows version of RATS.

```
source bjident.src

cal(q) 1961
all 1994:4
open data caemp.dat
data(format=prn,org=columns)
@bjident(number=12) caemp
diff caemp / dcaemp
corr(corrs=c1diff,stderrs=std1diff,number=30) dcaemp
report(action=define,hlabels=||"Lag","AutoCor"||)
do i=1,30
    report(row=new,atcol=1) i
    if abs(c1diff(i+1)/std1diff(i+1))>2.0
        report(row=curr,atcol=2,align=decim,special=onestar) c1diff(i+1)
    else
        report(row=current,atcol=2,align=decimal) c1diff(i+1)
end do i
report(action=format,width=6)
report(action=show,window="Correlations of CAEMP, 1st Difference")
```

![Correlations of CAEMP, 1st Difference](image)
The code below is taken from the `VARLAG.PRG` example program. This searches for the minimum values of the AIC and SBC criterion. Note the use of the `TAG` option to identify the minimum criterion values in columns 2 and 3 of the report:

```plaintext
report(action=define, hlabel=||"Lags","AIC","SBC","LR","P-Value"||)
dec real lastll
do lags=1,8
    system(model=usamodel)
    variables gdpq unemprate gpdi gdpdefl m2
    lags 1 to lags
det constant
end(system)
estimate(noprint) 1962:1 *
compute ll = %logl
compute sbc = -2.0*ll + %nregsystem*log(%nobs)/%nobs
compute aic = -2.0*ll + %nregsystem*2.0/%nobs
report(row=new, atcol=1, align=decimal) lags aic sbc
if lags>1
    report(row=current, atcol=4) $
    2*(ll-lastll) %chisqr(2*(ll- lastll),%nvar*%nvar)
    compute lastll=ll
end do lags
report(action=format, atcol=2, tocol=2, special=onestar, tag=min, align=decimal)
report(action=format, atcol=3, tocol=3, special=onestar, tag=min, align=decimal)
report(action=format, atcol=2, tocol=3, width=8)
report(action=format, atcol=4, tocol=5, picture="*.######")
report(action=show)
```
RESTRICT — Testing or Imposing General Linear Restrictions

RESTRICT has two functions: testing general linear restrictions, and (with the option CREATE) doing regressions subject to linear restrictions. After a discussion of the elements of the instruction common to both uses, there are separate subsections for each. MRESTRICT is a similar instruction which uses matrices rather than supplementary cards to specify the restrictions.

```
restrict(options) restrictions residuals coefs
# list of coefficients entering the restriction
# coefficient weights followed by restricted value
```

Wizard

Use Regression Tests on the Statistics menu, and select General Linear Restrictions.

Parameters

- **restrictions**: The number of linear restrictions.
- **residuals**: (Optional) With CREATE option only, this is a series for the residuals from the restricted regression. Use * if you want to use coefs and not residuals.
- **coeffs**: (Optional) With CREATE, the series for the restricted coefficients.

Supplementary Cards

Represent each restriction by a pair of supplementary cards. On the first, list the numbers of the coefficients which enter the restriction. On the second supplementary card in each pair, list the weights attached to the coefficients listed on the first card, followed by the value which this linear combination of coefficients takes.

Note that you list the coefficients entering the restriction by coefficient numbers rather than by variable names. RATS puts the coefficients for a LINREG or similar instruction in the regression in the order listed on the supplementary card, with a block of lags ordered from the lowest lag (highest lead) to the highest lag.

Examples of Supplementary Cards

\[ \beta_5 = 4.0 \quad (\text{or } 1.0 \times \beta_5 = 4.0) \]

```
# 5
# 1.0 4.0
```

\[ \beta_1 - 2\beta_2 + \beta_3 = 0.0 \]

```
# 1 2 3
# 1.0 -2.0 1.0 0.0
```
Hypothesis Tests

RESTRICT, when used without the option CREATE, operates like the other hypothesis testing instructions (EXCLUDE, TEST). It does not make any changes to the stored information about the last regression, so any additional hypothesis testing instructions will apply to the original regression. RATS ignores the residuals and coeffs parameters on the instruction line.

The main test statistic is usually shown as an $F$, but will be shown as a chi-squared when RESTRICT is applied to estimates from LDV or DDV or from any instruction for which the ROBUSTERRORS option was used during estimation.

For $F$ tests with one degree of freedom, RESTRICT will report a two-tailed $t$ test in addition to the $F$ test. For chi-squared tests with more than one degree of freedom, RESTRICT will report an $F$ with an infinite number of denominator degrees of freedom (that is, the chi-squared statistic divided by the numerator degrees of freedom) in addition to the chi-square.

You can also control the distribution yourself using the FORM option.

Options for Testing

You can use the following options when you use RESTRICT to do hypothesis testing:

**print/[noprint]**
By default, RESTRICT produces only the standard errors and $t$-statistics. If you use the PRINT option, RATS prints the restricted coefficient vector in a table with the label, lag and coefficient, but without standard errors and $t$-statistics. If you use NOPRINT explicitly, RATS suppresses all output from RESTRICT.

**form=f/chisquared**
This determines the form of the test statistic used. By default, RATS will select the appropriate form based upon the estimation technique used last. You can use FORM to manually select a distribution if you have made changes to the regression that require a different distribution, such as altering the %XX matrix in a way which incorporates the residual variance into %XX. See Section 5.13 in the User’s Guide.

**title="string for output title"**
You can use the TITLE option to include information in the output to identify what is being tested.

Restricted Regressions

There are two ways to estimate linear models subject to restrictions. One is to “code” the restrictions into the explanatory variables. RATS has an instruction ENCODE for implementing that strategy. The other way, used by RESTRICT, is to estimate the unrestricted model and then impose the restriction.
Use the option `CREATE` when you want to use `RESTRIC` to compute the restricted regression. `CREATE` does the following:

- It computes the new coefficient vector and covariance matrix of coefficients subject to the restrictions. See Section 6.2 of the User’s Guide for the formulas used in the calculation.
- It computes new summary statistics. It recomputes all the `LINREG` variables such as `%RSS` and `%NDF`.
- It replaces the old regression with the new restricted regression. *Any further hypothesis tests will apply to the restricted regression, not the original one.*

It does all of this *in addition* to the regular task of computing the test statistic for the restriction. You can save the residuals or coefficients from the restricted regression using the `residuals` and `coeffs` parameters on the instruction line.

### Options for Restricted Regression

The following options are available when you do restricted regressions with `RESTRIC`.

`create/ [nocreate]`
CREATE does the restricted regression.

`[print]/noprint`
`vcv/ [novcv]`
When you use these with `CREATE`, they perform the same task as they do for `LINREG`: controlling the printing of the regression results and covariance-correlation matrix of coefficients, respectively.

`unravel/ [nounravel]`
UNRAVEL causes a substitution for `ENCODEd` variables. This is one of the two steps in the other method of restricted regression.

`define= equation to define`
`frml=FRML to define`
Respectively, these define an equation and a formula from the results.

`form=f/chi squared`
This is described earlier. It affects only the test statistic, not the restricted regression.

`replace/ [noreplace]`
REPLACE is an alternative to `CREATE`. It computes the restricted coefficients and covariance matrix, *but does not print output or compute residuals as the CREATE option does.* This is most useful when doing restrictions on systems of equations (SUR) since `RESTRIC(CREATE)` can only print single equations.
Restrict

coeff=VECTOR for restricted coefficients
covmat=SYMMETRIC for restricted covariance matrix

You can use these options to save the restricted coefficient and/or covariance arrays. The REPLACE option is equivalent to the pair of options COEFF=%BETA and COVMAT=%XX.

Examples

Suppose you have the following equation:

\[ y_i = \beta_0 + \beta_1 x_{1t} + \beta_2 x_{2t} + \beta_3 x_{3t} + u_t \]

which can be estimated using:

```
linreg y
# constant x1 x2 x3
```

We want to test the simple hypothesis that \( \beta_1 = \beta_2 \). RESTRICT operates by testing whether a linear combination of coefficients is equal to a specific value, so we need to rewrite this hypothesis in this form: \( \beta_1 - \beta_2 = 0 \)

We are testing the second and third coefficients here, so the RESTRICT would be:

```
restrict 1
# 2 3
# 1.0 -1.0 0.0
```

If you want to compute the restricted regression, do:

```
restrict(create) 1
# 2 3
# 1.0 -1.0 0.0
```

The following estimates the translog cost function

\[
\log(C/Q) = \alpha + \sum_i \alpha_i \log w_i + \sum_j \sum_{j \geq i} \gamma_{ij} (\log w_i)(\log w_j)
\]

and tests (jointly) the following combination of restrictions:

\[
\begin{align*}
\alpha_1 + \alpha_2 &= 1.0 \\
\gamma_{11} + 0.5\gamma_{12} &= 0.0 \\
\gamma_{22} + 0.5\gamma_{12} &= 0.0 \\
\end{align*}
\]

```
set logulc = log(ulc)
set logw1 = log(w1)
set logw2 = log(w2)
set lw1w1 = logw1**2
set lw2w2 = logw2**2
set lw1w2 = logw1*logw2
```
linreg logulc
# constant logw1 logw2 lw1w1 lw1w2 lw2w2
restrict 3
#  2   3
# 1.0 1.0 1.0
#  4   5
# 1.0 0.5 0.0
#  5   6
# 0.5 1.0 0.0

The following estimates a piecewise linear regression of C on Y, where the knot is at \( T=29 \). The restriction forces the two segments to meet at the knot.

```
set d2  = (t>=29)
set dy2 = y*(t>=29)
linreg c
# constant y d2 dy2
restrict(create) 1
#  3    4
# 1.0  y(29) 0.0
```

The code below estimates a four lag autoregression on LGNP and forces the lag coefficients (numbers 2 through 5) to sum to one.

```
linreg lgnp
# constant lgnp{1 to 4}
restrict(create) 1
#  2 3 4 5
# 1 1 1 1 1
```

**Variables Defined by RESTRICT**

- %CDSTAT: the computed test statistic (real).
- %SIGNIF: the marginal significance level (real).

If you use CREATE, all the variables defined by LINREG will be defined as well.

**See Also . . .**

- **MRESTRICT** is similar to RESTRICT, but uses matrices rather than supplementary cards to specify the restrictions. **TEST** is more specialized than RESTRICT—it tests specific values for coefficients, and cannot test any linear combinations including more than one coefficient. **EXCLUDE** is even more specialized—it tests exclusion restrictions only.

See Section 5.11 of the User’s Guide for more on restricted regressions.
RETURN — Returning from a PROCEDURE or FUNCTION

`RETURN` returns control from a `PROCEDURE` or `FUNCTION`. Execution continues with the instruction following the `EXECUTE` that invoked the procedure.

RATS automatically puts a return at the end of a procedure, so you only need `RETURN` if you want to exit before hitting the end of the procedure.

Before you return from a `FUNCTION`, you should set the return value by assigning it to the function’s name.

```
return (no parameters)
```

**Example**

```
procedure regdiags resids
  type series resids
  *
  option arch integer 0
  option qstat integer 0
  *
  local series ressqr
  *
  if arch<0
    {
      display "REGDIAGS: ARCH option must be >=0"
      return
    }
  if arch>0
    ...

uses RETURN to abort the procedure if the user specifies an improper value for an option.
```

**See Also . . .**

- *UG*, Section 16.2 Procedures.
- `END` Signals the end of a procedure or loop.
- `HALT` Terminates RATS from within a procedure.
REWIND — Rewinding a RATS I/O Unit

REWIND positions a file (a RATS I/O unit) so it can be processed from the beginning again. Rewind is necessary only if you have to run through a single ASCII or binary data file more than one time in a single program. This is a rare situation.

```
rewind    RATS I/O unit
```

Parameters

- **RATS I/O unit**  The I/O unit to rewind.

See Also . . .

- **OPEN**  Opens a RATS I/O unit.
- **CLOSE**  Closes a RATS I/O unit. If you want to read from a file to which you have previously written, you must do a CLOSE followed by a new OPEN.
- **CNTRL**  With the RESET option, it restarts a SOURCE file.
RLS — Recursive Least Squares

RLS uses the Kalman filter to generate a set of least squares regressions over a set of entries, generating various series of statistics on the behavior of these regressions. These are often used in formal or informal tests of the stability of a regression relationship.

If you need some information beyond the recursively generated coefficients and residuals, such as forecasts from each stage, you need to use the KALMAN instruction instead. KALMAN will do the same type of calculation, but does so one entry at a time, allowing you to do whatever extra computations are required at each stage.

```
rls( options )  depvar  start  end  residuals
# explanatory variables in regression format
```

Parameters

- **depvar**: Dependent variable.
- **start**  **end**: Range to use in estimation. If you have not set a SMPL, this defaults to the largest common range for all the variables involved.
- **residuals**: (Optional) Series for the recursive residuals.

Options

- `[print]/noprint`
- `vcv/[novcv]`
- `title="title for output"` [“Recursive Least Squares”]

These control the printing of regression output and the printing of the estimated covariance/correlation matrix of the coefficients (User’s Guide, Section 5.1), and the title used in labeling the output.

- `smpl=SMPL  series or formula` (User’s Guide, Section 5.2)

You can supply a series or a formula that can be evaluated across entry numbers. Entries for which the series or formula is zero or “false” will be skipped, while entries that are non-zero or “true” will be included in the operation.

- `spread=Residual  variance  series` (Section 5.4)

Use SPREAD for weighted recursive least squares. The residual variances are assumed to be proportional to the indicated series.
**equation=**equation to estimate

**lastreg/[nolastreg]**

Use the EQUATION option to estimate a previously defined equation. If you use it, omit the supplementary card. LASTREG re-estimates the most recent regression using recursive least squares.

**cohistory=VECTOR[SERIES] of coefficient estimates [not used]**

Respectively, these store the sequential estimates of the coefficients and the sequential estimates of the standard errors of the coefficients. These are stored into VECTORS of SERIES, with each element of the vector being a series for a different coefficient. For instance, if you do:

```
rls(cohistory=coefs) y
# constant x1
```

the series COEFS(1) will contain the sequential coefficient estimates for the constant term, while COEFS(2) will contain the coefficient estimates for X1.

**sighistory=SERIES for the standard errors [not used]**

**dfhistory=SERIES for the degrees of freedom history [not used]**

SIGHISTORY saves the sequential estimates of the standard error of the regression into a series. DFHISTORY saves the degrees of freedom at each time period (the excess of observations over the number of parameters).

**csums=SERIES for cumulated sum of recursive residuals [not used]**

**csquared=SERIES for cumulated sum of squared residuals [not used]**

When scaled, these can be used for the CUSUM and CUSUMSQ tests for stability and homoscedasticity.

**order=**series or formula giving order entries are to be added

**index=**SERIES[INTEGER] showing the entry mapping actually used

You can use ORDER to add entries to the regression based upon a series other than the time sequence. The “history” series and residuals keep the original entry mapping. If you need to “remap” these into the sequence in which they were added to the regression, you can use the INDEX option to get that entry mapping. That is, if you do ORDER=POP,INDEX=IPOP and save the residuals into RES, the series RES will be the recursive residuals in their original order. So, for instance, a scatter plot of POP against RES will be sensible; while the series generated by RES(IPOP) will be the series of residuals in population order.

**condition=**# of initial periods for first regression

This sets the number of initial observations used in the first regression. It defaults to the number of regressors—the value you supply must be greater than or equal to that. Note that if you condition on greater than the number of regressors, the residuals for the conditioning period will not be mutually independent.
Technical Information

If there are \( K \) regressors, \texttt{RLS} will first find the smallest set of entries in the sample, added in the order indicated, which will give a full rank regression (unless you use the \texttt{CONDITION} option, in which case \textsc{rats} uses the number of entries you specify). This will give a coefficient estimate \( \beta_t \) and \( XX^{-1} \) matrix \( \Sigma_t \). The residual, \texttt{SIGHIST} and \texttt{SEHIST} entries for these early entries will be zeros. Call the starting entry \( T_0 \) and the point where we get to full rank \( T_1 \). Given a previous set of entries, the result of adding a new data point is

\[
(1) \quad \hat{e}_t = \frac{(y_t - X_t \beta_{t-1})}{\sqrt{1 + X_t \Sigma_{t-1} X_t'}}
\]

\[
(2) \quad \beta_t = \beta_{t-1} + \sum_{t-1} X_t' \frac{(y_t - X_t \beta_{t-1})}{1 + X_t \Sigma_{t-1} X_t'}
\]

\[
(3) \quad \Sigma_t = \Sigma_{t-1} - \frac{\sum_{t-1} X_t' X_t \Sigma_{t-1}}{(1 + X_t \Sigma_{t-1} X_t')}
\]

where \( \hat{e}_t \) is the recursive residual at \( t \). The estimated variance of the regression through \( t \) is

\[
(4) \quad \sigma^2_t = \sum_{s=t}^{t} \hat{e}_s^2 / (t - T_1)
\]

and the standard errors of the coefficient estimates are square roots of the diagonal elements of \( \sigma^2_t \Sigma_t \).

Examples

The following excerpts are taken from the example on pages 121-126 of Johnston and DiNardo (1997). The complete program is provided on the file \texttt{JOHNP121.PRG}.

The \texttt{COHIST} and \texttt{SEHIST} options provide a \texttt{VECTOR[SERIES]} with the “histories” of the coefficients and the standard errors of the coefficient estimates, while \texttt{SIGHIST} gives the standard errors of the regression. The \texttt{CSQUARED} option returns the sum of squared recursive residuals, which will also be the sequence of sums of squared residuals from the regressions.

\begin{verbatim}
rls(sehist=sehist,cohist=cohist,sighist=sighist, $ 
    dfhistory=dfhist,csquared=cusumsq) y 1959:1 1973:3 rresids 
# constant x2 x3
*
set lower = -2*sighist
set upper = 2*sighist
graph(header="Recursive Resids and S.E. Bands for Gasoline") 3 
# rresids
# lower
# upper / 2
\end{verbatim}
Do the sequential F-test graph shown in figure 4.4. Since the cusumsq series has the RSS’s, the F-tests themselves are fairly easy. The second stage takes the F-tests and converts them into a ratio to the critical value, which changes with t, since the denominator degrees of freedom changes. \%invftest is used for that.

```
set seqf = (t-%nreg-%regstart())*(cusumsq-cusumsq{1})/cusumsq{1}
set seqfcval %regstart()+%nreg+1 * = $
   seqf/%invftest(.05,1,dfhist(t))

graph(header=$
"Figure 4.4 Sequential F-Tests as Ratio to .05 Critical Value",$
   vgrid=||1.0||)
```

Do the RESET test using the “RESET” procedure

```
@reset(h=2) y 1959:1 1973:3
# constant x2 x3
```

Variables Defined by RLS

RLS defines the variables shown below, which includes most of the same variables as LINREG. For RLS, these will all show the end of sample values. The coefficients, covariance matrix, standard errors, etc. should all match. The Durbin-Watson won’t match, because it’s computed using the recursive rather than non-recursive residuals. Note that it doesn’t have the distribution that a DW would have when applied to regular regression residuals, though it will still be near 2 if the residuals aren’t serially correlated.

- %BETA: Coefficient vector (VECTOR)
- %XX: X'X^{-1} matrix (SYMMETRIC)
- %TSTATS: Vector containing the t-stats for the coefficients (VECTOR)
- %STDERRS: Vector of coefficient standard errors (VECTOR)
- %MEAN: Mean of dependent variable (real)
- %NDF: Degrees of freedom (integer)
- %NOBS: Number of observations (integer)
- %NREG: Number of regressors (integer)
- %RSQUARED: Centered R^2 (real)
- %RBARSQ: Adjusted R^2 (real)
- %TRSQ: No. of observations times raw R^2 (real)
- %DURBIN: Durbin-Watson statistic (real)
- %RSS: Residual sum of squares (real)
- %SEESQ: Standard error of estimate squared (real)
- %QSTAT: Q-statistic (real)
- %QSIGNIF: Significance level of Q-statistic (real)
- %RHO: First lag correlation coefficient (real)
- %VARIANCE: Variance of dependent variable (real)
RReg

RREG — Robust Regression

RREG does several types of “robust” estimation procedures for a linear regression. One of these is the LAD or MAD (Least or Minimum Absolute Deviations), the other the generalization of LAD known as quantile regression (Koenker and Bassett (1978)).

RREG can be applied to linear models only. The general design is similar to LINREG.

```
rreg( options )   depvar   start   end   residuals
# explanatory variables in regression format
```

Wizard

To do robust regression via a menu-driven wizard, use the Regressions wizard on the Statistics menu and select Robust Regression as the technique.

Parameters

- **depvar**: Dependent variable.
- **start end**: Range to use in estimation. If you have not set a SMPL, this defaults to the largest common range for all the variables involved.
- **residuals**: (Optional) Series for the residuals.

A fifth parameter, 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.

Options

- [print]/noprint
- vcv/[novcv]
- title="title for output"  [depends upon options]

These control the printing of regression output and the printing of the estimated covariance/correlation matrix of the coefficients (User’s Guide, Section 5.1), and the title used in labeling the output.

- smpl=SMPL series or formula (User’s Guide, Section 5.2)

You can supply a series or a formula that can be evaluated across entry numbers. Entries for which the series or formula is zero or “false” will be skipped, while entries that are non-zero or “true” will be included in the operation.
method=lad/quantile
quantile=quantile to use for METHOD=QUANTILE [not used]

METHOD chooses between LAD and quantile estimation methods. If using
METHOD=QUANTILE, you can use the QUANTILE option to specify the quantile to
use. See the “Technical Information” for details.

iters=number of iterations

Allows the user to control the number of iterations used for the linear program-
ing algorithm. The default value depends on the number of parameters, but is
generally set to 100.

bandwidth=bandwidth for computing scale factor
xxscale=direct value for the scale factor

The BANDWIDTH option allows you to provide the bandwidth to use in computing
the scale factor for the covariance matrix. Use XXSCALE if you want to supply the
scale factor yourself. See the Technical Information for a description of the
default values for these.

lastreg/[nolastreg]
equation=equation to estimate

LASTREG will re-estimate the most recent regression. Use the EQUATION option
to estimate a previously defined equation. If you use it, omit the supplementary
card.

define=equation to define (User’s Guide, Section 1.1.9)
frml=formula to define (Section 1.1.9)

These define an equation and formula, respectively, using the results of the
estimation. You can use the equation/formula for forecasting or other purposes.

Technical Information

LAD chooses $\beta$ to minimize

\[
(1) \quad \sum |y_t - X_t\beta|
\]

while the quantile regression minimizes

\[
(2) \quad \sum_{y_t - X_t\beta > 0} \alpha |y_t - X_t\beta| + \sum_{y_t - X_t\beta < 0} (1-\alpha) |y_t - X_t\beta|
\]

where $\alpha$ is the quantile requested. Note that LAD is a special case of the quantile
regression with $\alpha = 0.5$. The only difference is that the function value would be half
as large.

Because the functions being minimized aren’t differentiable, $\beta$ can’t be estimated
using standard “hill-climbing” methods. Instead, a variant of linear programming is
used.
For either estimator, the optimum is at the value of $\beta$ which gives an exact fit for at least $K$ data points, where $K$ is the size of $\beta$—a specialized linear programming algorithm is used which identifies the best set of those zeroed data points.

LAD is a direct substitute for least squares. Because it uses the absolute value rather than the square of the residuals, it is less sensitive to extreme values. It will be less efficient than least squares when the residuals are well-behaved (for normal residuals, the efficiency is about 60% that of least squares), but will be more efficient for fat-tailed residuals.

The quantile regression isn’t centered at the same point as LAD or least squares, so you should not expect estimates to be necessarily similar. The estimator for a single quantile is usually used in combination with other quantile regressions to provide higher efficiency than could be achieved by using LAD alone.

Koenker and Bassett, for instance, suggest weighted symmetric (about 0.5) combinations of quantile regressions, such as weights of $1/4$, $1/2$ and $1/4$ on quantiles of $1/4$, $1/2$ and $3/4$.

The covariance matrices are estimated as

\[
\eta \times (X'X)^{-1}
\]

where $(X'X)^{-1}$ is the standard inverse cross product of the regressors. With $f$ as an estimate of the density function, the scale factor $\eta$ is

\[
\frac{0.25}{f(0)^2}
\]

for LAD and

\[
\frac{\alpha(1-\alpha)}{f(x_\alpha)^2}
\]

where $x_\alpha$ is (an) $\alpha$ fractile of the residuals.

Note that while the parameter estimates are robust to non-normality, the estimates of the covariance matrix are not robust against heteroscedasticity and similar problems. More complex quantile regression techniques exist that can produce robust covariance estimates in these circumstances—these are currently not supported in RREG.

RREG will compute $f$ using a Gaussian kernel (see the DENSITY instruction), and bandwidth

\[
\frac{0.79 \times IQR}{N^{1/5}}
\]

where $IQR$ is the interquartile range, and $N$ is the number of observations. This choice has certain general optimality properties (see discussion in Pagan and Ullah (1999)), but can be too narrow in some circumstances.
If you want to override this choice of bandwidth, you can use the BANDWIDTH option. And if you want to choose your own scale factor, use the XXSCALE option.

Example

```
linreg logy / resols
# constant logk logl
linreg(smpl=t<>4.and.t<>10) logy
# constant logk logl
rreg logy / reslad
# constant logk logl
rreg(smpl=t<>4.and.t<>10) logy / reslad
# constant logk logl
```

This is part of an example from Greene (2003), page 445. It estimates a Cobb-Douglas production function. There are two very large outliers, so the second LINREG estimates with those omitted. As an alternative to dropping the outliers entirely, the RREG estimates by LAD to reduce their effect on the estimates. In fact, the LAD estimates with and without the two outliers turn out to be identical.

**Variables Defined by RREG**

- **%BETA**: Coefficient vector (VECTOR)
- **%XX**: Covariance matrix of coefficients, or $X'X^{-1}$ (SYMMETRIC)
- **%TSTATS**: Vector containing the t-stats for the coefficients (VECTOR)
- **%STDERRS**: Vector of coefficient standard errors (VECTOR)
- **%NOBS**: Number of observations (integer)
- **%NREG**: Number of regressors (integer)
- **%NDF**: Degrees of freedom (integer)
- **%FUNCVAL**: Minimized values of equations (1) or (2) (real)
- **%MEAN**: Mean of dependent variable (real)
- **%RESIDS**: Series containing the residuals (series)
- **%DURBIN**: Durbin-Watson statistic (real)
- **%RHO**: First lag correlation coefficient (real)
- **%VARIANCE**: Variance of dependent variable (real)
- **%EBW**: Bandwidth used in estimate of the density (real)
RTOC: Transferring from Real to Complex

RATS has no instruction which allows you to read data directly into complex series. Instead, you can read data into regular real-valued series, and then use \texttt{RTOC} (Real TO Complex) to transfer values to a complex series. \texttt{CTOR} (Complex TO Real) transfers data in the other direction.

\begin{verbatim}
rtoc(options) start end newstart
# list of real series to copy
# list of complex series to copy to
\end{verbatim}

**Parameters**

\begin{itemize}
  \item \texttt{start end} Range of entries to transfer. If you have not set a \texttt{SMPL}, this defaults to the defined range of each real series, determined separately for each series.
  \item \texttt{newstart} Entry in complex series for the \texttt{start} entry of the real series. By default, same as \texttt{start}. The default starting entry may change from series to series if you use the default \texttt{start} and \texttt{end} range.
\end{itemize}

**Description**

\texttt{RTOC} transfers data from the real series on the first supplementary card to the corresponding complex series on the second card. It sets the imaginary parts of the complex series to zero.

See Section 15.5 of the \textit{User’s Guide} for a discussion of preparing data for frequency domain analysis.

**Options**

\begin{verbatim}
[pad]/nopad
\end{verbatim}

By default, RATS sets to zero all the entries of the complex series which it doesn’t explicitly transfer from the real series. Thus the “defined length” of the complex series is just its total length. You can suppress this automatic padding with the \texttt{NOPAD} option.

RATS will pad both ends of the series if necessary. For instance, residuals from a regression involving lags have beginning entries which aren’t defined. If you use the defaults for the \texttt{RTOC} parameters, it will transfer the defined portion of the real series and set the initial undefined observations to zero.
even/[noeven]
With **EVEN**, RATS will take a one-sided sequence and turn it into a symmetric two-sided sequence by reflecting the sequence around the midpoint of the series. This can be useful if you have a set of autocovariances which have only been computed in one direction in the real domain.

[labels]/nolabels
**RTOC** transfers the labels of the real series to the complex series unless you use **NOLABELS**.

**Missing Values**
**RTOC** sets to zero any entries of the complex series which correspond to missing values in the real series.

**Example**
This transfers series **RESIDS1** to complex series 1 and **RESIDS2** to complex series 2 and pads both complex series to 256 entries.

```
frequency 5 256
rtoe
#  resids1  resids2
#   1       2
```

**Using Matrices**
It is possible to read and write complex series with the matrix input/output instructions **READ**, **INPUT** and **WRITE**. You must use **OVERLAY** to place a complex vector over the set of entries to be read or written.

For example, the following overlays complex series 2 with **CSERIES** (a **VECTOR** of **COMPLEX** numbers) and reads data for it using **READ**:

```
freq 5 128
open data freqs.dat
declare vector[complex] cseries
overlay %z(1,2) with cseries(128)
read cseries
```

**Variables Defined**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%NOBS</td>
<td>Number of observations transferred (taken from the last series on the list)</td>
</tr>
</tbody>
</table>
SAMPLE — Extracting Data and Changing Frequencies

SAMPLE extracts a specified set of values from series and puts them into news-series. You can extract values by specifying a sampling interval or by using an indicator series.

\[ \text{sample( options ) \ series \ start \ end \ newseries \ newstart} \]

Parameters

- **series**: The series providing the values.
- **start**, **end**: The sampled range of series. If you have not set a SMPL, this defaults to the defined length of series.
- **newseries**: Resulting series, which will contain the extracted values. By default, newseries=series.
- **newstart**: Starting entry for newseries. newstart=start by default.

Options

The two options for SAMPLE are mutually exclusive. If you don’t specify an option, SAMPLE uses INTERVAL=1.

- **interval=sampling interval[1]**
  Use this option for a regular sampling interval. SAMPLE will copy the entries:
  - start, start+interval, start+2*interval, start+3*interval, ...
  into consecutive entries of newseries, beginning at entry newstart.

  You can use this to change the frequency of a data series after it has been read into RATS. It is, however, rather clumsy compared to the options available on the DATA instruction, which has many choices for method of compaction and does the translation automatically. See Section 2.3 in the User’s Guide.

- **smpl=SMPL series or formula (User’s Guide, Section 5.2)**
  With this option, SAMPLE extracts only those entries of series which correspond to entries where the SMPL series or formula is non-zero or “true”.

  SAMPLE with the SMPL option is the simplest way to filter observations out of a data set, if you actually need a compressed data set without gaps. For instance, if you have a daily data set with some missing data and want RATS to skip back to the next valid data point when it needs a lag, you need to use SAMPLE to remove the missing values.

  If you just want to skip certain entries when executing a particular instruction, you can use the SMPL option available on many instructions.
Examples

calendar(q) 1980:1
all 2006:4
open data quarters.rat
data(format=rats) / qseries
sample(interval=4) qseries 1980:1 * q1 1
sample(interval=4) qseries 1980:2 * q2 1
sample(interval=4) qseries 1980:3 * q3 1
sample(interval=4) qseries 1980:4 * q4 1
calendar(a) 1980:1

reads a quarterly series, breaks out separate series (q1, q2, q3 and q4) for each quarter, and then resets the calendar to annual. Note that it’s very important for the start and newstart parameters to be correct—it’s the start parameter that determines which period within the year that you get.

set filter_missing = %valid(sp500)
sample(smpl=filter_missing) sp500 / c_sp500
cal(irregular)

C_SP500 has the same set of values as SP500, except that the missing values (for non-trading days) have been removed. Note that, while SP500 is regular daily data, C_SP500 is not, so the CALENDAR is changed to IRREGULAR.

Notes

SAMPLE is not your best choice for drawing random subsamples. The instruction BOOT combined with SET is the correct way to do that. For instance, to draw a random sample of size 20 from a data series (x) with 100 observations:

boot entries 1 20 1 100
set draw 1 20 = x(entries)

See Section 13.2 in the User’s Guide for more information.

Variables Defined

%NOBS Number of observations in the new series (INTEGER)
SAVE — Saving an Edited RATS Format File

SAVE saves the modified RATS data file that you are currently editing. SAVE is the only instruction that actually alters the data file itself: RATS keeps all changes in memory until you do a SAVE. SAVE does not close the file, so you can save some changes and continue editing.

```
save  (no parameters)
```

Example

dedit(new) prices.rat
store price90 price80 price70
save

This creates a new RATS format file called PRICES.RAT, stores three series, and then saves the changes to the file.

See Also . . .

UG, Section 2.5 RATS format data files. See this section for complete list of instructions for working with RATS format files.
DEDIT Opens a RATS format file for editing.
QUIT Aborts data editing without making any changes.
**SCATTER — High-Resolution X-Y Scatter Plots**

`SCATTER` produces high-resolution scatter plots. It has many similarities to the `GRAPH` instruction, which produces time-series graphs.

```
scatter(options)  number of pairs  hfield  vfield
  #  x-series  y-series  start  end  representation
```

### Parameters

- **pairs**
  
  Number of pairs of series to plot against each other. You can graph up to 20 pairs with a single instruction.

- **hfield  vfield**
  
  These parameters, in conjunction with the `HFIELDS` and `VFIELDS` options of `SPGRAPH`, allow you to put multiple graphs on a single page. By default, the fields are filled in columns, starting at the top left (field 1,1).

### Supplementary Cards

For each pair of series on the graph, there is one supplementary card.

- **x-series**
  
  The series on the horizontal axis. Usually all pairs use the same `x-series`, but that is not required.

- **y-series**
  
  The series on the vertical axis.

- **start  end**
  
  (Optional) Range of entries for which this (x,y) pair is graphed. If you have not set a `SMPL`, this defaults to the common defined range of the `x-series` and the `y-series`. *Note that start and end can be different for each pair of series on the graph.*

- **representation**
  
  (Optional) This lets you use an integer number between 1 and 30 to select the color, pattern or symbol that RATS will use for this (x-series,y-series) pair. You can choose from the default styles provided by RATS, or you can use graph style sheets to customize these. See Section 3.5 for details.

  Normally, you can just omit this parameter. RATS will automatically assign a different color or pattern to each pair.

### Missing Values

RATS leaves out any entry for which *either* of the two series in the pair is missing.
Options—Quick Reference

The following is a list of all of the options for SCATTER. Many of these are identical to options on GRAPH, some are unique to SCATTER, and some operate differently with SCATTER than with GRAPH. See the section on GRAPH for details on the options common to both instructions. We describe the options specific to SCATTER in more detail below, with options grouped by function.

### SCATTER Options

**axis**=none/vertical/horizontal/both

**extend**=[none]/vertical/horizontal/both

**hgrid**=VECTOR of grid values

**hlabel**=horizontal scale label

**hlog**=base for a log scale

**hmax**=value for right boundary

**hmin**=value for left boundary

**hpicture**=picture clause for x-axis

**hscale**=[lower]/upper/both/none

**hshade**=RECTANGULAR with shading zones

**hticks**=max number of horizontal ticks

**lines**=RECTANGULAR with intercept/slope

**omax**=max value for overlay scale

**omin**=min value for overlay scale

**ovcount**=number of series for overlay

**overlay**=dots/symbols/line/bar/poly/ filled/spike/step

**ovkey**/[nooverkey]

**ovlabel**=label for the overlay scale

**ovsamescale**/[noovsamescale]

**style**=dots/[symbols]/line/bar/poly/

filled/spike/step

**vgrid**=vector of grid line values

**vlabel**=vertical scale label

**vlog**=base for a log scale for y-axis

**vmax**=value for upper boundary

**vmin**=value for lower boundary

**vpicture**=picture clause for y-axis

**vscale**=[left]/right/both/none

**vshade**=RECTANGULAR with shading zones

**vticks**=max number of vertical ticks

**xlabel**=VECT[STRING] for x-axis labels

### Function

- **Draw x=0 and/or y=0 axes**
- **Extend grid lines across graph**
- **Sets position of grid lines**
- **Adds a label to the x-axis**
- **Selects a log scale for x-axis**
- **Sets the maximum x-axis value**
- **Sets the minimum x-axis value**
- **Format of x-axis scale values**
- **Placement of x-axis scale**
- **Shading zones for x-axis**
- **Number of tick marks on x-axis**
- **Draw lines given slope/intercept**
- **Sets max value of overlay scale**
- **Sets min value of overlay scale**
- **# of series using overlay scale**
- **Style used for overlay series**
- **Adds a key for overlay series**
- **Label for the overlay scale**
- **Same scale for regular & overlay**
- **Style of graph**
- **Sets grid for vertical axis**
- **Label for vertical axis**
- **Selects a log scale for y-axis**
- **Sets maximum y-axis value**
- **Sets minimum y-axis value**
- **Formatting of y-axis scale values**
- **Placement of vertical scale**
- **Shading zones for y-axis**
- **Number of tick marks on y-axis**
- **Strings for labeling x-axis**

### Options Common to GRAPH and SCATTER

**[box]/nobox**

**footer**=footer label

**frame**=[full]/half/none/bottom

**header**=header string for graph

### Function

- **Superseded by FRAME option**
- **Adds a footer label below graph**
- **Controls frame around the graph**
- **Adds a header to the graph**
Scatter

[kbox]/nokbox
key=[none]/upleft/upright/loleft/loright/above/below/left/right
kheight=fraction of graph height
kwidth=vector of labels or strings
[ksample]/noksample
patterns/[nopatterns]
smpl=series or formula for smpl
subhead=subheader string
window=string for window title

Draws a box around the key
Location of Key
Sets height of the key box
Use to supply custom key labels
Sets width of the key box
Include line/fill samples in key
Use patterns rather than colors
Selects subset of entries to graph
Adds a sub-header to the graph
Custom title for graph window

Representation and Style Options

style=dots/[symbols]/lines/bar/polygon/spike/step

DOTS draws a filled circle at each point. These become smaller as the number of points plotted grows. It will use different colors to represent different pairs if color is available.

SYMBOLS uses a small symbol to represent each data point. If you plot more than one pair on a graph, RATS will use a different color symbol for each pair of series, or, with NOPATTERNS, a different type of symbol for each pair.

LINES connects the consecutive pairs of points on the graph with lines.

BAR, POLYGON, and FILLED should be used only if the x-axis series is in increasing order. A bar graph draws a filled rectangle centered at each x value to its corresponding y value. The POLYGON style is similar but connects the (x,y) points with a line and fills in the area between that line and the axis. FILLED is currently identical to POLYGON, but may be differentiated in future releases.

SPIKE is similar to BAR, but uses narrow “spikes” instead of wide bars.

STEP does a “step” graph, a line graph with constant values across each interval.

patterns/[nopatterns]

This chooses the way SCATTER distinguishes among multiple pairs of series. Ordinarily, RATS will use a different color dot or box for each pair. If you print the graph on a black and white printer, RATS uses a different type of symbol for each pair of series. If you want to see on the screen (approximately) how the hard copy will appear, use the PATTERNS option—RATS will use different symbols rather than different colors on the graph.

Labelling Options

hlabel=horizontal scale label ("..." or string) [none]
vlabel=vertical scale label ("..." or string) [none]

These provide labels for the horizontal and vertical scales. The placement of the labels depends upon your choices for HSCALE and VSCALE. The HLABEL will be centered at the bottom just below the horizontal tick marks for HSCALE=LOWER or...
Scatter

BOTH, or centered at the top above the tick marks if you use HSCALE=UPPER. The VLABEL is centered at the left for VSCALE=LEFT or NONE, and centered at the right for VSCALE=RIGHT. It appears on both sides with VSCALE=BOTH.

xlabels=VECTOR of STRINGS for arbitrary tick mark labels [unused]
Use this option to supply an arbitrary set of labels for the tick marks on the x-axis (by default, SCATTER uses values taken from the x-axis series itself). You can supply any size VECTOR of STRINGS. SCATTER will evenly space the supplied strings across the x-axis.

Axis and Scale Options

axis=[both]/vertical/horizontal/none
This option controls the plotting of vertical (x=0) and horizontal (y=0) axes on the graph. Regardless of your choice of AXIS, you will only get a particular axis if the 0 value lies within the graph range.

hscale=[lower]/upper/both/none
vscale=[left]/right/both/none
These control the placement of the horizontal and vertical scales on the graph. The horizontal scale indicates the values of the x-series. You can place it on the bottom of the graph (the default), on the top of the graph, on both the top and bottom, or you can omit it entirely. The vertical scale indicates the values of the y-series. You can place it on the left of the graph (the default), the right, both left and right or omit it.

hmax=value for right boundary [largest value of x-series]
hmin=value for left boundary [smallest value of x-series]
vmax=value for upper boundary [largest value of y-series]
vmin=value for lower boundary [smallest value of y-series]
These set the boundary values for the plot range. For instance, HMIN=0.0, VMIN=0.0 puts the origin at the lower left corner of the graph. Pairs that don’t fit within the specified ranges are omitted from the graph.

hlog=base for a log scale for the horizontal axis [not used]
vlog=base for a log scale for the vertical axis [not used]
Use one or both of these to graph data on a semi-log or log-log scale. The base chosen really affects only the levels that get labeled, which will always be powers of the base. The values of 10, 2, 4 and 5 are most likely to work best.

hticks=maximum horizontal ticks [7]
vticks=maximum vertical ticks [9]
These set the maximum number of labeled tick marks on the horizontal scale and vertical scales, respectively.
hpicture=picture clause for horizontal axis [shortest that works]
vpicture=picture clause for vertical axis [shortest that works]

You can use these options to set the representation for the numeric labels on the two axes. For instance, VPICTURE=' *.## ' will show the numbers on the vertical axis with two digits right of the decimal. By default, SCATTER chooses the shortest representation that can show all values accurately. See DISPLAY for details on picture clauses.

extend=[none]/both/vertical/horizontal

Normally, RATS marks the vertical and horizontal axis with small tick marks outside the graph. You can use the EXTEND option to have RATS draw dotted grid lines from each tick mark all the way across or down the graph. HORIZONTAL draws horizontal grid lines, VERTICAL draws vertical grid lines, and BOTH draws horizontal and vertical grid lines.

hgrid=VECTOR of grid line values for the horizontal axis [unused]
vgrid=VECTOR of grid line values for the vertical axis [unused]
hshade=RECTANGULAR with shading zones for horizontal axis [unused]
vshade=RECTANGULAR with shading zones for vertical axis [unused]

These options provide you with the ability to highlight single values (with grids) or ranges of values (with shading). The “grid” options give a VECTOR with the x (y for VGRID) values at which a vertical (horizontal) line will be placed. The “shade” options take a RECTANGULAR array with two columns, where each row of the array gives start and end values for a zone which will be shaded from top to bottom (for HSHADE) or side to side (for VSHADE).

lines=Kx2 RECTANGULAR array with intercept/slope pairs [unused]

This allows you to draw lines on the graph by supplying pairs of intercept and slope values for each line. The array should have one row for each line you wish to draw. For each row, put the intercept value in the first column and the slope value in the second column.

Two-Scale Graph Options

overlay=dots/symbols/lines/bar/polygon

This gives the style for an “overlay” graph, where one or more of the pairs is graphed using a different style or on a different scale. (See the STYLE option for a description of the choices). The OVCOUNT option tells how many of the series are given this treatment—their scale is placed on the right vertical axis. With OVSAMESCALE you can force all the data onto the same vertical scale, so the only difference is in the presentation style for the overlay series. OMAX, OMIN, OVLABEL, and OVKEY control other aspects of the overlay-graph presentation.

ovcount=Number of series for right-side scale [1]

The last Number series listed on the supplementary cards are graphed using the right-side (overlay) scale and style. The other series are graphed using the left-side scale and style.
Scatter

\textbf{omax} = Maximum value for right-side scale \textbf{[largest value]}
\textbf{omin} = Minimum value for right-side scale \textbf{[smallest value]}
\textbf{ovlabel} = Vertical scale label (in quotes: "..." or \textbf{STRING}) \textbf{[none]}

OMAX and OMIN allow you to set the maximum and minimum values, respectively, for the overlay scale. These function like the \textbf{MAX} and \textbf{MIN} options (which control the left-side scale when doing a two-scale graph). OVLABEL allows you to supply a label for the right-side scale.

\textbf{[ovkey]/noovkey}

You can use NOOVKEY to eliminate the key for the overlay series, if the meaning is either obvious, or provided using the labels.

\textbf{ovsamescale/[noovsamescale]}

You can use OVSAFESCALE to force both the regular and the overlay series to share a common scale.

**Examples**

This graphs inflation vs unemployment with one set of symbols (squares) for the period 1954 to 1968 and another (diamonds) for 1969 to 1983.

\begin{verbatim}
scatter(style=symbol,header="Inflation vs. Unemployment", $ patterns,hlabel="Unemployment",vlabel="Inflation") 2
# unemp inflation 1954:1 1968:1 1
# unemp inflation 1969:1 1983:1 2
\end{verbatim}

This computes a histogram for the series AGES, presented as a bar graph. The title on the graph is “Histogram of Ages” and it goes into a window labeled “Histogram”.

\begin{verbatim}
density(type=histogram) ages / grid density
scatter(style=bar,window='Histogram',header='Histogram of Ages')
# grid density
\end{verbatim}
This draws a set of data from a gamma distribution, estimates the sample density, then graphs the estimate with the true density. The actual density is done using the (filled) polygon style, while the estimated density overlays that with a line. The colors are adjusted so the line comes in as solid black. `OVSAME` is used to force both to use a single scale. The graph is shown below.

```plaintext
all 1000
set test = %rangamma(4.0)
density(bandwidth=1.00) test / x fx
set actual = exp(log(x)*3.0-x-%lngamma(4.0))
scatter(style=polygon,overlay=line,ovsame) 2
 # x actual / 4
 # x fx / 1
```
This rather complex example is taken from the cumulated periodogram procedure (on the file CUMPENGDM.SRC).

```
ctor 1 half
    # 1 2
    # actual white_noise

label actual white_noise
    # "Actual" "White Noise"

set freqs 1 half = t-1
grparm(font="Symbol") axislabels *
spgraph

scatter(style=line,key=upleft, $
    header="Cumulated Periodogram Test", $
    xlabels=||"0","p/4","p/2","3p/4","p"||) 2
    # freqs white_noise 1 half
    # freqs actual 1 half
display(store=gaplabel) "gap = " #.##### %maximum

if actual(%maxent)<white_noise(%maxent)
    grtext(align=left,x=%maxent-1,y=actual(%maxent)-.01) gaplabel
else
    grtext(align=right,x=%maxent-1,y=actual(%maxent)+.01) gaplabel
spgraph(done)
```

**Cumulated Periodogram Test**

![Graph of Cumulated Periodogram Test](image)
SEASONAL — Creating Seasonal Dummies

SEASONAL creates a seasonal dummy series.

```
seasonal  series  start   end
```

Wizard

You can use the Trend/Seasonals/Dummies wizard on the Data menu to create seasonal dummies.

Parameters

- **series**: Series to set as a seasonal dummy.
- **start**  **end**: Range of entries to set. If you have not set a SMPL, this defaults to the ALLOCATE range plus seasonal-1. See the explanation on the next page.

Two additional parameters (seasonal and period) used in older versions have been replaced by the SPAN and PERIOD options described below. Version 7 will still recognize the deprecated parameters, but use the options for new programs.

Options

- **span=** the seasonal span  [CALENDAR seasonal]
- **period=** first period to receive value 1  [last period in year]

SPAN sets the seasonal span, or periodicity, in terms of the number of periods per year (12 for monthly, 4 for quarterly). This defaults to the seasonal defined by the CALENDAR instruction.

PERIOD indicates the first period (between start and end) which is to get the value 1. In other words, this determines the month or quarter represented by the dummy. This defaults to the entry start+SEASONAL-1, that is, the dummy is set up for the last period within the year. See the next page for details.

- **centered/ [nocentered]**

If you use CENTERED, SEASONAL produces a centered seasonal dummy rather than a standard 0-1 dummy. From a standard dummy, subtract 1.0/seasonal from each entry. For instance, a 4th quarter dummy will use the sequence -.25, -.25, -.25, .75. Centered dummies have certain advantages when you use seasonal–1 of them together with a CONSTANT. You need to be careful, however, as a full set of seasonal centered dummies is linearly dependent.
Seasonal

Examples

seasonal(period=1948:12) december 1948:1 2006:12
seasonal(period=1948:6) june 1948:1 2006:12

sets up the series DECEMBER as a December seasonal (one in every 12th entry, beginning with 1948:12) and the series JUNE as a June seasonal. Make sure you define the dummy over the full range that you will need, using start and end as needed.

calendar(q) 1947:1
allocate 2006:4
seasonal seasons

defines a 4th quarter dummy over a period which runs 3 entries beyond 2006:4. It is equivalent to: seasonal(span=4,period=1947:4) seasons 1947:1 2007:3

Using Complete Sets of Dummies

To put a complete set of seasonals in a regression, the simplest procedure is to create a single dummy for the last period of the year and use its leads. That way, you don’t need to create separate dummies for each period. The default for end is larger than the highest entry on ALLOCATE so that all these leads will be defined.

You can use the series created in the second example to cover all four quarters by

linreg . . .
# seasons{-3 to 0} ... 

(Use SEASONS{-2 TO 0} if you include the CONSTANT). The only tricky part to this is determining which quarter corresponds to each lead in the regression output. SEASONS, itself, is the 4th quarter dummy. SEASONS{-1} (lead 1) is the third quarter dummy, SEASONS{-2} is the second quarter.

If you define centered seasonals, you should use seasonal-1 of them together with CONSTANT to cover all the seasons, for instance,

seasonal(centered) centered_seas
linreg . . .
# constant centered_seas{-2 to 0} ... 

There are two advantages for this versus using uncentered (0-1) dummies:

- The coefficient on CONSTANT has the same basic meaning as in an analogous non-seasonal regression. With 0-1 dummies, the CONSTANT is the intercept for the omitted seasonal.
- The coefficients on the dummies are the difference between each season and the overall average. With 0–1 dummies, they are the difference between each season and the omitted one.

There is, of course, no difference in fit, just in interpretation of the coefficients.
Sometimes, however, you may need a separate series for each period. For quarterly
data, it is probably simplest to just do three or four separate \texttt{SEASONAL} instructions. With more periods per year, however, you will probably want to automate the process. Use a vector of series, and define the dummies in a loop:

\begin{verbatim}
dec vect[series] seasonals(12)
do i=1,12
   seasonal(period=i) seasonals(i) 1947:1 2003:12
end do i
\end{verbatim}

creates a vector of series which have the twelve dummies as its twelve elements. You can then use \texttt{SEASONALS} in a regressor list to add the full set.

\textbf{SEASONAL with Panel Data}

Panel data requires different treatment from standard time series data. Leads and lags are not allowed to cross from one individual to another. Thus, the leads of a single dummy will not work correctly: \texttt{LINREG} will drop entries at the end of each cross-section. \textit{With panel data, you will have to create a separate dummy for each quarter or month.} The process for doing this is described at the bottom of the previous page.

\textbf{SEASONAL with Daily/Weekly Data}

\texttt{SEASONAL} is unlikely to be very helpful with daily, weekly and related frequencies. The only type of useful dummy variable you can create is one for a particular week-day, using as \texttt{SPAN} the number of days per week.

Use \texttt{SET} with logical and date functions to generate other types of calendar dummy variables. For instance,

\begin{verbatim}
cal(daily) 1990:1:1
cal 2006:12:31
set monday = %weekday(t)==1
print 1 20
\end{verbatim}

This uses the \texttt{%WEEKDAY} function (which returns “1” for Mondays) to create a \texttt{MONDAY} dummy variable.

You can use other date functions, such as \texttt{%MONTH} or \texttt{%PERIOD} as an alternative to \texttt{SEASONAL} for monthly, quarterly and other dummies:

\begin{verbatim}
set january = %month(t)==1
\end{verbatim}

creates a January dummy variable (1 for all entries which fall in a January, 0 otherwise).

See Section 2 for a list of date functions.


**SEED — Initializing the Random Number Generator**

The **SEED** instruction allows you to supply a seed value for the random number generator. This is generally used to help you test programs that involve random draws, by forcing the same random sequence each time you run the program.

```
seed value
```

**Parameters**

- **value**: The desired seed. If `value` is 0 or omitted, RATS computes a new seed using the current date and time. You can select any other value as your seed. While it is common to choose “random-looking” numbers like the 13939 below, they have no particular advantage over seeds like 1 or 777.

**Description**

The seed is an integer which initializes the RATS random number generator. This generator is used by the instructions **SIMULATE** and **BOOT** and by the functions such as `%RAN`, `%UNIFORM`, `%RANGamma` and `%RANWishart`. The seed uniquely determines the sequence of (pseudo-) random numbers. The algorithms used are provided on the next page.

RATS normally sets the seed using the time and date at which the program begins execution. This default seed will be different each time you run the program.

Instead, you can use **SEED** to rerun a program with the *same* set of generated numbers. This is very helpful when you are testing a program which draws random numbers, as it is easier to find and fix coding errors when your “data” doesn’t change each time you run the program.

Put **SEED** right at the beginning of your code so it will be easy to remove once the program is running correctly.

**Notes**

**SEED** only permits you to reproduce random numbers when you use *exactly* the same instructions in the same sequence. Consider, for instance,

```
declare vector b1(5) b2(5) b(10)
seed 13939
compute b=%ran(1.0)
seed 13939
compute b1=%ran(1.0)
compute b2=%ran(1.0)
```
The entries of $B_1$ match the first five entries of $B$. However, the entries of $B_2$ are different from the last five entries of $B$. Technically, RATS uses an acceptance-rejection algorithm for generating Normal values from uniform. This requires an extra draw from the uniform at the beginning of each set of numbers.

**Algorithms**

RATS uses the period $2^{191}$ pseudo-random number generator from L’Ecuyer (1991). This generates the same set of numbers from a given seed on any platform supported by RATS that is version 6.10 or later. (The random number generator was changed with version 6.10). Real numbers in the range $[0,1]$ are obtained from this by dividing the integer by its period. These are used directly by the function \%UNIFORM (possibly with scaling and translation to fit the desired range) and \texttt{BOOT}.

Normal and gamma deviates are generated from the uniforms by acceptance-rejection algorithms.

**See Also . . .**

- **\texttt{SIMULATE}** Forecasts a model with random shocks.
- **\texttt{BOOT}** Draws random entry numbers.
- \%\texttt{RAN(x)} Function returning draws from Normal($0,x^2$) distribution.
- \%\texttt{UNIFORM(x1,x2)} Function returning draws from a Uniform($x_1,x_2$)
- \%\texttt{RANINTEGER(L,U)} Function returns integer in $[L,U]$
SELECT — User Selection from a List

SELECT brings up a dialog box that allows the user to select items from a list of integers, strings, or data series. You can set this to allow the user to make only one choice or many. You can also include selection boxes as an item in \texttt{DBOX}.

\begin{verbatim}
select(options) number
select(options) number selectlist
\end{verbatim}

Parameters

\textit{number}\quad If you use just the one parameter, the user can make only one choice from the list, and \texttt{number} will be an integer indicating which it was. The meaning of the return value depends upon the list type—see the options below.

If you use both parameters, the user can select any number of items and \texttt{number} returns the number of items selected.

\textit{selectlist}\quad If you want to allow the user to select more than one item, supply a new variable name for the \texttt{selectlist} parameter. \texttt{RATS} will create this variable as a \texttt{VECTOR} of \texttt{INTEGERS} containing the list of return values for the selected items. The meaning of the return values depends upon the list type.

Options

\textit{series}\quad \textit{strings}=VECTOR of STRINGS to list
\textit{list}=VECTOR of INTEGERS to list
\textit{regressors}\quad

These four (mutually exclusive) options determine the type of list displayed:

\begin{itemize}
  \item Use \texttt{SERIES} to request a selection from the list of series currently in memory. When you use \texttt{SERIES}, the return values are the series numbers. You can use a \texttt{selectlist} array directly in a regression list, and in other locations where a \texttt{RATS} instruction requests a “list of series.” Note that the special series \texttt{CONSTANT} is never displayed as one of the series in the list.
  \item Use \texttt{STRINGS} to request a selection from a list of labels. You must declare and set the \texttt{VECTOR[STRINGS]} before executing \texttt{SELECT}. The return values are the selected positions (element numbers) in \texttt{stringvector}.
  \item Use the \texttt{LIST} option to request a selection from an arbitrary list of integers. You must declare and set the \texttt{VECTOR[INTEGERS]} before executing \texttt{SELECT}. \texttt{SELECT} returns the actual integer values selected (not the positions in the array).
\end{itemize}
• Use the `REGRESSORS` option to request a selection of regressors from the last completed regression. The return values are the selected positions in the list.

`prompt="prompt string"`

Use PROMPT to display a message at the top of the dialog box.

`status=INTEGER variable for status`

The `STATUS` option sets this to 0 if the user cancels the selection and 1 if the user OK's it. You should always get and check the status.

`limit=limit on list length`

By default, SELECT will list all series in memory or all elements of the `stringvector` or `integervector`. You can use the `LIMIT` option to have SELECT display only the first `limit` items available. For example, the instruction `SELECT(SERIES,LIMIT=N)` will list only series numbers 1 through N. Note: `LIMIT` has nothing to do with limiting the number of items the user can select.

`LIMIT` is most useful with the `SERIES` option, since you may want to restrict the user to choosing only from the original data series, and not derived series like residuals. If you know that there were four original series, with

```
select(prompt="Graph Which Series",series,limit=4,status=ok)  $
  seriesnum
if ok {
  graph(key=upleft) 1
  # seriesnum
}
end if
```

you could ask the user to select one of the first four series.

**Example**

```
select(series,prompt="Select Explanatory Variables",status=ok)  $
  nselect reglist
if ok {
  linreg y
  # constant reglist
}
end if
```

This requests the user to select one or more series. The selected variables are then used in a regression.
Set

SET — General Data Transformations

**SET** is the general data transformation instruction. By using arithmetic operators and functions, you can create almost any transformation. RATS also has a number of special purpose instructions such as **SEASONAL** for creating seasonal dummies and **FILTER** and **DIFFERENCE** for difference transformations. **SET** operates on complete data series. Use **COMPUTE** for calculations with scalars or matrices.

```
set( options )  series  start   end = function(T)
```

Wizard

You can use the Transformations wizard on the Data menu to do general transformations.

Parameters

- **series**
  - The series to create or set with this instruction.
- **start**  **end**
  - (Optional) The range of entries to set. If you have not used a **SMPL** instruction, this defaults to the **ALLOCATE** range, regardless of the series involved. However, in effect, RATS cuts this back to the maximum range available given the transformation by setting to “missing” all entries which it cannot compute.
- **function(T)**
  - This is the function of the entry number \( T \) which gives the value of entry \( T \) of **series**. There should be at least one blank on each side of the \( = \). The function can actually include multiple expressions, separated by commas. The series will be set to the values returned by the final expression in the list.

Options

- **first=** expression for first entry set
  - Use **FIRST** when the first entry of **series** requires a different formula than the remaining entries; for instance, a benchmark value.
- **scratch/ [noscratch]**
  - You need to use **SCRATCH** whenever you redefine an existing series (that is, when **series** appears in the **function**) and the transformation uses data from more than just the current time period \( T \) of **series**. You don’t need **SCRATCH** if only one of these conditions is true. See the examples for more on this.
- **smpl=SMPL  series or formula**
  - You can supply a series or a formula that can be evaluated across entry numbers. Only entries of the **series** for which the **SMPL** series or formula are non-zero will be set.
Description

This sets the values of entries start to end of series by evaluating the function at each entry T, substituting for T the number of the entry being set. You can omit the T subscript in most cases—see the note below.

The Variable T

When you do a SET instruction, the variable T is always equal to the number of the entry being computed. You can use this to create trend variables and time period dummies:

\[
\begin{align*}
\text{set} & \quad \text{trendsq} = t^{**2} \quad \text{Creates “time” squared.} \\
\text{set} & \quad \text{postwar} = t\geq1946:1 \quad \text{Creates a postwar dummy for 1946 on.}
\end{align*}
\]

Series References and Lag Notation

You can refer to the current entry of a series by just using the series name. A lag of a series can be written series\{lag\}. Leads are negative lags—so GDP\{1\} is GDP in the previous period and GDP\{-1\} is the next period.

If you are referencing a series indirectly, using numbers or a loop index, use index\{0\} to get the current entry. The \{0\} makes it clear that you are referring to a series, not an integer.

Missing Values

SET propagates missing values through the formula. With only two exceptions (the %VALID(x) function and the %IF(x,y,z) function), any operation which involves a missing value returns a missing value.

SET also sets to missing any observation which involves an illegal operation, such as divide by zero and square root of a negative number.

Examples

This replace the series GNP by its log:

\[
\text{set gnp} = \log(\text{gnp})
\]

This accumulates an investment series to create a capital stock series. The 1920:1 entry of CAPITAL is set to 100. Remaining entries set according to the expression:

\[
\text{smpl} \quad 1920:1 \quad \text{1941:1}
\]
\[
\text{set(first=100.)} \quad \text{capital} = .90*\text{capital}\{1\}+\text{investment}
\]

This makes XSWITCH equal to either the current entry of XPOS or of XNEG, depending upon whether X is positive or not:

\[
\text{set xswitch} \quad 1951:1 \quad 1998:4 = %\text{if}(x\geq0.0,\text{xpos, xneg})
\]

See Sections 1.1.5 and 1.7 of the User’s Guide for many more examples using SET.
SHOW — Run-Time Information

You can use the instruction SHOW to display different types of information.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>show memory</td>
<td>This displays the amount of memory available to RATS for workspace, the number of bytes used and the number of bytes remaining.</td>
</tr>
<tr>
<td>show series</td>
<td>Lists the names of all the series currently in use.</td>
</tr>
<tr>
<td>show files file identifier</td>
<td>Lists all files which match the file identifier. You can use the usual wildcards (? and *) in a SHOW FILES instructions to list a specific set of files. For instance,</td>
</tr>
<tr>
<td></td>
<td>show *.rat</td>
</tr>
<tr>
<td></td>
<td>will list all the files in the current directory with .RAT extensions.</td>
</tr>
<tr>
<td></td>
<td>If you omit file identifier, RATS will list all files on the current directory.</td>
</tr>
<tr>
<td></td>
<td>SHOW FILES will not work on the Macintosh.</td>
</tr>
</tbody>
</table>
SIMULATE — Random Simulations of a Model

SIMULATE solves a model, drawing shocks to the equations from a Normal distribution.

```
simulate(options) equations
  # equation forecasts newstart column  (one card per equation)
```

Parameters

- **equations**: Number of equations in the system. Omit this parameter when using the MODEL option.

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. Version 7 will still recognize these older parameters, but we recommend that you use the new options in any new programs.

Options

- **model=model name**
  This is an alternative way to specify the system of equations and is the only way to forecast with a set of FRMLs. MODELS are usually created by GROUP or SYSTEM. **If the model includes any identities, those should be last in the model.**

- **from=starting period for the forecast interval**
- **to=ending period for the forecast interval**
- **steps=number of forecast periods to compute**
  These determine the periods for which forecasts will be computed. If you have set a SMPL, these default to forecast over that range. Otherwise, FROM and TO default to the beginning and end of the most recent estimation range, respectively. If you want something other than the defaults, you can use:
  - FROM and TO to set the starting and ending periods for the forecasts, or
  - FROM and STEPS to set the starting date and number of steps (periods)

- **iters=iteration limit for solution algorithm** [50]
- **cvcrit=convergence criterion** [.00001]
- **damp=damping factor** (1.0=no damping) [1.0]
  These apply if you are simulating a model which contains FRMLs and not just EQUATIONS, and thus requires an iterative solution (User’s Guide, Section 11.3).
**Simulate**

\textbf{results}=$\text{VECTOR[SERIES]}$ for result series

This provides a \text{VECTOR} of \text{SERIES} which will be filled with the results. For instance, \text{RESULTS}=\text{FCASTS} will make \text{FCASTS}(i)$ the series of forecasts for the $i$th equation in the model.

\textbf{cv}=$\text{SYMMETRIC} \ \text{covariance matrix of residuals}$

\textbf{factor}=$\text{RECTANGULAR} \ \text{decomposition of } \Sigma$

You can use \text{CV} to supply a $\Sigma$ matrix, which can be computed using the \text{VCV} instruction. For VAR's, you can just use the \%SIGMA variable defined by \text{ESTIMATE}. If you are using a \text{MODEL}, and you provided the covariance matrix when setting up the model, you can omit the \text{CV} option.

As an alternative, you can use the \text{FACTOR} option to provide a factorization of the covariance matrix of residuals. See Section 10.5 of the \textit{User's Guide}. Note that the distribution (though not the actual simulations) of the forecasts \textit{doesn't depend upon which factorization is chosen for }\Sigma. This option was called \text{DECOMP} in earlier versions. \text{DECOMP} is still recognized as a synonym for \text{FACTOR}.

See “Description” below for details.

\textbf{print/}\textbf{[noprinit]}\textbf{[noprinit]}

Use \text{PRINT} if you want RATS to print the forecasts; \textit{this is not done automatically}.

**Supplementary Cards**

There is one supplementary card for each equation. Include these only if you are not inputting a \text{MODEL}. \textit{Identities should be listed last}.

\textbf{equation} \quad \text{The equation.}

\textbf{forecasts} \quad (Optional) The series for the forecasts of the dependent variable of \textit{equation}. \textit{forecasts} can be the same as the dependent variable.

\textbf{newstart} \quad (Optional) The starting entry for \textit{forecasts}. By default, the same as \text{FROM}.

\textbf{column} \quad If you use the \text{FACTOR} option, this is the column in that matrix which corresponds to this equation. By default, RATS assumes that the order in which you list the \textit{equations} matches the order of the columns in the \text{FACTOR} matrix.
Description

SIMULATE forecasts a system of equations using a random number generator to add shocks to those equations which are not identities. Note that identities must be last in the list of supplementary cards, or at the end of the model.

The shocks at each period are generated by a draw from a Normal(0, Σ) distribution. You put the variance/covariance matrix Σ into SIMULATE in one of the following ways:

1. You can supply it using the CV option. If you use the column fields on the supplementary cards, you will get a rearrangement of this.
2. You can supply a factorization of Σ using the FACTOR option.
3. If you use a MODEL, you can provide the covariance matrix with the CV option when you create the model with GROUP.
4. In all other cases, Σ is taken to be a diagonal matrix with diagonal equal to the variances of the individual equations or formulas.

Comments

The standard way to use SIMULATE is to loop over the number of draws you want to make, compiling statistics on characteristics of the simulated series. Chapter 13 of the User’s Guide describes this in greater detail.

The instruction SEED can be helpful in checking programs which use SIMULATE because it allows you to control the set of random numbers that RATS generates.

Note that SIMULATE isn’t needed to create a series of random numbers or simple transformations of them. A SET instruction using functions like %RAN and %UNIFORM is easier to set up.

If you need the shocks to have some distribution other than Normal, use FORECAST with the PATHS option, using SET instructions to create the series of shocks.
Examples

This computes and copies to a file a 100 element realization of the AR(2) process

\[ y_t = 6.0 + 1.4y_{t-1} - 0.8y_{t-2} + u_t; \quad \text{Var}(u_t) = 4.0 \]

Since the simulation starts out “cold” with zeros for the lagged values, the program generates an additional 48 points at the beginning for burn-in. Simulations start at period 3 because of the two lags:

\[
\text{allocate 150} \\
\text{set } y = 0.0 \\
\text{equation(variance=4.0,coeffs=||6.0,1.4,-.8||) simeq y 2 0} \\
\text{simulate(from=3,steps=148) 1} \\
\text{# simeq y} \\
\text{open copy simul.dat} \\
\text{copy 51 150 y}
\]

This next example does 100 simulations of a six-variable VAR over the period 2007:1 to 2012:4 and fills a pair of vectors with the minimum and maximum achieved by the exchange rate over that period. (The simulated values for the exchange rate are in SIMULS(2), since it is the second equation in the model.)

\[
\text{system(model=canmodel)} \\
\text{variables usargdps canusxsr cancd90d canm1s canrgdps cancpinf} \\
\text{lags 1 to nlags} \\
\text{det constant} \\
\text{end(system)} \\
\text{estimate(noprint)} \\
\text{declare vector minxrate(100) maxxrate(100)} \\
\text{do draw=1,100} \\
\text{\quad simulate(model=canmodel,results=simuls,} \\
\text{\quad steps=24,start=2007:1)} \\
\text{\quad ext(noprint) simuls(2)} \\
\text{\quad compute minxrate(draw)=%minimum,maxxrate(draw)=%maximum} \\
\text{end do draw}
\]
SMPL — Setting the Default Entry Range

SMPL allows you to set the entry range which will be used by default for the start and end on subsequent instructions. SMPL is not as important in RATS as similar instructions in some other time series programs, because most RATS instructions will automatically determine the range to use given the series involved. As you get more familiar with the style used in RATS, you may find yourself using SMPL less and less.

```
smpl  start  end
smpl(series=smpl series)
smpl(reglist)
# list of variables in regression format
```

**Parameters**

`start`  
The starting entry number or date of the range to be used in subsequent instructions.

`end`  
The ending entry or date of the range.

You can use an asterisk (*) for either parameter if you want to fix only one end of the range. RATS will determine the *’ed end separately for each instruction. See the examples.

SMPL with no parameters tells RATS to return to using the maximum possible range.

**Options**

`series=SMPL series or formula` *(User’s Guide, Section 5.2)*
This sets the sample based on the values of a data series or a logical expression that can be evaluated across entry numbers. Any entries for which the supplied series or formula are zero, missing (NA), or “false” will be excluded from the sample. This offers an alternative to the SMPL options available on many instructions, and allows you to do transformations that would be very difficult otherwise.

`reglist/[noreglist]`
Sets the sample range based on a set of regressors, listed in regression format.

**Description**

Following a SMPL instruction, RATS will use the SMPL range on any subsequent instructions for which you do not give an explicit entry range. You can always override the current SMPL by providing the entry range on an instruction.

You can set and clear the SMPL any number of times.

Also, see the description of the SMPL option in Section 5.2 of the User’s Guide.
Examples

Following are two functionally identical sets of instructions. The first uses the range parameters on each instruction, while the second uses both default ranges and a SMPL. In both cases, the DATA instruction reads from 1955:1 to 2006:12, then the other computations are done from 1965:1 to 2006:12.

cal(m) 1955:1
data 1955:1 2006:12  inter infla
linreg inter 1965:1 2006:12
# constant infla
set dif 1965:1 2006:12 = inter-infla
set lgs 1965:1 2006:12 = log(dif)

cal(m) 1955:1
all 2006:12
data  /  inter infla
smpl 1965:1 2006:12
linreg inter
# constant infla
set dif = inter-infla
set lgs = log(dif)

The / tells RATS to use default range.
Sets a new default range
Range omitted, so uses the default range
set by the SMPL instruction.
Again, range omitted, so SMPL range used.

In the next example, both regressions begin in 1950:1. The LINREG runs through the last entry for which both CONS and GNP are defined. The AR1 runs through the last entry for which INVEST, YDIFF, GNP{1} and RATE{4} are defined.

smpl 1950:1 *
linreg(inst) cons
# constant gnp cons{1}
arl(inst,frml=investeq)  invest
# constant ydiff gnp{1} rate{4}

The last example runs a series of NLLS instructions with increasing numbers of lags in the instrument list. By using SMPL(REGLIST) with the maximum number of lags we plan to use, we ensure that the regressions will all be run over a standard range.

smpl(reglist)
# constant consgrow{0 to 6} realret{0 to 6}
dofor nlag = 1 2 4 6
    instruments constant consgrow{1 to nlag} realret{1 to nlag}
    nlls(inst,noprint,optimalweights,frml=h)
end dofor
SOURCE — Secondary Input Files

SOURCE reads in and processes the RATS instructions on the indicated file, then switches processing back to the original input window or file. It is often used to read in procedures stored in separate files, although this is often no longer necessary, as RATS now searches for procedure files automatically. If you have procedures you use regularly, you can also use define a “Procedure Library” that will be read in automatically right at the start of your program—see Section 16.2.1 in the User’s Guide.

\[\text{source( options ) file name}\]

Options

- **echo/[noecho]**
  Normally, RATS won’t show the lines in the SOURCE file as it reads them. Use ECHO if you want to see them. Note that the default setting is now NOECHO.

- **key="key for decrypting file"**
  Contact Estima if you want to obtain software to encrypt source files.

- **status=INTEGER variable returning status [unused]**
  If you use the STATUS option, RATS will set the variable you supply to 1 if the file was successfully opened, or 0 if it was not.

Notes

SOURCE has two primary uses:

- You can save on separate files PROCEDURES and FUNCTIONS, or groups of RATS instructions you use often, and SOURCE them into different programs.
- You can save your initial instructions: CALENDAR, ALLOCATE, DATA, and transformations, and start programs by SOURCEing these, rather than repeating them in each program.

If you use SOURCE within a compiled section, you should note that RATS does not execute the SOURCE instruction during the compilation phase; it is not like an “include” allowed in many programming languages. During the execution phase, RATS temporarily stops executing the compiled code to execute the instructions on the SOURCE file, then switches back. Because the file is not processed during the compilation phase, it cannot include references to procedure parameters or local variables.

Example

```
source dfunit.src
@dfunit(lags=4) tbills
```

This example brings in a procedure called DFUNIT (the Dickey–Fuller unit root test) from the file DFUNIT.SRC and then applies the procedure to the series TBILLS.
**SPECIFY — Bayesian Priors for Vector Autoregressions**

`SPECIFY` defines the prior for a vector autoregressive system. It is a sub-command of `SYSTEM` and must appear in the system definition after the `VARIABLES` and `LAGS` instructions. See Section 10.10 of the *User’s Guide* for more on the use of priors.

```plaintext
specify(type=symmetric, other options)  other’s weight
specify(type=general, other options)
# weights   (if you don’t use the MATRIX option)
```

**Wizard**

The *VAR Estimation* wizard, located on the *Statistics* menu, provides an easy, dialog-driven interface for defining and estimating VAR models.

**Parameter**

- **other’s weight**
  
  This is the relative weight on other variables in a `SYMOMETRIC` prior; it is irrelevant for other prior types. If you omit it, its default value is 1.0.

**Description**

We use the following notation throughout this section:

- variable $j$ refers to the $j$th variable listed on the `VARIABLES` instruction.
- equation $i$ refers to the equation whose dependent variable is variable $i$.
- the standard deviation of the prior distribution for lag $l$ of variable $j$ in equation $i$: denoted $S(i, j, l)$.

The function $S(i, j, l)$ takes the following form:

$$S(i, j, l) = \frac{\{\gamma g(l) f(i, j)\} s_i}{s_j}; \quad f(i,i) = g(l) = 1.0$$

where $s_i$ is the standard error of a univariate autoregression on the dependent variable of equation $i$. We multiply by $s_i/s_j$ to correct for the different scales of the variables. The various options control $f(i,j)$, $\gamma$, $g(l)$.

**Options**

- `type=[symmetric]/general`
- `matrix=RECTANGULAR array of weights for TYPE=GENERAL`

`TYPE` selects the $f(i,j)$ function. `SYMOMETRIC` provides a restricted form of the function, while `GENERAL` allows complete generality. See the discussions below.
**tightness=** overall tightness [.20]

The **overall tightness** is the $\gamma$ parameter in the formula above.

**mean=** first lag mean [1.0]

**mvector=** VECTOR of first lag means

Use one of these if you want the means of the prior distributions on the first own lag to be something other than 1.0. The **MEAN** option sets them all to the specified **first lag mean** value. The **MVECTOR** option sets the mean for the first own lag in the $i$th equation to the $i$th entry of the **VECTOR of first lag means**.

**lagtype=** [harmonic]/geometric

**decay=** lag decay parameter [no decay with lag]

These two options control the function $g(l)$: how the standard deviation changes with increasing lags. With $d$=**lag decay parameter**, RATS uses the formulas:

$$g(l) = l^{-d} \text{ for } \text{LAGTYPE}=\text{HARMONIC} \text{ and}$$

$$g(l) = d^{-1} \text{ for } \text{LAGTYPE}=\text{GEOMETRIC}$$

By default there is no decay of standard deviations with increasing lags. Notice that for **HARMONIC**, $d=0$ is the default and larger values produce a tighter prior, while for **GEOMETRIC**, $d=1$ is the default and smaller values produce a tighter prior.

**full=** RECTANGULAR of information on full system prior

**[scale]/noscale**

These allow you to use priors more general than the standard ones. See the description later in this section.

**The TYPE=SYMMETRIC Option**

This specifies the $f(i, j)$ function as follows:

$$f(i, j) = \begin{cases} 
1.0 & \text{if } i = j \\
1 & \text{otherwise}
\end{cases}$$

where $w$ is the other’s weight parameter. For example:

system(model=sixvar)
variables ip m1 cpr unemp wage cpi
lags 1 to 12
det constant
specify(type=symmetric,tightness=.10,decay=1.0) .5
end(system)
The **TYPE=GENERAL** Option

**TYPE=GENERAL** allows complete freedom in selecting the $f(i, j)$ function. $f$ is either

- input using a supplementary card. You should type the values for $f$ by rows (that is, group by equation). The # goes only on the *first supplementary card line*, and you must use line continuations ($) if all the numbers do not fit on one card.

- provided using the option **MATRIX=RECTANGULAR array**. You must **DECLARE**, dimension and set this array up in advance. Element $i, j$ of this array is $f(i, j)$.

For example:

```plaintext
system(model=sixvar)
variables ip m1 cpr unemp wage cpi
lags 1 to 12
det constant
specify(type=general)
# 1.0 0.5 0.5 0.2 0.2 0.2 $
 0.2 1.0 1.0 0.2 0.2 0.5 $
 0.2 1.0 1.0 0.2 0.2 0.2 $
 0.2 0.5 0.5 1.0 0.2 0.2 $
 0.2 0.5 0.2 0.2 1.0 0.5 $
 0.0 0.5 0.2 0.2 0.5 1.0
end(system)
```

**Full Specification of Priors**

The primary options of **SPECIFY** put rather stringent restrictions on the type of priors that you can use. The options **FULL** and **SCALE** allow some relaxation of these.

**full=** **RECTANGULAR** of information on full system prior

**[scale]/noscale**

With the notation

- $N$: number of equations
- $L$: number of lags
- $D$: number of deterministic variables,

you use **FULL** to input an $(NL + D + 1) \times N$ matrix of “dummy observations.” The structure of each column is as follows:

- **First $NL$ rows** the reciprocals of the standard deviations of the priors on the lags. Use a 0 for a flat prior.

- **Next $D$ rows** the reciprocals of the standard deviations of the priors on the deterministic variables. Use a 0 for a flat prior.

- **Last row** the prior mean of the first own lag divided by its prior standard deviation.
**Dummy Observations**

Suppose you impose the prior $\beta_k \sim N(b, \lambda^2)$ upon a coefficient $\beta_k$. You can represent this as a “dummy observation” with

$$\Psi = \frac{\sigma}{\lambda} \quad \text{and} \quad r = \frac{\sigma}{\lambda} b$$

where $\sigma$ is the standard deviation of the equation being estimated.

The first $NL+D$ elements of each column of the FULL array provide the $\Psi$ values for the dummy observations for the coefficients. The last element provides the $r$ for the first own lag. This allows you:

- complete freedom in setting standard deviations on the lags
- the ability to put mean zero priors on any of the deterministic variables.

**The Option SCALE**

The SCALE option causes ESTIMATE to compute and insert the $s_i$ and $s_j$ factors into the formulas above. This lets you concentrate on the form of the prior without worrying about relative magnitudes. NOSCALE (the default with FULL) requires that you provide directly the $\Psi$ and $r$ values that you want.
SPGraph

SPGRAPH — Special Graphs

SPGRAPH (Special Graph) allows you to create special graphs and graphics presentations. In particular, you can use SPGRAPH to put several complete graphs on a single page, or add one or more strings of text to a graph using GRTEXT. You can also supply a header, footer, and other labels for the special graph.

```plaintext
spgraph(options)
    One or more graph, scatter, or grtext instructions
spgraph(done)  To signal end of special graphs.
```

Description

SPGRAPH simply tells RATS which type of special graph you want. As shown above, you follow this with the instructions which create the graphs themselves, then SPGRAPH(DONE) when you are finished. You can also “nest” SPGRAPH blocks inside other SPGRAPH blocks to create more complex graph presentations.

Please note that you must use SPGRAPH(DONE) at the end even if you only need a single graphing instruction.

Options

- **hfields=number of horizontal fields [1]**
- **vfields=number of vertical fields [1]**

You can use these options, alone or together, to put several graphs on a single page. They tell RATS to divide the specified axis into multiple fields. If you use these options, you can then use the hfield and vfield parameters on GRAPH, GCONTOUR, and SCATTER to control the positioning of each graph. By default, the graph instructions fill the fields beginning with the top left, working down.

- **samesize/[nosamesize]**

Use SAMESIZE if you want all the graphs in the SPGRAPH to be the same size.

- **done**

You must use SPGRAPH(DONE) after all the graphics instructions required for the special graph. RATS will draw the graph as soon as you execute the SPGRAPH(DONE).

- **window="Window title" [none]**

The WINDOW option allows you to set a title for the graph window that will be associated with the graph. By default, graph windows are titled as “Graph.01,” “Graph.02,” etc.
header=header string for graph page [none]
subheader=subheader string [none]
footer=footer string [none]
hlabel=horizontal scale label [none]
vlabel=vertical scale label [none]

These are nearly identical to the options of the same name on GRAPH, SCATTER, and GCONTOUR. They allow you to add labels for the entire SPGRAPH page, rather than the individual graphs. You can use a STRING type variable, or an explicit string in quotes for all five options. See the second example graph. Note: use the analogous GRAPH, SCATTER or GCONTOUR options to label the individual graphs.

xlabels=VECTOR[STRINGS] with labels for the horizontal fields
xpos=[upper]/lower/both/none
ylabels=VECTOR[STRINGS] with labels for the vertical fields
ypos=[left]/right/both/none

These options allow you to add labels for the horizontal and vertical fields. You can supply the labels as a variable of type VECTOR of STRINGS, or in literal matrix notation (such as YLABELS=||"label1","label2"||). The XPOS and YPOS options determine the positioning of the XLABELS and YLABELS respectively. See the examples.

Notes

HFIELDS and VFIELDS offer an alternative to creating full size graphs and then pasting and resizing them into another application (or photoreducing them) in order to put several on a page. RATS will automatically scale all elements of the graphs, including labels, to fit the page.

Use HFIELDS and VFIELDS together to create a “matrix” of graphs. See, for instance, the example below. Once you get beyond two or three graphs on a page, you will probably want to eliminate the tick marks and axis labeling. The graphs will get too cluttered for the labeling to be useful.

You can use the SAMESIZE option to force the graph boxes to be the same size.

Examples

This generates a 3x3 “matrix” of pairwise scatter plots for the three series SLIST(1), SLIST(2) and SLIST(3). The “diagonal” is left empty by skipping the SCATTER instruction if I is equal to J.

spgraph(vfield=3,hfield=3,header="Scatter Plots")
do i=1,3 ; do j=1,3
   if (i<>j)
      scatter(vscale=none,hscale=none,style=symbols) 1 i j
      # slist(i) slist(j)
   end do j ; end do i
spgraph(done)
The example below is taken from the MONTEVAR.PRG example program (Example 13.2 in the User’s Guide). Note the use of the labeling options available on SPGRAPH:

```
grparm(bold) hlabel 18 matrixlabels 14
grparm  axislabel 24
spgraph(header="Impulse responses",xpos=both,xlab=xlabel, $
   ylab=ylabel,vlab="Shock to",vfields=nvar,hfields=nvar)
smpl 1 nstep
do i=1,nvar

  Determine the range needed for all responses of variable I. Calculate from the
  lowest 2 s.e. lower bound to the highest 2 s.e upper bound.

  compute minlower=maxupper=0.0
  do j=1,nvar
      set resp   = first(i,j)(t)
      set stderr = sqrt(second(i,j)(t))
      set upper  = resp + 2 * stderr
      set lower  = resp - 2 * stderr
      compute maxupper=%max(maxupper,%maxvalue(upper))
      compute minlower=%min(minlower,%minvalue(lower))
  end do j

  do j=1,nvar
      set resp   = first(i,j)(t)
      set stderr = sqrt(second(i,j)(t))
      set upper  = resp + 2 * stderr
      set lower  = resp - 2 * stderr
      graph(ticks,min=minlower,max=maxupper,number=0) 3 i j
      # resp
      # upper / 2
      # lower / 2
  end do j
end do i
spgraph(done)
```

See Also . . .

- **GCONTOUR** Generates high-resolution contour plots.
- **GRAPH** Generates high-resolution time series graphs.
- **GRPARM** Setting parameters for graphics instructions.
- **GRTEXT** Adds text to a graph or scatter plot.
- **SCATTER** Generates high-resolution X-Y scatter plots.
SStats — Series Statistics

SStats computes the sum, product, mean, maximum, or minimum of a range of values from a series or formula. You supply an expression providing the values on which the computation will be based. See STATISTICS and TABLE for other ways to get statistical information on a number of series.

\[ \text{sstats( options )} \ \text{start end (pairs of) expression} \rightarrow \text{result} \]

Parameters

- **start**  
  Range to use. If you have not set a SMPL, this defaults to the current ALLOCATE range. Unlike many RATS instructions, you will need to use the SMPL option to exclude any observations that would cause the expression to return a missing value.

- **expression**  
  A variable or formula

- **>>result**  
  The variable into which the computed result will be stored.

Options

- **mean**  
- **product**  
- **maximum**  
- **minimum**  

  Use one of these (mutually exclusive) options to select the statistic you want to compute. If you don’t use any of the options, SStats computes the sum.

- **smpl=SMPL series or formula**  
  (User’s Guide, Section 5.2)

  You can supply a series or a formula that can be evaluated across entry numbers. Entries for which the series or formula is zero or “false” will be omitted from the calculations.

Examples

\[ \text{sstats(mean,smpl=lfp) / delta} \rightarrow \text{deltabar} \]

Computes the mean of the series DELTA over the observations were LFP is non-zero, putting the result into DELTABAR.

\[ \text{sstat(smpl=yesvm==0) / 1} \rightarrow \text{nos testvm<.5} \rightarrow \text{nnnos prfitted<.5} \rightarrow \text{prbnos} \]
\[ \text{sstat(smpl=yesvm==1) / 1} \rightarrow \text{yes testvm>.5} \rightarrow \text{nnyes prfitted>.5} \rightarrow \text{prbyes} \]

The first acts only on the entries where YESVM is 0. NOS will be the count of the observations (sum of the value of 1), NNNOS is the number of observations where TESTVM is less than .5 (sum of the logical operator TESTVM<.5) and PRBNOS is the number where PRFITTED is less than .5. The second acts only on entries where YESVM is 1, computing an analogous set of count variables.
This example is taken from Chapter 14 of Stock and Watson. Here, we compute pseudo out-of-sample forecasts and use **SSTATS** to compute the (uncentered) second moments of forecast errors. The **UFOREERRORS** procedure uses a **SSTATS** in a similar way to compute forecast performance statistics.

```plaintext
set fcst_adl = 0
set fcst_const = 0

do t1=1992:12,2002:11
   linreg(noprint) exreturn 1960:1 t1
   # constant exreturn{1} ln_divyield{1}
   prj fcst_adl t1+1 t1+1

   linreg(noprint) exreturn 1960:1 t1
   # constant
   prj fcst_const t1+1 t1+1
end do t1

declare vector rmse(3)
sstats(mean) 1993:1 2002:12 exreturn**2>>rmse(1) $(exreturn-fcst_const)**2>>rmse(2) (exreturn-fcst_adl)**2>>rmse(3)

disp "RMSFE of Zero Return" @25 *## sqrt(rmse(1))
disp "RMSFE of Constant" @25 *## sqrt(rmse(2))
disp "RMSFE of ADL" @25 *## sqrt(rmse(3))
```

This code is taken from the replication program for Gali (1999). It uses **SSTATS** to compute long-run impulse response functions:

```plaintext
impulse(model=varmodel,factor=%identity(2),results=e,steps=nstep)
sstats 1 nstep e(1,1)>>emats(1,1) e(1,2)>>emats(1,2) $ e(2,1)>>emats(2,1) e(2,2)>>emats(2,2)

compute esemat = emats*sigmat*tr(emats)
compute cmats = %decomp(esemat)
compute smat = inv(emats)*cmats
compute sinv = inv(smat)
```

### Variables Defined

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%NOBS</td>
<td>Number of observations (INTEGER)</td>
</tr>
<tr>
<td>%MAXENT</td>
<td>Entry number of maximum value (INTEGER)</td>
</tr>
<tr>
<td>%MINENT</td>
<td>Entry number of minimum value (INTEGER)</td>
</tr>
</tbody>
</table>
STATISTICS — Sample Statistics

`STATISTICS` computes basic sample statistics on a single series. You can use the instruction `TABLE` to print a table of statistical information on a number of series. See also the `SSTATS` instruction.

```
statistics(options) series start end
```

Wizard

In the Univariate Statistics wizard on the Statistics menu, choose Sample Statistics.

Parameters

- **series**: Series to analyze.
- **start** and **end**: Range to use. If you have not set a `SMPL`, this defaults to the defined range of `series`.

Options

- `[print]/noprint`: Use `NOPRINT` to suppress the output.
- `window="Title of window"`: If you use the `WINDOW` option, the output goes to a (read-only) spreadsheet window with the given title rather than being inserted into the output window or file as text.
- `smpl=SMPL series or formula (User’s Guide, Section 5.2)`: You can supply a series or a formula that can be evaluated across entry numbers. Entries for which the series or formula is zero or “false” will be omitted from the calculations, while entries that are non-zero or “true” will be included.
- `[moments]/nomoments`: By default, `STATISTICS` computes the following:
  - sample mean, variance and standard error
  - test for $\mu = 0$
  - skewness
  - (excess) kurtosis
  - Jarque–Bera (1987) normality test

The skewness and kurtosis statistics include a test of the null hypotheses that each is zero (the population values if `series` is i.i.d. Normal.) Jarque–Bera is a test for normality based upon the skewness and kurtosis measures combined. We list the formulas for these later in this section.

If you want only the fractiles and not these moment-based statistics, use the option `NOMOMENTS`. 

---

RATS Reference Manual 439
fractiles/[nofractiles]

If you use the FRACTILES option, STATISTICS computes the maximum, minimum, median and a number of other sample fractiles (1%, 5%, 10%, 25%, 75%, 90%, 95% and 99%).

Notes

RATS has a number of other related instructions which you may find useful. EXTREME computes the maximum and minimum values only. MVSTATS computes mean and variance for a moving window on the series and MVFRACTILE computes fractiles for a moving window.

Output

The following instructions read data from the file HAVERSAMPLE.RAT and compute statistics on two series derived from the data (real GDP growth and trade surplus as a fraction of GDP). The first does the moment statistics, the second fractiles.

calendar(q) 1947:1
allocate 2006:4
open data haversample7.rat
data(format=rats,verbose) / x m gdp gdph
set reltrade = (x-m)/gdp
set gdpgrowth = 400*log(gdph/gdph{1})
stats gdpgrowth
stats(fRACTILES,nomoments) reltrade

<table>
<thead>
<tr>
<th>Statistics on Series GDPGROWTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quarterly Data From 1947:02 To 2006:04</td>
</tr>
<tr>
<td>Observations</td>
</tr>
<tr>
<td>Sample Mean</td>
</tr>
<tr>
<td>Standard Error</td>
</tr>
<tr>
<td>t-Statistic (Mean=0)</td>
</tr>
<tr>
<td>Skewness</td>
</tr>
<tr>
<td>Kurtosis (excess)</td>
</tr>
<tr>
<td>Jarque-Bera</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Statistics on Series RELTRADE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quarterly Data From 1947:01 To 2006:04</td>
</tr>
<tr>
<td>Observations</td>
</tr>
<tr>
<td>Minimum</td>
</tr>
<tr>
<td>01-%ile</td>
</tr>
<tr>
<td>05-%ile</td>
</tr>
<tr>
<td>10-%ile</td>
</tr>
<tr>
<td>25-%ile</td>
</tr>
<tr>
<td>Median</td>
</tr>
<tr>
<td>Maximum</td>
</tr>
<tr>
<td>99-%ile</td>
</tr>
<tr>
<td>95-%ile</td>
</tr>
<tr>
<td>90-%ile</td>
</tr>
<tr>
<td>75-%ile</td>
</tr>
</tbody>
</table>
Technical Information

The skewness and kurtosis formulas and the test statistics based upon them are from Kendall and Stuart (1958).

For the sample $X_1, X_2, X_3, ..., X_N$:

Mean ($\bar{X}$) \[ \frac{1}{N} \sum_{i=1}^{N} X_i \]

Variance ($s^2$) \[ \frac{1}{N-1} \sum_{i=1}^{N} (X_i - \bar{X})^2 \]

Standard Error of Mean \[ \frac{s}{\sqrt{N}} \]

t-statistic for mean=0 \[ \frac{\bar{X} \sqrt{N}}{s} \]

$m_k$(used below) \[ \frac{1}{N} \sum_{i=1}^{N} (X_i - \bar{X})^k \]

Skewness (Sk) \[ \frac{N^2}{(N-1)(N-2)} \frac{m_3}{s^3} \]

Sk=0 statistic \[ z = Sk \sqrt{\frac{(N-1)(N-2)}{6N}} \]

Kurtosis (Ku) \[ \frac{N^2}{(N-1)(N-2)(N-3)} \frac{(N+1)m_4 - 3(N-1)m_2^2}{s^4} \]

Ku=0 statistic \[ z = Ku \sqrt{\frac{(N-1)(N-2)(N-3)}{24N(N+1)}} \]

Jarque-Bera \[ j_b = N \left( \frac{(Ku)^2}{24} + \frac{(Sk)^2}{6} \right) \]

Jarque-Bera test \[ j_b \text{ as a } \chi^2(2) \]
Notes

For accuracy, the calculations are all done as written here, and are not done using (theoretically) equivalent expressions with uncentered moments.

Estimates of the variance can also be obtained from many other instructions, such as **CORRELATE**, **VCV**, **CMOMENT**. Those estimates will be different from those produced by **STATISTICS** as only **STATISTICS** uses an \(N-1\) divisor. The estimates from **VCV** and **CMOMENT** might show an even greater difference because those use a set of entries common to all series involved, which may not be the same as the range which would be used for each separately.

The RATS formulas for skewness and kurtosis include small-sample corrections which are omitted by many other software packages. As a result, values for these will often be somewhat different from those obtained using other software.

### Variables Defined

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%NOBS</td>
<td>Number of observations (INTEGER)</td>
</tr>
</tbody>
</table>

### Variables Defined (MOMENTS)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%MEAN</td>
<td>Sample mean (REAL)</td>
</tr>
<tr>
<td>%VARIANCE</td>
<td>Sample variance (REAL)</td>
</tr>
<tr>
<td>%CDSTAT</td>
<td>Test statistic for mean=0 (REAL)</td>
</tr>
<tr>
<td>%SIGNIF</td>
<td>Significance level of %CDSTAT (REAL)</td>
</tr>
<tr>
<td>%SKEWNESS</td>
<td>Skewness (REAL)</td>
</tr>
<tr>
<td>%KURTOSIS</td>
<td>Kurtosis (REAL)</td>
</tr>
<tr>
<td>%JBSTAT</td>
<td>Jarque–Bera statistic (REAL)</td>
</tr>
<tr>
<td>%JBSIGNIF</td>
<td>Significance level of %JBSTAT (REAL)</td>
</tr>
</tbody>
</table>

### Variables Defined (FRACTILES)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%FRACT01</td>
<td>1st percentile (REAL)</td>
</tr>
<tr>
<td>%FRACT05</td>
<td>5th percentile (REAL)</td>
</tr>
<tr>
<td>%FRACT10</td>
<td>10th percentile (REAL)</td>
</tr>
<tr>
<td>%FRACT25</td>
<td>25th percentile (REAL)</td>
</tr>
<tr>
<td>%FRACT75</td>
<td>75th percentile (REAL)</td>
</tr>
<tr>
<td>%FRACT90</td>
<td>90th percentile (REAL)</td>
</tr>
<tr>
<td>%FRACT95</td>
<td>95th percentile (REAL)</td>
</tr>
<tr>
<td>%FRACT99</td>
<td>99th percentile (REAL)</td>
</tr>
<tr>
<td>%MINIMUM</td>
<td>minimum value (REAL)</td>
</tr>
<tr>
<td>%MAXIMUM</td>
<td>minimum value (REAL)</td>
</tr>
<tr>
<td>%MEDIAN</td>
<td>median (REAL)</td>
</tr>
</tbody>
</table>

### See Also . . .

**TABLE** Computes a table of statistics on one or more series.

**EXTREMUM** Locates the maximum and minimum values of a series.
**STEPS — Static Forecasts**

*STEPS* generates a series of one-step forecasts. You can also use it to compute residuals or forecast errors for a model. *STEPS* takes all of its data from the actual data series at all steps. This contrasts with *FORECAST*, which (by default) uses the forecasts from the early steps as input for the forecasts for later steps. Thus *STEPS* does static forecasting as opposed to dynamic forecasting.

*There is no difference between forecasts produced by *FORECAST* and *STEPS* if your model has no lagged endogenous variables.* Also, *FORECAST* now has a *STATIC* option for producing static forecasts, so you may find it easier to just use *FORECAST* for both dynamic and static forecasting tasks.

The instruction *PRJ* can compute one-step forecasts for single equation linear models, as those are simply the fitted values from the regression. You can also use *UFORECAST* (with the *STATIC* option) to get static forecasts for a single equation. *THEIL* can be used to compute statistics on forecast errors.

<table>
<thead>
<tr>
<th>steps( options )</th>
<th>equations</th>
</tr>
</thead>
<tbody>
<tr>
<td># equation</td>
<td>forecasts</td>
</tr>
</tbody>
</table>

**Parameters**

- **equations**: Number of equations in the system. You can use a * if you are using the MODEL option.

Two additional parameters (*steps* and *start*) used in older versions have been replaced by the *STEPS*, FROM, and TO options described below. Version 7 will still recognize the older syntax, but we recommend that you use the options in any new programs.

**Supplementary Cards (if MODEL option isn’t used)**

There is one supplementary card per equation.

- **equation**: The equation name or number.
- **forecast**: (Optional) The series for the forecasts of the dependent variable. Use * for this parameter if you don’t want to save the forecasts, but do want to use *newstart* or *errors*.
- **newstart**: Start entry for *forecasts* and/or *errors*. By default, the same as START.
- **errors**: (Optional) The series for the forecast errors or residuals.
Options

**model**=**model name**
This is an alternative way to specify the system of equations and is the only way
to forecast with a set of FRMLs. MODELS are usually created by **GROUP** or **SYSTEM**.

**from**=starting period for the forecast interval

**to**=ending period for the forecast interval

**steps**=number of forecast periods to compute
These determine the periods for which forecasts will be computed. If you have set
a **SMPL**, these default to forecast over that range. Otherwise, FROM and TO default
to the beginning and end of the most recent estimation range, respectively. If you
want something other than the defaults, you can use:

- FROM and TO to set the starting and ending periods for the forecasts, or
- FROM and STEPS to set the starting date and number of steps (periods)

**iters**=iteration limit for solution algorithm [50]

**cvcrit**=convergence criterion [.00001]

**damp**=damping factor (1=no damping) [1]
These apply if you are simulating a model which contains FRMLs and not just
EQUATIONS, and thus requires an iterative solution. See Section 11.3 of the **User’s Guide**.

**results**=**RECTANGULAR**[SERIES] for result series
This provides a **RECTANGULAR** array of **SERIES** which will be filled with the
results. This has two rows. RESULTS=STEPSFORE will make STEPSFORE(1,i)
the series of one-step forecasts for the ith equation in the model, and
STEPSFORE(2,i) the series of one-step forecast errors.

**print**/[**noprint**]
If PRINT, RATS prints the forecasts and actual values over the forecast period.

**window**="**Title of window**"
If you use the WINDOW option, a (read-only) spreadsheet window is created with
the indicated title and displayed on the screen. This will show the forecasts in
columns below the name of the dependent variable.
Example

```r
smpl 1990:1 2006:4
canmodel
steps(model=canmodel,print,results=stepsfore)
do i=1,6
    graph(header=$
    "Forecasts of "+%l(%modeldepvars(canmodel)(i))) 2
    # stepsfore(1,i)
    # %modeldepvars(canmodel)(i)
end do i
```

This computes and graphs one-step forecast errors over the period 1990:1 to 2006:4 for a six variable model. The forecast graphs include the actual data for comparison. Note that this type of graph is likely to be very uninteresting if plotted over a long span of time, as the forecast errors are generally quite small compared to the overall range of the series. As a result, the actuals and forecasts may be almost on top of each other.
STORE — Saving Data on a RATS Data File

*STORE* adds the listed series to the RATS data file you are currently editing. If a listed series is already present on the file, *STORE* will replace the existing data. *STORE* is easier to use than *INCLUDE* if you have many series to save, but does not allow you to attach comments to the series. The instruction *COPY (FORMAT=RATS)* can be used as an alternative to *STORE* when you are creating a new RATS format file from data in working memory.

*STORE* also has an option *CONVERT* which adds series from other types of data files to the open RATS format file.

```
store( options )  list of series
```

**Wizard**

If you open the RATS data file with *Open RATSData* from the *File* menu and do *Show Series Window* on the *Data* menu, you can add series to the RATS file by dragging them from the Series Window onto the RATSData window.

**Parameters**

*list of series*  RATS stores the listed series on the data file. If you have not set a *SMPL*, *STORE* will use the defined range of each individual series. Omit this list if you use the *CONVERT* option.

**Options for File Conversions**

*convert=rats/portable/wks/dif/prn/xls/dbf/odbc/cdf/tsd*

*CONVERT* indicates the format of the source file for the data you are converting. The files must be organized as described in Chapter 2 of the *User’s Guide*. Omit the *list of series* when using this option.

*refresh/* [norefresh]  *CONVERT* normally copies all of the series on the source file. With *REFRESH*, RATS updates only those series which already exist on the file being edited.

*unit=[data]/input/other unit*

This specifies the source I/O unit for the data you are converting. RATS will read the series *from* this unit, and store them on the RATS format file. Usually, this will just be the default setting of UNIT=DATA.

*org=[rows]/columns*

For *XLS*, *WKS*, *DIF* and *PRN* only. *ORG=COLS* is for a worksheet with the series running down columns, while *ORG=ROWS* is for a worksheet with series running across rows. Note: The older *VARIABLES* and *OBSERVATION* arguments for this option are still supported.
**sheet="worksheet name"**

When converting data from an Excel workbook, you can use the SHEET option to provide the name of the particular worksheet from which you want to read data. By default, RATS will use the first sheet in the file.

**sql="SQL string"**

Used with FORMAT=ODBC, this allows you to connect to a database and read data using SQL commands. See Chapter 2 for more details.

**[adddates]/noadddates**

For XLS, DIF, PRN, WKS, ODBC and DBF only. If there are dates on the file, RATS will transfer them to the RATS format file. If there are none, however, you can still attach dates to the series by using a CALENDAR instruction (before the STORE). The CALENDAR tells RATS the date of the first observation on the file and the data frequency. NOADDDATES is only necessary in the rare instance that you have used a CALENDAR instruction, but want to save the series as undated.

**missing=code (numeric) for missing values**

When you use the MISSING option, any observations of the source data which are equal to code (-999, for example) are stored as missing values on the RATS format file.

**like=template string for series to copy**

**skip=template string for series to omit**

These allow you to selectively include or exclude series from the conversion operation based on series name templates. These support the standard filename wildcard characters "*" and "?". For example, LIKE="X*" will convert all series whose name begins with X, while LIKE="X?" converts only those series whose names are two letters long, with the first letter being X. With SKIP, all series except those matching the SKIP template are copied.

### Usage

To use STORE, you must first open a RATS format data file for editing using DEDIT. Be sure to do a SAVE after the STORE to put the changes out to the file. To use STORE with CONVERT, you also need to open the source file for the data.

### Examples

```plaintext
cal(q) 1947:1
all 2006:4
open data country.dat
data / real_gnp deflator
set nom_gnp = real_gnp*deflator/100
dedit(new) country.rat
store real_gnp deflator nom_gnp
save
quit
```
reads REAL_GNP and DEFLATOR (the GNP deflator) from a free-format file, creates a nominal GNP series (NOM_GNP), and stores all three series on the file COUNTRY.RAT.

```
open mydata mydata.dat
dedit(new) mydata.rat
store(unit=mydata,convert=portable)
save
```

This is a complete RATS program to convert a file (in this case a PORTABLE file) to RATS format. This opens the source file using a user-defined unit name (MYDATA).

```
open data division.xls
dedit division.rat
calendar(m) 1980:1
store(convert=xls,org=obs,refresh)
save
```

updates series stored on the RATS file DIVISION.RAT with newer versions from a spreadsheet file.

```
open data basics.xls
dedit(new) basics.rat
store(convert=xls,org=columns,like="m*")
save
prtdata
```

This opens the data file BASICS.XLS, creates a new RATS file called BASICS.RAT, and then copies all series whose names begin with “m” into the RATS file. The SAVE command saves the changes to the new file, and the PRTDATA command displays the contents of the file to verify that the transfer was done properly.

**Notes**

When converting from RATS or PORTABLE files, STORE will extract up to two lines of comments per series, and store them on the RATS file. With the other formats, if you want comment lines, you will have to use RENAME to add them after doing the STORE.

Series names longer than sixteen characters will be trimmed to the first sixteen. For data downloaded from a commercial data base, you may want to do some RENAMEing after you convert the file, since the data base names for series often are very cryptic.

**See Also . . .**

- **UG**, Section 2.5: RATS format data files.
- **DEDIT**: Instruction required to open a RATS format file for editing.
- **INCLUDE**: Adds a single series to a RATS format file.
- **RENAME**: Renames or adds comments to a series on a RATS format file.
- **SAVE**: Instruction required to save changes to a RATS format file.
STWISE — Stepwise Regressions

STWISE (STepWISE) performs stepwise regressions using one of three methods. It only works with ordinary or weighted least squares; you cannot use instrumental variables.

```
stwise( options ) depvar start end residuals
# list of explanatory variables in regression format
```

Wizard

Use the Regressions wizard on the Statistics menu, and choose Stepwise Regression as the technique.

Parameters

- **depvar**: Dependent variable in the regression.
- **start end**: Estimation range. If you have not set a SMPL, this defaults to the maximum range over which all of the variables involved, dependent and explanatory, are defined.
- **residuals**: (Optional) Series for the residuals.

A fifth parameter, *coeffs*, still works but is not recommended. The coefficients are automatically saved in the vector `%BETA`.

Supplementary card

The order of listing is important if you want to force variables (such as the CONSTANT) into the model, because the FORCE option, described later, will only act on the first variables listed on the card.

Please note that no variables are automatically included in each model unless you use the FORCE option.

Options

- **method=[stepwise]/forward/backward**: Sets the method to be used (see description later).

  - **slenter=Threshold significance level for entering the model [.2]**

  SLENTER is used with the forward and stepwise procedures. At each stage, each variable not yet in the model is added on a trial basis. The one with the highest *t*-statistic is added if the significance level of its *t* is less than the threshold, otherwise the procedure is ended.
STWISE

slstay=Threshold significance level for staying in the model [.2]
SLSTAY is used with the backward and stepwise procedures. The t-statistic on
each variable already in the model is examined. If the significance level on the
one with the smallest t-statistic is larger than the threshold value, it is deleted
and the procedure is repeated with the reduced regression.

If you choose METHOD=STEPWISE, STWISE will not allow you to set SLSTAY to a
lower value than SLENTER.

force=number of regressors to include in all models [0]
By default, all variables listed on the supplementary card are treated identically.
Any may be included or omitted from the final regression. To force regressors
into every model, list those variables first on the supplementary card and use the
option FORCE to indicate how many are to be forced in. If some of these variables
are handled using lag fields, count the number of actual regressors. Examples:

stwise(force=2) gnp_82       All models include CONSTANT and TREND
# constant trend x1 x2 x3 ....

stwise brdip                      STWISE determines which dummies
# seasonal{-11 to 0} ... to include

The following options are the same as those for the LINREG instruction:

[print]/noprint
vcv/ [novcv]
smpl=SMPL series or formula (User’s Guide, Section 5.2)
spread=Residual variance series (Section 5.4)
dfc=Degrees of Freedom Correction (Section 5.15)
define=Equation to define (Section 1.1.9)
frml=Formula to define
unravel/[nounravel] (Section 5.11)

Stepwise Methods

STWISE performs stepwise regressions using one of three methods:

Forward selection    Variables are added to the model sequentially until no
variable not yet in the model would, when added, have a t-
statistic with a P-value (significance level) smaller than the
SLENTER threshold value.

Backward selection    Starting from the full set of regressors, variables with the
lowest t-statistics are deleted until all remaining variables
have a P-value smaller that the SLSTAY threshold.
Full stepwise

This is the default method and combines the other two. At each stage in the forward selection procedure, the backward selection algorithm is run to delete variables which now have small $t$-statistics.

Missing Values

Any observation for which any of the variables is missing is omitted for the analysis.

Output

In addition to the standard regression output, STWISE prints the steps taken in arriving at the final model, for instance,

```r
stwise(force=1) yesvm
# constant public1_2 public3_4 public5 private years $
  teacher loginc logproptax
```

produces

```
Stepping In with P = 0.042916 Variable TEACHER
Stepping In with P = 0.018895 Variable LOGINC
Stepping In with P = 0.011499 Variable LOGPROPTAX
Stepping In with P = 0.075135 Variable PUBLIC3_4
Stepping In with P = 0.143873 Variable PRIVATE

Stepwise Regression
Dependent Variable YESVM
Usable Observations 95 Degrees of Freedom 89
Centered R**2 0.209133 R Bar **2 0.164703
Uncentered R**2 0.708628 T x R**2 67.320
Mean of Dependent Variable 0.6315789474
Std Error of Dependent Variable 0.4849354328
Standard Error of Estimate 0.4432048524
Sum of Squared Residuals 17.482318163
Regression F(5,89) 4.7070
Significance Level of F 0.00073464
Durbin-Watson Statistic 1.918704

<table>
<thead>
<tr>
<th>Variable</th>
<th>Coeff</th>
<th>Std Error</th>
<th>T-Stat</th>
<th>Signif</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Constant</td>
<td>-0.616413702</td>
<td>1.285473681</td>
<td>-0.47952</td>
<td>0.63274314</td>
</tr>
<tr>
<td>2. PUBLIC3_4</td>
<td>0.161132508</td>
<td>0.102510118</td>
<td>1.57187</td>
<td>0.11952936</td>
</tr>
<tr>
<td>3. PRIVATE</td>
<td>-0.223345666</td>
<td>0.151471178</td>
<td>-1.47451</td>
<td>0.14387316</td>
</tr>
<tr>
<td>4. TEACHER</td>
<td>0.282827421</td>
<td>0.150827272</td>
<td>1.87517</td>
<td>0.06404781</td>
</tr>
<tr>
<td>5. LOGINC</td>
<td>0.458872777</td>
<td>0.130462014</td>
<td>3.51729</td>
<td>0.00068869</td>
</tr>
<tr>
<td>6. LOGPROPTAX</td>
<td>-0.488016288</td>
<td>0.169741839</td>
<td>-2.87505</td>
<td>0.0050310</td>
</tr>
</tbody>
</table>
```
Notes

You can use DEFINE or FRML to save the form of the estimated regression as an EQUATION or FRML. If, however, your program needs to examine the choices made for the regressors, you can “fetch” the final regression by using the functions %EQNTABLE(0), %EQNCOEFFS(0) and %EQNSIZE(0). %EQNSIZE(0) is the number of regressors, %EQNCOEFFS(0) is the coefficient vector and %EQNTABLE(0) is a 2 x regressors INTEGER array, where the first row elements are the series numbers of the chosen regressors and the second row are their lags. If we do the following after the STWISE instruction executed above,

dec vect[labels] vlabel(%eqnsize(0))
ewise vlabel(i) = %l(%eqntable(0)(1,i))
display vlabel

the following will be displayed:

Constant PUBLIC3_4 PRIVATE TEACHER LOGINC LOGPROPTAX

Variables Defined by STWISE

STWISE defines most of the variables defined by LINREG.

Fit Statistics
%MEAN, %RBARSQ, %RSQUARED, %RSS, %SEESQ,
%TRSQ, %TRSQUARED, %VARIANCE

Serial Correlation Statistics
%DURBIN, %RHO

Count Variables
%NDF, %NOBS, %NREG

Vectors, Matrices, and Series
%BETA, %STDERRS, %TSTATS, %RESIDS, %XX
**SUMMARIZE — Sums of Coefficients**

**SUMMARIZE** has two functions: It can compute the sum of the coefficients for the listed variables and print the sum, its standard error, the \( t \)-statistic (for a test that the sum is zero) and the significance level of the \( t \)-statistic. It can also be used to analyze any combination of the coefficients of the regression.

\[
\text{summarize(options)}
\]

#  list of variables in regression format (omit with \text{VECTOR} or \text{ALL} options)

\[
\text{summarize(options)} \text{ expression}
\]

**Parameter**

\textit{expression}  
To analyze a linear or nonlinear combination of the coefficients, supply the desired expression as a parameter, using the \%BETA vector to represent the coefficients. Omit the supplementary card.

**Supplementary Card**

On the supplementary card, list in regression format the set of variables from the previous regression that you want to sum. Omit this if you use the \text{VECTOR} or \text{ALL} options, or if you are using the \textit{expression} parameter to analyze your own function of the coefficients.

**Example**

The most common use of **SUMMARIZE** is the analysis of a set of coefficients in a distributed lag. For instance, this **SUMMARIZE** computes the sum of the lags of \texttt{SHORTRATE}:

\begin{verbatim}
linreg longrate  
# constant shortrate{0 to 24}  
summarize  
# shortrate{0 to 24}
\end{verbatim}

**Output**

The example above produces the following output:

<table>
<thead>
<tr>
<th>Summary of Linear Combination of Coefficients</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SHORTRATE</td>
<td>Lag(s) 0 to 24</td>
</tr>
<tr>
<td>Value</td>
<td>0.97229804</td>
</tr>
<tr>
<td>Standard Error</td>
<td>0.01007179</td>
</tr>
</tbody>
</table>
**Summarize**

**Applicability**

The **SUMMARIZE** command, in the form described above, can be used only after **LINREG, STWISE, AR1, DDV** (logit and probit), **LDV, SUR** or **ITERATE**, as those instructions use regressor lists either directly or indirectly. It can be used after the estimation instructions which use parameter sets (such as **MAXIMIZE**), but only with the use of the **VECTOR** option or the expression using the %BETA vector.

**Options**

- **[print]/noprint**
  - If you are just using **SUMMARIZE** to compute its variables, and don’t need the printed output, use **NOPRINT**.

- **all/[noall]**
  - Use **ALL** to test the sum of all of the coefficients. Omit the supplementary card when using either of these options.

- **vector=vector of weights**
  - **VECTOR** allows you to compute any linear combination of coefficients, not just the sum. The **vector of weights** is a **VECTOR** with dimension equal to the number of regressors. It gives the weights to be applied to the regressors in computing the linear combination.

- **title="string for output title"**
  - You can use the **TITLE** option to include information in the output to identify what is being computed.

- **form=f/chisquared**
  - This determines the form of the test statistic used. By default, RATS will select the appropriate form based upon the estimation technique used last. You can use **FORM** to manually select a distribution if you have made changes to the regression that require a different distribution, such as altering the %XX matrix in a way which incorporates the residual variance into %XX. See Section 5.13 in the **User’s Guide**.

- **value=input value for function [not used]**
  - As an alternative to using the **expression** parameter to compute a standard error of a non-linear function of the coefficients, you can use the **VECTOR** option to input the weights on the function, and the **VALUE** option to input the value of that function.
**Technical Information**

If the preceding estimation instruction produces $\hat{\beta}$ as the estimator, with covariance matrix

1. $\hat{\sigma}^2 (X'X)^{-1}$

then the linear combination $c\hat{\beta}$ will have variance

2. $\hat{\sigma}^2 c(X'X)^{-1}c'$

The test statistic for $c\beta = 0$

3. $\left(c\hat{\beta}\right)/\hat{\sigma}\sqrt{c(X'X)^{-1}c'}$

will be treated as a $t$ with degrees of freedom from the last regression.

If, instead, the covariance matrix is $\Sigma_X$, that is, a factor of $\hat{\sigma}^2$ can't be pulled out of it, then the test statistic of

4. $\left(c\hat{\beta}\right)/\sqrt{c\Sigma_X c'}$

is treated as a standard Normal.

**Examples with VECTOR option**

This is from example 6.2 in Greene (2003). It estimates a translog production function, and computes the elasticity of output with respect to capital at the mean values for the logs of the two inputs. The linear combination is:

1 $\times \left(\text{LOGK coefficient}\right) + \left(\text{mean of LOG}\text{K}\right) \times \left(\text{LOGK2 coefficient}\right)$

+ $\left(\text{mean LOGL}\right) \times \left(\text{LOGLK coefficient}\right)$

```
linreg logq
# constant logl logk logl2 logk2 loglk
sstats(mean) / logk>>logkmean logl>>loglmean
summarize(title='Estimate of K elasticity',$
vector=||0.0,0.0,1.0,0.0,logkmean,loglmean||)
```
The example below demonstrates a technique commonly referred to as the “delta method”. It computes a non-linear function from a linear regression and prints the value and the estimated standard errors from a linearization. This example estimates the long-run marginal propensity to consume from the regression

\[ C_t = b_1 + b_2 Y_t + b_3 C_{t-1} \]

The non-linear function required is \( \frac{b_2}{1-b_3} \). This is from example 6.3 in Greene (2003).

```
linreg(vcv) logc
# constant logy logc{1}
summarize(title="Long-run MPC") %beta(2)/(1-%beta(3))
```

**Variables Defined by SUMMARIZE**

- `%CDSTAT` the test statistic (for zero) (REAL).
- `%SIGNIF` the marginal significance level of `%CDSTAT` (REAL).
- `%SUMLC` SUM of the Linear Combination (REAL).
- `%VARLC` VARIANCE of Linear Combination (REAL).
SUR — Linear Systems Estimation

SUR computes estimates of a system of linear equations using the technique of joint GLS, or three-stage least squares (for instrumental variables). SUR (Seemingly Unrelated Regressions) is a bit of a misnomer because this instruction includes options for cross-equation restrictions.

See Section 5.8 of the User’s Guide for technical information concerning the techniques. The instruction NLSYSTEM can be used for estimating systems of non-linear equations. Anything you can do with SUR, you can also do with NLSYSTEM, but if SUR is capable of handling the problem, it is usually easier to set up and quite a bit faster.

\[
\text{sur( options ) equations start end equate list}
\]

# equation resids coeffs (one card per equation, omit with MODEL option)

**Parameters**

- **equations**
  - Number of equations in the system you want to estimate.
- **start end**
  - Estimation range. If you have not set a SMPL, this defaults to the common defined range of all variables involved in the complete regression.
- **equate**
  - This is just the word EQUATE. Omit this and the list if you want the standard estimation without cross-equation restrictions. The differences between SUR with and without EQUATE are described later in this section.
- **list**
  - (Optional) List of coefficient positions to be equated across equations. If you use EQUATE without list, all equations have the same coefficient vector.

**Supplementary Cards**

Supply one supplementary card for each equation in the system being estimated. Omit the supplementary cards if using the MODEL option to supply the model.

- **equation**
  - The equation name or number. You must set up the equations in advance using EQUATION or LINREG with DEFINE.
- **residuals**
  - (Optional) The series for the residuals for equation. Use * if you want to use coeffs but not residuals.
- **coeffs**
  - (Optional) The series of the coefficient estimates for equation.

SUR also offers RESIDS and COEFFS options for storing the residuals and coefficients.
General Options

\texttt{[print]/noprint}
\texttt{vcv/[novcv]}
\texttt{[sigma]/nosigma}

These control the printing of the regression output, the covariance matrix of the complete coefficient vector, and the final estimate of the residual covariance matrix, respectively.

\texttt{model=\textit{model name}}

This is an alternative way to specify the system of equations to be estimated. MODELs are usually created by \texttt{GROUP} or \texttt{SYSTEM}. For \texttt{SUR}, the MODEL must contain only EQUATIONS, and no FRMLs.

\texttt{resids=VECTOR[SERIES] for the residuals [unused]}
\texttt{coeffs=RECTANGULAR array of the coefficients [unused]}

These allow you to save the estimated residuals and coefficients. Residuals are saved into a VECTOR of SERIES, with element \textit{n} of the array containing the residuals for equation \textit{n}. Coefficients are saved into a RECTANGULAR array, with the coefficients for equation \textit{n} stored in column \textit{n}.

\texttt{smpl=SMPL series or formula (User's Guide, Section 5.2)}
\texttt{spread=Series of residual variances (Section 5.4)}
\texttt{dfc=Degrees of Freedom Correction (Section 5.15)}

These options are the same as for \texttt{LINREG}, but the single option applies to all equations. The \texttt{SPREAD} option implies a covariance matrix of disturbances of the form $\Sigma \otimes D$, with $D$ diagonal.

\texttt{iterations=Iteration Limit [0]}
\texttt{cvcrit=Convergence criterion [.00001]}
\texttt{trace/[notrace]}

If you give \texttt{ITERATIONS} a non-zero value, RATS will iterate on the estimation process until either it reaches the iteration limit or until the largest change in any coefficient value is less than the \textit{Convergence criterion.} TRACE prints the intermediate results.

\texttt{cv=Input $\Sigma$ matrix (SYMMETRIC)}
\texttt{cvout=Output $\Sigma$ matrix (SYMMETRIC)}

\texttt{CV} allows you to feed in an initial covariance matrix ($\Sigma$) and \texttt{CVOUT} allows you to save the final estimate of the covariance matrix. For \texttt{CVOUT}, you don’t need to \texttt{DECLARE} or \texttt{DIMENSION} the array. The final $\Sigma$ matrix is also stored automatically in the reserved variable %SIGMA.

When you use \texttt{CV}, the standard errors and covariance matrix of coefficients will be correct only if the \texttt{CV} matrix incorporates the residual variances. For instance, you can obtain two-stage least squares estimates of the coefficients of a system of
equations using $\text{SUR}(\text{INST})$ with an $\text{CV}$ of the identity matrix, but the covariance matrix will be incorrect. Example 6.7 in the *User’s Guide* uses both of these options and shows how to obtain a correct full system covariance matrix for 2SLS, taking into account the cross equation residual covariances.

These options were previously called $\text{ISIGMA}$ and $\text{OUTSIGMA}$, respectively. The old option names are still recognized in Version 7.

**cmom/ [nocmom]**
This pulls cross products out of the cross product matrix computed previously with a $\text{CMOM}$ instruction. This can improve calculation time if the $\text{SUR}$ is being executed many times with different input $\text{CV}$ matrices.

**create/ [nocreate]**
**setup/ [nosetup]**
Use $\text{CREATE}$ to print the output from the system if you recompute the coefficients and/or covariance matrix using an instruction other than $\text{SUR}$. This is the system's analogue of the $\text{CREATE}$ option for $\text{LINREG}$. $\text{SETUP}$ does no estimation: it sets up the $\%\text{BETA}$ and $\%\text{XX}$ arrays (described below) so that you can compute the coefficients and covariance matrix using matrix instructions. You can then use $\text{SUR(CREATE...)}$ to get the output.

**unravel/ [nounravel]**
Substitutes for $\text{ENCODED}$ variables (*User’s Guide*, Section 5.11). RATS does not print the intermediate regression (in terms of encoded variables).

**title="title for output" [“Seemingly Unrelated Regressions”]**
This option allows you to supply your own title to label the resulting output.

**robusterrors/ [norobusterrors]**
**lags=** correlated lags [0]
**lwindow=neweywest/bartlett/damped/parzen/quadratic/[flat]/panel/white**
**damp=value of $\gamma$ for lwindow=damped [0.0]**
**lwform=VECTOR with the window form [not used]**
**cluster=SERIES for clustered std. errors [not used]**

When you use these *without* the $\text{INSTRUMENTS}$ option, they allow you to calculate a consistent covariance matrix allowing for heteroscedasticity (with $\text{ROBUSTERRORS}$), serial correlation (with $\text{ROBUSTERRORS}$ and $\text{LAGS}$), or clustered standard errors (with $\text{ROBUSTERRORS}$ and $\text{CLUSTER}$). See Sections 5.3, 5.4, 5.5, and 7.5 of the *User’s Guide* and the description of the instruction $\text{MCOV}$ for more information.

None of these affect the parameter estimates. As with $\text{LINREG}$ and $\text{NLLS}$, they only come into play when the covariance matrix of the estimates is computed. They behave differently when used with $\text{INSTRUMENTS}$. 

RATS Reference Manual 459
Instrumental Variables Options

\texttt{instruments/[noinstruments]}

Use the \texttt{INSTRUMENTS} option to do three-stage least squares or GMM. You must construct the instruments list in advance using the instruction \texttt{INSTRUMENTS}.

\texttt{zudependent/[nozudep]}

\texttt{wmatrix=} SYMMETRIC weighting matrix for instruments $[(Z'Z)^{-1}]$

\texttt{sw=} SYMMETRIC grand weighting matrix [not used]

\texttt{swout=} estimated SYMMETRIC grand weighting matrix [not used]

\texttt{NOZUDEP} (the default) is the special case for the \texttt{SW} matrix. We call this \texttt{NOZUDEP} because the most important case is where $u$ is (serially uncorrelated and) independent of the instruments $Z$. More generally, this is Case (i) in Hansen (1982, page 1043).

With \texttt{NOZUDEP}, you can use the \texttt{WMATRIX} option to set the $W$ part of the $\Sigma^{-1} \otimes W$ and the \texttt{CV} option to set $\Sigma$. Otherwise, \texttt{SUR} estimates a new $\Sigma$ after each iteration.

If \texttt{ZUDEP}, you can use the \texttt{SW} option to set the full $SW$ array. This is an $nr \times nr$ SYMMETRIC array. Otherwise, \texttt{SUR} determines a new $SW$ matrix after each iteration by taking the inverse of

\begin{equation}
\frac{1}{T} \sum \left( u_i \otimes Z_i \right) \left( u_i \otimes Z_i \right)'
\end{equation}

(or the generalization of this if you use the \texttt{LAGS} option). The \texttt{SWOUT} option allows you to save the estimated $SW$ matrix into the specified array.

\texttt{zumean=} VECTOR of means of moment conditions [all zero]

This allows you to supply a VECTOR (with dimensions equal to the number of instruments times the number of equations) with a set of known (non-zero) means for $E(u \otimes Z)$. By default, these are zero.

\texttt{center/[nocenter]}

\texttt{CENTER} adjusts the weight matrix formula to subtract off the (sample) means of $u \otimes Z$, which may be non-zero for an overidentified model. See the description of \texttt{MCOV} for more information.

\texttt{update=} none/once/continuous [default depends on other options]

This controls the updating of the weighting matrix. The default is normally \texttt{UPDATE=CONTINUOUS}, which recalculates the weight matrix at each iteration, except in the following cases:

- If you use the \texttt{SW} option, the default is \texttt{UPDATE=NONE}.
- If you use the \texttt{CV} option with \texttt{NOZUDEP}, the default is \texttt{UPDATE=ONCE}.
lags=correlated lags [0]
lwindow=neweywest/bartlett/damped/parzen/quadratic/[flat]/panel/white

damp=value of $\gamma$ for lwindow=damped [0.0]
lwform=VECTOR with the window form [not used]
cluster=SERIES for clustered serial correlation [not used]

Use LAGS or LWINDOW=PANEL or the CLUSTER option to handle serial correlation in the $u \otimes Z$ process. See Section 5.3 of the User’s Guide for more information. Note that, unlike SUR without INSTRUMENTS, these options do affect the estimated coefficients by changing the weight matrix.

robusterrors/[norobusterrors]
If you use ROBUSTERRORS combined with an input CV or SW matrix, SUR will compute the coefficients using the “suboptimal” weighting matrix and then correct the covariance matrix of the coefficients based upon the choices for the LAGS, LWINDOW and other options immediately above.

jrobust=statistic/[distribution]
You can use this option to adjust the $J$-statistic specification test when the weighting matrix used is not the optimal one. See Section 7.8 in the User’s Guide for more information.

Variables Defined by SUR
Because SUR estimates a whole set of equations, most of the single equation fit statistics aren’t defined. The matrices %XX, %BETA, %STDERRS and %TSTATS are all defined for the full set of coefficients and %NREG is the count of the number of regressors in the full system. %NOBS is the number of observations, %NVAR is the total number of variables, and the log likelihood is stored in %LOGL. There are two special variables which are defined by SUR and not by LINREG:

%LOGDET log determinant of the estimate of $\Sigma$ (REAL)
%SIGMA final estimate of $\Sigma$

Output
The next page shows the output from a two equation SUR. Note that SUR prints the regression output (controlled by [PRINT]/NOPRINT) separately for each equation. However, the covariance/correlation matrix of the coefficient estimates (controlled by VCV/[NOVCV]) is for the full system. The final part of the output is the covariance/correlation matrix of the residuals (controlled by [SIGMA]/NOSIGMA). RATS labels the rows and columns with the names of the dependent variables of the equations.
### Linear Systems - Estimation by Seemingly Unrelated Regressions

**Iterations Taken:** 2  
**Annual Data From:** 1935:01 To 1954:01  
**Usable Observations:** 20

**Dependent Variable IGE**

- **Centered R**\(^2\): 0.692557  
- **R Bar**\(^2\): 0.746026  
- **Uncentered R**\(^2\): 0.945739  
- **T x R**\(^2\): 18.915

**Mean of Dependent Variable:** 102.29000000  
**Std Error of Dependent Variable:** 48.58449937

**Centered R**\(^2\): 0.740401  
**R Bar**\(^2\): 0.785549  
**Uncentered R**\(^2\): 0.958811  
**T x R**\(^2\): 19.176

**Mean of Dependent Variable:** 42.891500000  
**Std Error of Dependent Variable:** 19.110188596  
**Standard Error of Estimate:** 8.849713526

**Sum of Squared Residuals:** 13788.375833  
**Durbin-Watson Statistic:** 0.985603

**Variable** | **Coeff** | **Std Error** | **T-Stat** | **Signif**
---|---|---|---|---
1. Constant | -27.71931712 | 27.03282800 | -1.02539 | 0.30517701
2. FGE | 0.03831021 | 0.01329011 | 2.88261 | 0.00394396
3. CGE | 0.13903627 | 0.02303559 | 6.03572 | 0.00000000

**Variable** | **Coeff** | **Std Error** | **T-Stat** | **Signif**
---|---|---|---|---
4. Constant | -1.251988228 | 6.956346688 | -0.17998 | 0.85716997
5. FWEST | 0.057629796 | 0.013411012 | 4.29720 | 0.00001730
6. CWEST | 0.063978067 | 0.048900998 | 1.30832 | 0.19076540

**Covariance/Correlation Matrix of Coefficients**

<table>
<thead>
<tr>
<th></th>
<th>Constant</th>
<th>FGE</th>
<th>CGE</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>730.773789708</td>
<td>-0.9164877100</td>
<td>-0.2348445300</td>
<td>0.6751545251</td>
</tr>
<tr>
<td>FGE</td>
<td>-0.329265961</td>
<td>0.000176627</td>
<td>-0.1112155480</td>
<td>-0.5699084719</td>
</tr>
<tr>
<td>CGE</td>
<td>-0.146241701</td>
<td>-0.000034048</td>
<td>0.000530638</td>
<td>-0.2472566852</td>
</tr>
<tr>
<td>Constant</td>
<td>126.962621780</td>
<td>-0.052688404</td>
<td>-0.039621285</td>
<td>48.390759242</td>
</tr>
<tr>
<td>FWEST</td>
<td>-0.226192982</td>
<td>0.000119993</td>
<td>-0.000016872</td>
<td>-0.080022511</td>
</tr>
<tr>
<td>CWEST</td>
<td>0.392514767</td>
<td>-0.000324800</td>
<td>0.000594829</td>
<td>0.113617804</td>
</tr>
</tbody>
</table>

**Covariance/Correlation Matrix of Residuals**

<table>
<thead>
<tr>
<th></th>
<th>IGE</th>
<th>IWEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>IGE</td>
<td>689.4187916587</td>
<td>0.7650429357</td>
</tr>
<tr>
<td>IWEST</td>
<td>190.6362560894</td>
<td>90.0650439232</td>
</tr>
</tbody>
</table>

---

### Additional Tables and Statistics

- **Variables:** Constant, FGE, CGE, FWEST, CWEST
- **Significance Levels:** 0.05, 0.01, 0.001
- **Model Fit Measures:** R**2, R Bar**2, Uncentered R**2, T x R**2, Mean of Dependent Variable, Std Error of Dependent Variable, Standard Error of Estimate, Sum of Squared Residuals, Durbin-Watson Statistic
- **Variable Coefficients:** Include constant terms and independent variables with their respective coefficients, standard errors, t-statistics, and significance levels.
Restriction Across Equations: SUR with EQUIATE

Applicability
You should only use EQUIATE on a system of equations with similar form:

- each equation should have the same number of explanatory variables.
- corresponding variables should be in the same positions in each equation.

For each position in list, SUR forces the coefficients in all equations at that position to be equal. For example, you would put a 2 in list to equate the 2nd coefficients in all equations.

Example
equation geeq ige
# constant fge cge
equation westeq iwest
# constant fwest cwest
sur 2 / equate 2 3
# geeq
# westeq

This restricts the coefficient in position 2 of the first equation (the FGE coefficient) to be equal to the coefficient in position 2 of the second equation (the FWEST coefficient). It also restricts to be equal the coefficients in position 3: CGE and CWEST.

Output
The output for SUR with EQUIATE is the same as for standard SUR with one exception: the covariance matrix of coefficients does not include duplicates of the equated coefficients. The equated coefficients are listed first, followed by the coefficients which are estimated separately. For the example above:

Covariance\Correlation Matrix of Coefficients

<table>
<thead>
<tr>
<th></th>
<th>FWEST</th>
<th>CWEST</th>
<th>Constant</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>FWEST</td>
<td>0.0000655342</td>
<td>-0.0160284252</td>
<td>-0.8202103255</td>
<td>-0.8753618770</td>
</tr>
<tr>
<td>CWEST</td>
<td>-0.000002975</td>
<td>0.000527399</td>
<td>-0.4716036803</td>
<td>-0.3051918796</td>
</tr>
<tr>
<td>Constant</td>
<td>-0.125659410</td>
<td>-0.205267560</td>
<td>359.208892358</td>
<td>0.9500871042</td>
</tr>
<tr>
<td>Constant</td>
<td>-0.043583788</td>
<td>-0.043170161</td>
<td>110.911878782</td>
<td>37.938678405</td>
</tr>
</tbody>
</table>

Restricted coefficients will take their labels from the last equation. The first CONSTANT is the intercept from equation 1, the second is from equation 2.

The variables %XX, %BETA, %STDERRS, %TSTATS and %NREG are all set up in this order as well. For instance, %NREG will just be four (the number of free coefficients) and %BETA will have four entries.
SWEEP

SWEEP — Sweep Regressions

SWEEP performs a regression of a set of “targets” on a set of “instruments”.

\[ \text{sweep( options ) start end} \]
# list of target variables
# list of instrument variables

Options

\text{smpl=} \text{SMPL series or formula (User's Guide, Section 5.2)}
You can supply a series or a formula that can be evaluated across entry numbers. Entries for which the series or formula is zero or “false” will be skipped, while entries that are non-zero or “true” will be included in the operation.

\text{spread=} \text{Residual variance series (Section 5.4)}
Use SPREAD for weighted least squares. The residual variances are assumed to be proportional to the indicated series.

\text{equation=} \text{EQUATION for target variables}
Omit target supplementary card if you use this option.

\text{model=} \text{MODEL to use for target variables}
This includes all variables from all equations in the MODEL. Omit the target supplementary card if you use this option.

\text{depvar/} [\text{nodepvar}]
Controls whether or not the dependent variable(s) from EQUATION or MODEL are included as target variables.

\text{instruments/} [\text{noinstruments}]
If INSTRUMENTS, use the current list from the INSTRUMENTS instruction rather than a list on a supplementary card. Omit the “instruments” card if you use INSTRUMENTS.

\text{coeffs=} m \times n \text{ matrix of coefficients}
For \( m \) instruments and \( n \) targets, saves the estimated coefficients into an \( m \times n \) array.

\text{series=} \text{VECT[SERIES] for residuals/fitted values}
\text{prj=} [\text{residuals}]/\text{fitted}
For \( n \) targets, SERIES creates an \( n \)-element VECTOR of SERIES containing the series of residuals or fitted values (depending on the choice for PRJ) for each target.
group=SERIES or FRML with values on which to group regressions
variances=[homogeneous]/heterogeneous
average=[count]/simple/precision

If you use GROUP, a separate regression will be run for each unique value in the series or formula. By default (i.e. if GROUP is not used), RATS does one regression across the entire sample.

The VARIANCES option indicates whether the variances are HOMOGENEOUS (same across groups) or HETEROGENEOUS (different).

The AVERAGE option indicates how the coefficient vectors for the different groups are combined. COUNT weights them by the size of the group, SIMPLE weights each group equally, PRECISION weights by the size of the group.

cvout=Output \( \Sigma \) matrix (SYMMETRIC)
CVOUT allows you to store the final estimate of the covariance matrix. You don’t need to DECLARE or DIMENSION the array ahead of time.

Examples

sweep(group=%indiv(t),var=hetero)
# dc
# lpc{1} constant lndi dp dy ddp

computes into %BETA the average regression coefficients from regressing DC on the variables in the second list, with a separate regression on each individual in a panel data set, allowing the variances to differ from individual to individual.

set group = 1+(z{1}>=g1t)+(z{1}>=g2t)
sweep(grouping=group)
# df ds
# constant df{1 to 8} ds{1 to 8} z{1}

does a bivariate systems regression of DF and DS on the second list, with regressions done on three subsamples using a common variance matrix (since VARIANCES is the default HOMOGENEOUS).

Variables Defined

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%BETA</td>
<td>Averaged vector of regression coefficients</td>
</tr>
<tr>
<td>%XX</td>
<td>Covariance matrix of averaged regression coefficients</td>
</tr>
<tr>
<td>%SIGMA</td>
<td>Covariance matrix of residuals</td>
</tr>
<tr>
<td>%NREG</td>
<td>Total number of regressors across groups</td>
</tr>
<tr>
<td>%NREGSYSTEM</td>
<td>Total number of regressors across groups and targets</td>
</tr>
<tr>
<td>%LOGL</td>
<td>Log likelihood</td>
</tr>
</tbody>
</table>
SYSTEM — Setting Up a VAR System

`SYSTEM` is the first of a set of instructions which create a vector autoregressive (VAR) system of equations. You also need to use `SYSTEM` when setting up an equation or set of equations for Kalman filtering. See Chapter 10 of the *User’s Guide* for more information about Vector Autoregressions.

```
    system(options)  list of equations in the system
```

Wizard

The `VAR (Setup/Estimate)` wizard, located on the *Statistics* menu, provides an easy, dialog-driven interface for defining and estimating VAR models.

Parameters

`equations` The list of equations you want to include in the system. Omit this if you use the `MODEL` option. For a VAR, it is usually most convenient to use numbers for these and to let `VARIABLES`, `LAGS` and `DETERMINISTIC` create the equations. If you are grouping some equations for Kalman filtering, use `EQUATION` or `LINREG DEFINE=xxx` to define them before using `SYSTEM`.

Options

`model=model name [unused]`

The `MODEL` option defines the equations in the system as a `MODEL` variable. If you use `MODEL`, omit the list of equations.

The `MODEL` option makes using instructions such as `FORECAST`, `IMPULSE`, and `HISTORY` easier—rather than having to list each equation on a supplementary card, you just use the `MODEL` option on those instructions to reference the system. If you are going to use the `ECT` instruction to add error-correction terms to your model, you *must* use `MODEL`.

`cmoment/[nocmoment]`

Use the `CMOMENT` option when you are putting together a system of very similar but not identical equations. It will reduce the computation time in estimating the systems.

Description

A system is specified beginning with `SYSTEM`, ending with `END SYSTEM` and (optionally) including the following instructions:

```
    VARIABLES, LAGS, DETERMINISTIC
    list the dependent variables, lags and exogenous variables which form a VAR.
```
**SPECIFY**
specifies the prior distribution for a Bayesian VAR (BVAR).

**ECT**
lists equations which describe error-correction terms

**KFSET, TVARYING**
set various options for standard and time-varying coefficients applications of the Kalman filter.

**Examples**

```plaintext
system 1 to 4
variables shortrat longrate m1 twdollar
lags 1 2 3 4 6 9 12
det constant
end(system)
```

creates a four variable VAR, represented by equations 1 to 4. Using the **MODEL** option, this would be:

```plaintext
system(model=ratemod)
variables shortrat longrate m1 twdollar
lags 1 2 3 4 6 9 12
det constant
end(system)
```

```plaintext
linreg(define=olseqn) depvar
# constant x1 x2 x3
system olseqn
end(system)
```

estimates a regression with **LINREG**, defining it as equation **OLSEQN**. This equation then goes into a one-equation system, which can be estimated sequentially using the Kalman filter.

```plaintext
equation(coeffs=||1.0,1.0||) pppeq logpx
# logpy logexr
system(model=cointmodel)
variables logpx logpy logexr
lags 1 to 6
det constant
ect pppeq
end(system)
```

creates a cointegrated system of three variables with the cointegrating relationship

$$\log P_x = \log P_y + \log X$$

where $P_x$ and $P_y$ are price indices and $X$ is the exchange rate linking the countries’ currencies.
TABLE — Basic Statistics on Series

**TABLE** prints out a table of basic statistical information about a set of series. The statistics computed are the number of observations, sample mean, sample standard error, minimum and maximum. It also computes the overall maximum and minimum values of the whole set of series.

```
  table(options)  start   end   list of series
```

**Wizards**

If you use the *Show Series Window* wizard on the *Data* menu, you can get a statistics table by selecting the desired series and then using the *Statistics* toolbar icon:

**Parameters**

- **start**  **end**  Range of entries to use in computing statistics. If you have not set a **SMPL**, **TABLE** uses all available data for each series, determined separately.

- **list of series**  The list of series to include in the table. If you omit the list, *all* current series are included.

**Options**

- **smpl=SMPL series or formula**  *(User's Guide, Section 5.2)*

  You can supply a series or a formula that can be evaluated across entry numbers. Entries for which the series or formula is zero or “false” will be omitted from the computation, while entries that are non-zero or “true” will be included.

- **window=”Title of window”**

  If you use the **WINDOW** option, the output goes to a (read-only) spreadsheet window with the given title, rather than being inserted into the output window or file as text. From this window, you can easily export data (using *File–Export...*) to a spreadsheet or word processing program to prepare a table for publication.

- **[print]/noprint**

  Use **NOPRINT** if you want to suppress the displaying of the output on the screen.

- **picture=picture clause for data**

  You can use **PICTURE** to control the formatting of the numbers in the output table. A picture clause takes a form like "###.###" or "*.#". The first requests two digits left of the decimal and three digits to the right. The second asks for one digit right and as many digits as are needed to the left. See **DISPLAY** for more on picture codes.
matrix=Kx5 RECTANGULAR array of the statistics [unused]

Use the MATRIX option to store the computed results into a $K \times 5$ array, where $K$ is the number of variables listed. For each series, TABLE stores the number of observations in column 1, the mean in column 2, the standard error in column 3, and the minimum and maximum values in columns 4 and 5 respectively.

title="title for output" [none]

This option allows you to supply your own title to label the resulting output.

Notes

It’s always a good idea to use a TABLE instruction immediately after the DATA instruction, particularly the first time you use a data set. It gives you a quick way to check whether your data are in good shape. For instance, you can easily detect series which have missing data because the number of observations will not match with those for other series. You might also have a series whose data range doesn’t match with what you expect. TABLE can help you spot series which have problems, but you will have to use PRINT or some other instruction to isolate the cause.

The sample standard error is computed using an $N-1$ divisor where $N$ is the number of data points in a series.

%MAXIMUM and %MINIMUM are the only variables that TABLE defines that you can access from within your RATS program. Use the STATISTICS instruction applied to the series one at a time if you need access to such things as the mean and number of observations.

Examples

```
open data states.wks
data(org=obs,format=wks) 1 50 expend pcaid pop pcinc
set pcexp = expend/pop
table
```

produces the table of statistics in the sample output for the four series read from the data file plus the created series PCEXP.

```
table(noprint) / canusxsr frausxsr jpnusxsr gbrusxsr
dofor i = canusxsr frausxsr jpnusxsr gbrusxsr
  graph(max=%maximum,min=%minimum,header=%1(i)) 1
  # i
end dofor
```

uses TABLE to find the maximum and minimum values (%MAXIMUM and %MINIMUM) for the group of four series so they can be graphed on a common scale.
Variables Defined by TABLE

- **%MAXIMUM**: maximum value found in the list of series (real).
- **%MINIMUM**: minimum value found in the list of series (real).

Sample Output

<table>
<thead>
<tr>
<th>Series</th>
<th>Obs</th>
<th>Mean</th>
<th>Std Error</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPEND</td>
<td>50</td>
<td>3316.1600000</td>
<td>4360.4221634</td>
<td>368.0000000</td>
<td>22750.0000000</td>
</tr>
<tr>
<td>PCAID</td>
<td>50</td>
<td>185.0200000</td>
<td>74.1984350</td>
<td>103.0000000</td>
<td>570.0000000</td>
</tr>
<tr>
<td>POP</td>
<td>50</td>
<td>4149.5200000</td>
<td>4399.9075744</td>
<td>325.0000000</td>
<td>20411.0000000</td>
</tr>
<tr>
<td>PCINC</td>
<td>50</td>
<td>4309.4600000</td>
<td>604.9145735</td>
<td>3188.0000000</td>
<td>5414.0000000</td>
</tr>
<tr>
<td>PCEXP</td>
<td>50</td>
<td>0.7941836</td>
<td>0.2472352</td>
<td>0.5049801</td>
<td>2.1476923</td>
</tr>
</tbody>
</table>

See Also:

- **STATISTICS** computes more detailed statistics on a single series. **EXTREMUM** locates the maximum and minimum values of a single series.
TAPER: Tapering Data

A taper is applied to the unpadded part of a complex series prior to taking the Fourier Transform. The taper reduces the “window leakage” by scaling the ends of the data so they merge smoothly with the zeros on either side.

Use of a taper is more important when you are smoothing (with \textsc{window}) using a window with a sharp cutoff such as the \textsc{flat} window. It has much less of an effect with \textsc{triangular} or \textsc{quadratic}.

\begin{verbatim}
taper( options ) cseries start end newcseries newstart
\end{verbatim}

Parameters

\begin{itemize}
  \item \textit{cseries} Complex series to taper.
  \item \textit{start \ and \ end} Range to taper. By default, \textit{this is the unpadded range of cseries}.
  \item \textit{newcseries} Series for the result of the tapering. By default, same as \textit{cseries}.
  \item \textit{newstart} Starting entry of the tapered series. By default, same as \textit{start}.
\end{itemize}

Options

\begin{itemize}
  \item \textit{type=[trapezoidal]/cosine} This gives the type of taper. See the discussion below.
  \item \textit{fraction=fraction of entries affected at each end \ [.25]} \textit{affected=number of entries affected at each end} Use one of these options to define the number of entries at each end of the series which are affected by the taper. This is the \textit{m} in the formulas below. \textsc{fraction} is the most convenient, specifying a fraction of the overall length. The default is .25 of the length.
\end{itemize}
Taper

The TYPE option

RATS provides two types of tapers: TRAPEZOID and COSINE. The taper multiplies the input series by the following \( m \) in these formulas is the number of affected entries at each end:

**TRAPEZOID:**

\[
b(t) = \begin{cases} 
  \frac{t}{m} & \text{for } 1 \leq t \leq m \\
  1 & \text{for } m + 1 \leq t \leq N - m \\
  \frac{(N - t + 1)}{m} & \text{for } N - m + 1 \leq t \leq N 
\end{cases}
\]

**COSINE:**

\[
b(t) = \begin{cases} 
  0.5[1 - \cos(\pi t/m)] & \text{for } 1 \leq t \leq m \\
  1 & \text{for } m + 1 \leq t \leq N - m \\
  0.5[1 - \cos(\pi (N - t + 1)/m)] & \text{for } N - m + 1 \leq t \leq N 
\end{cases}
\]

Variables Defined by TAPER

- **%SCALETAP** — the sum of squares of the taper weights (real). You should use this in place of the number of data points when scaling the periodogram. See the example below.

- **%KAPPA** — the factor to divide into %EDF and %EBW (computed by WINDOW) to correct for the effects of the taper.

The values of **%KAPPA** for the tapers are

\[
%KAPPA = \begin{cases} 
  \frac{[1 - 8m/5N]}{[1 - 4m/3N]^2} & \text{TYPE = TRAP} \\
  \frac{[1 - 2(m/N)(93/128)]}{[1 - 2(m/N)(5/8)]^2} & \text{TYPE = COSINE} 
\end{cases}
\]

Example

This uses a cosine taper affecting 20% of the data on either end.

```plaintext
frequency 3 768
rtoc 1956:1 2002:12
#  prices
#  1
taper(type=cosine,fraction=.20) 1 1956:1 2002:12
fft 1
cmult(scale=1./(2*%pi*%scaletap)) 1 1
```
**TEST — Testing Specific Coefficient Values**

**TEST** tests restrictions of the form $\beta = $ constant. You can also use it for tests of equality between two coefficient vectors and to do a simple Hausman test. In terms of complexity, it falls between **EXCLUDE**, which is simpler, but tests only zero restrictions, and **RESTRICT** and **MRESTRICT**, which test general linear restrictions.

```plaintext
test( options ) (no parameters)
# list of coefficient numbers for restriction (omit with ALL option)
# restricted values (omit with ZEROS or VECTOR options)
```

**Wizard**

In the *Regression Tests* wizard on the *Statistics* menu, use *Exclusion Restrictions* if you’re testing for zero values, and *Other Constant Restrictions* if some are non-zero.

**Supplementary Cards**

1. List the numbers of the coefficients which enter the restriction, by coefficient numbers, not by variable names. RATS puts coefficients for a **LINREG** or similar instruction in the regression in the order listed on the supplementary card. You can use TO triples, like “1 TO 5”, to abbreviate the list.

2. List the values you want these coefficients to assume under the restriction. **TEST** computes a joint test of these restrictions.

**Options**

`zeros/[nozeros]`

ZERO tests exclusion (zero) restrictions. Omit the second supplementary card if you use this option.

`[print]/noprint`

NOPRINT suppresses the regular output of **TEST**. You may want to use this option if you are just using **TEST** to compute the variables %CDSTAT or %SIGNIF.

`all/[noall]`

Use ALL to test whether all of the coefficients can be excluded. Omit the supplementary card if you use this. This option was called WHOLE in previous versions.

`form=f/chisquared`

This determines the form of the test statistic used. By default, RATS will select the appropriate form based upon the estimation technique used last. You can use FORM to manually select a distribution if you have made changes to the regression that require a different distribution, such as altering the %XX matrix in a way which incorporates the residual variance into %XX. See Section 5.13 in the *User’s Guide*.
vector=coefficient vector
The coefficient vector is a VECTOR which supplies the restricted values. If you use it, omit the second supplementary card. It must have the same size as the current regression. Used properly, this permits tests between two estimated coefficient vectors. See the note below.

covmat=input covariance matrix
coeffs=input coefficient matrix
These options provide an easy way to implement a Hausman test. Just save the covariance matrix and coefficient vector from the first (less efficient) estimation, do the second estimation, and then do TEST, using the options to input the saved covariance and coefficient arrays from the first regression.

title="string for output title"
You can use the TITLE option to include information in the output to identify what is being tested.

Notes
If you use the VECTOR option with a coefficient vector which has been estimated, you will get an incorrect test unless you replace the current %XX array by the covariance matrix of the difference between the two estimators. If not, TEST will use the covariance matrix of the more recent estimator, which is correct only for a test against a fixed set of values. See Section 6.9 of the User’s Guide for an example of a correct computation.

If the current covariance matrix is not full-rank, TEST will adjust the degrees of freedom of the test appropriately.

The main test statistic is usually shown as an $F$, but will be shown as a chi-squared when TEST is applied to likelihood-based estimates (from, for instance, DDV or GARCH instructions) or from any instruction for which the ROBUSTERRORS option was used during estimation.

For $F$-tests with one degree of freedom, TEST will report a two-tailed $t$ test in addition to the $F$ test.

For chi-squared tests with more than one degree of freedom, TEST will report an $F$ with an infinite number of denominator degrees of freedom (that is, the chi-squared statistic divided by the numerator degrees of freedom) in addition to the chi-square.
Examples

```
linreg logd 1 150
# constant logy logown logother
test(title="Test for Unit Elasticity")
#   3
# -1.0
```

This tests whether the LOGOWN coefficient is –1.0.

```
instruments constant z1 z2
linreg(inst) y
# constant x1 x2
compute [vect] beta2sls = %beta, [symm] xx2s1s=%xx
instruments(add) z3 z4 z5
linreg(inst) y
# constant x1 x2
test(coeffs=beta2sls,covmat=xx2s1s-%xx)
```

performs a Hausman test, comparing the 2SLS estimators from a small instrument set with that from a larger set which contains the first.

Variables Defined by TEST

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%CDSTAT</td>
<td>the computed test statistic (real).</td>
</tr>
<tr>
<td>%SIGNIF</td>
<td>the marginal significance level (real).</td>
</tr>
</tbody>
</table>
THEIL — Computing Forecast Performance Statistics

THEIL computes statistics on forecast performance for the variables of a model. You can also use the UForeErrors procedure to compute forecast performance statistics for univariate forecasting models. See Section 8.4 in the User’s Guide for more information on the subject.

\[ \text{theil(setup, other options)} \quad \text{equations} \]
\[
\# \text{ list of equations (omit if you use the \text{MODEL} option)}
\]
\[ \text{theil( options)} \quad \text{start} \]
\[ \text{theil(dump, other options)} \]

Description

For each variable and each forecast step, THEIL computes:

- the mean error
- the mean absolute error
- the root mean square (RMS) error
- Theil’s \( U \) statistic: a ratio of the RMS error to the RMS error of the “naive” forecast of no change in the dependent variable.

Using THEIL is a three-step process:

1) **Initialization**, using THEIL with the \text{SETUP} option. This describes the model and is done once for each analysis.

2) **Forecasting**, using THEIL \textit{without} the \text{SETUP} or \text{DUMP} options. This usually goes inside a loop, and does the forecasting and accumulation of forecast performance statistics. If you want to save the forecasts, use a \text{FORECAST} instruction in addition to THEIL.

3) **Final Statistics**, using THEIL with the option \text{DUMP}. This takes the forecast statistics and prints a table for each variable in the model.

Parameter—With the SETUP Option

\[ \text{equations} \quad \text{Number of equations in the system. List the equations on the supplementary card if you are not using the \text{MODEL} option.} \]

Two additional parameters used in older versions, \text{steps} and \text{data \_end}, have been replaced by options of the same name. Version 7 will recognize these parameters, but we recommend that you use the options in any new programs.
Options—With the SETUP Option

\begin{itemize}
  \item \texttt{steps=number of forecast steps}
  \item \texttt{to=last period for sample data [ALLOCATE length]}
\end{itemize}

Use \texttt{STEPS} to indicate the number of forecast steps (periods) to compute.

Use \texttt{TO} to indicate the last period in the sample. Since \texttt{THEIL} needs to compare the forecasts with actual data, it will not compute any forecasts for periods beyond that period. \texttt{TO} defaults to the \texttt{ALLOCATE length}.

\begin{itemize}
  \item \texttt{model=model name}
  \item \texttt{iters=iteration limit for solution algorithm [50]}
  \item \texttt{cvcrit=convergence criterion [.00001]}
  \item \texttt{damp=damping factor (1=no damping) [1]}
\end{itemize}

Use these when you are solving a model which you have created using \texttt{GROUP}. Omit the supplementary card when you use \texttt{MODEL}. The last three apply only if the \texttt{MODEL} contains \texttt{FRML}’s.

\begin{itemize}
  \item \texttt{estimate=re-estimation interval [1]}
\end{itemize}

Use \texttt{ESTIMATE} if you re-estimate the model at some interval other than every period (or never). When you apply the forecasting form of \texttt{THEIL}, \texttt{RATS} uses the current estimates to generate sets of forecasts beginning in the \textit{re-estimation interval} periods after \texttt{start}. \textit{You have to code the re-estimation yourself.}

Parameters—Forecasting Form

\begin{itemize}
  \item \texttt{start} \hspace{1cm} Start of the forecast period. Omit if using the \texttt{FROM} option.
\end{itemize}

Options—Forecasting Form

\begin{itemize}
  \item \texttt{print/ [noprint]}
    \begin{itemize}
      \item Use \texttt{PRINT} to get \texttt{THEIL} to print the forecasts together with the actual values. \textit{This can generate a lot of output.}
    \end{itemize}

  \item \texttt{from=starting period of the forecast interval}
    \begin{itemize}
      \item You can use the \texttt{FROM} option as an alternative to the \texttt{start} option for specifying the first forecasting period.
    \end{itemize}
\end{itemize}

Options—With the DUMP Option

\begin{itemize}
  \item \texttt{replace/ [noreplace]}
    \begin{itemize}
      \item With \texttt{REPLACE}, \texttt{THEIL} replaces the raw forecast statistics accumulated in the array \texttt{%THEIL} (defined by this instruction) with the finished statistics.
    \end{itemize}
    \begin{itemize}
      \item You can only use \texttt{REPLACE} when you have reached the end of the period that you are evaluating, since it destroys the raw statistics.
    \end{itemize}
\end{itemize}
**Theil**

`window="Title of window"`

If you use the `WINDOW` option, the output will be displayed in a (read-only) spreadsheet window with the title you supply. You can export the contents of this window to various file formats using `File–Export`.

**Description—With the DUMP Option**

`THEIL` with the `DUMP` option processes and prints the statistics that the previous commands have compiled. You can include several `THEIL(DUMP,...)` instructions in a sequence. For instance, you can check the performance through the first five years as well as through the full period.

**Using the THEIL instruction**

The recommended way to use `THEIL` is to estimate the model through some time period before the end of the data and evaluate the performance of the model over that last stretch of data, with periodic reestimation of the coefficients. This is as close as you can come to putting the model to an “honest” test, since you are always making the forecasts using coefficients estimated without use of the data from the forecast period itself.

You can easily do periodic reestimation for vector autoregressions by using `KALMAN`. For ARIMA models, you need to do a new `BOXJENK` instruction, and to save computation time, you may decide not to do this every period.

**Interpreting the Output**

This is an example of the summary for one variable.

<table>
<thead>
<tr>
<th>Forecast Statistics for Series USARGNP</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Step</td>
<td>Mean Error</td>
<td>Mean Abs Error</td>
<td>RMS Error</td>
<td>Theil U</td>
<td>N.Obs</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-0.002185156</td>
<td>0.004646036</td>
<td>0.006342988</td>
<td>0.6099094</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-0.005635293</td>
<td>0.008570236</td>
<td>0.011294940</td>
<td>0.6021547</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-0.010004031</td>
<td>0.012008270</td>
<td>0.016554306</td>
<td>0.6168643</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-0.014327303</td>
<td>0.016584875</td>
<td>0.021560530</td>
<td>0.6157698</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-0.018220694</td>
<td>0.020893524</td>
<td>0.026150736</td>
<td>0.5982887</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-0.021564104</td>
<td>0.024578887</td>
<td>0.029866540</td>
<td>0.5736811</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>-0.023924666</td>
<td>0.026974598</td>
<td>0.032002156</td>
<td>0.5278573</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>-0.025539256</td>
<td>0.028233874</td>
<td>0.033064425</td>
<td>0.4780999</td>
<td>17</td>
<td></td>
</tr>
</tbody>
</table>

**Mean Error vs. Mean Absolute Error**

If the Mean Error is approximately the same magnitude as the Mean Absolute Error, the model consistently forecasts either low (if the mean error is positive) or high (if negative). For instance, the model above apparently overestimates `USARGNP`, especially at longer time horizons.
Mean Absolute Error vs. RMS Error

The RMS error will always be at least as large as the mean absolute error. They will be equal only if all errors were exactly the same size. If the RMSE is several times as large as the MAE, there may very well be an error in your program (such as a mistake on the _data end_ parameter on _THEIL(setup)_), as this only happens if there are a few very large errors.

Theil’s U

Theil’s U statistic has several advantages over the RMS error when you are comparing models.

First, as a unit-free measurement, it is often easier to work with than the unit-bound RMSE. For instance, Theil U’s for interest rate series usually are between .8 and 1.0, while RMSE’s will vary depending upon term.

Second, it provides an immediate comparison of the forecasts with those of the naive scheme of forecasting no change over time.

A value in excess of one is not promising, since it means the model did worse than the naive method. However, a value substantially less than one should not necessarily be construed as a major success—almost any reasonable procedure will produce such a value for a series with a strong trend.

N. Obs column

The N. Obs column lists the number of different forecasts upon which the statistics at that horizon are based. Here, we’ve used _THEIL_ in a loop over 24 different starting periods. As the starting period approaches the end of the sample, there is no data against which to compare the forecasts of the later steps. Thus the total number of available data points is less for the longer horizons. For example, the table shows that the one-step horizon results are based on 24 different one-step forecasts (one for each starting period used), while the two-step horizon results are based on 23 forecasts, and so on.

Examples

```
boxjenk(define=bjeq,sdiffs=1,ar=2,sma=1) ldeuip * 1997:12
theil(setup,estimate=6,steps=24,to=2003:12) 1
# bjeq
do time=1998:1,2003:12,6
  theil time
  boxjenk(define=bjeq,sdiffs=1,ar=2,sma=1) ldeuip * time+5
end do time
theil(dump)
```

This evaluates a Box-Jenkins model over the period 1998:1 to 2003:12. To reduce computation time, the model is re-estimated only every 6th period. The _THEIL_ instruction inside the loop will do one sets of forecasts beginning with period _TIME_, another beginning with _TIME+1_, etc. through _TIME+5_.

---

**Theil**

RATS Reference Manual 479
Theil

system(model=canmodel)
variables canrgnp canm1s cantbill cancpinf canusxsr usargnp
lags 1 to 4
det constant
specify(tightness=.15,type=symmetric) 0.5
end(system)

* theil(setup,model=canmodel,steps=8,to=2006:4)
estimate(noprint,noftests) * 1997:4
do time=1998:1,2006:4
  if time==2003:1 ; theil(dump)
    theil time
    kalman
end
theil(dump)

This evaluates a six-variable VAR. The evaluation period is 1998:1 to 2006:4. The initial ESTIMATE goes through 1997:4, then the KALMAN instructions update the coefficients. The THEIL(DUMP) after the IF statement prints the intermediate results as of 2003:1.

Variables Defined by THEIL

%THEIL A VECTOR[RECTANGULAR] which holds the forecast statistics.
%THEIL(I) is a 5 x steps array with the statistics for the Ith equation.

The five rows hold the following raw/finished statistics:

1. sum of forecast errors / mean error
2. sum of absolute errors / mean absolute error
3. sum of squared errors / root mean square error
4. sum of squared errors of no-change forecasts / Theil U
5. number of forecast observations for this step

The REPLACE option of THEIL(DUMP) replaces the raw statistics with the finished statistics. For instance, if you replace the final THEIL(DUMP) in the second example with THEIL(DUMP,REPLACE), then %THEIL(3) (4,8) will be the Theil U (row 4) for step 8 for CANTBILL (3rd equation) and %THEIL(6) (3,5) will be the RMSE (row 3) for step 5 for USARGNP.

Technical Information

Each time you execute "THEIL start", RATS computes forecasts for steps horizons (or fewer, if the data end period is hit first). From these and the actual data, it computes forecast errors, absolute errors, squared errors, and squared errors of the naive (flat) forecasts. The program stores running sums of these statistics, and keeps track of the number of times that statistics have been computed for each horizon. We'll call this latter value $N_i$—the number of times that a forecast has been com-
puted for horizon $t$ ($t = 1$ for one-step-ahead forecasts, $t = 2$ for two-step-ahead forecasts, etc.). In the sample shown earlier in “Interpreting the Output”, $N_1$ was 24, $N_2$ was 23, and so on.

When you use `THEIL(DUMP)`, RATS divides the sums by $N_t$ to convert them into means, and from these it computes the RMS error and Theil $U$ statistics. The formulas for these computations (at forecast horizon $t$) are given below:

**Sum of Forecast Errors**

$$ SFE_t = \sum_{i=1}^{N_t} e_{it}, \quad e_{it} = y_t - \hat{y}_{it} $$

where $\hat{y}_{it}$ is the forecast at step $t$ from the $i$th call to `THEIL start`, and $y_t$ is the actual value of the dependent variable.

**Mean Error**

$$ ME_t = SFE_t / N_t $$

**Sum of Absolute Errors**

$$ SAE_t = \sum_{i=1}^{N_t} |e_{it}| $$

**Mean Absolute Error**

$$ MAE_t = SAE_t / N_t $$

**Sum of Squared Errors**

$$ SSE_t = \sum_{i=1}^{N_t} e_{it}^2 $$

**Root Mean Square error**

$$ RMS_t = \sqrt{SSE_t / N_t} $$

**SSE of no-change forecasts**

$$ SSENCF_t = \sum_{i=1}^{N_t} (y_t - y_{i0})^2 $$

where $y_{i0}$ is the “naive” or flat forecast—simply the value of the dependent variable at the period $(start-1)$ for the $i$th use of `THEIL start`.

**RMS of no-change forecasts**

$$ RMSNCF_t = \sqrt{SSENCF_t / N_t} $$

**Theil $U$**

$$ U_t = RMS_t / RMSNCF_t $$
TRFunction

TRFUNCTION: Transfer Functions

TRFUNCTION computes the transfer function of the lag polynomial which is implied by an equation. Unfortunately, the phrase “transfer function” is used in time series analysis in two different ways. If you are interested in a Box-Jenkins transfer function model, please see the instruction BOXJENK.

```
trfunction( option )       cseries   start   end
```

Parameters

- **cseries**: Complex series for the computed transfer function.
- **start, end**: Range of entries (frequencies) for which you want to compute the transfer function. By default: 1 and the FREQUENCY length.

Option

- **equation=equation**  [no default—you must specify an equation]
  
  Equation for which you want to compute the transfer function. See below for details.

Description

The equation must have one of the forms:

1. \( y_t = A(L)y_t \) (univariate autoregression)
2. \( y_t = A(L)y_t + B(L)u_t \) (univariate ARMA)
3. \( y_t = C(L)x_i \) (univariate distributed lag)
4. \( y_t = C(L)x_i + B(L)u_t \) (univariate distributed lag with MA error)
5. \( y_t = B(L)u_t \) (MA process)

In all of these, the variable CONSTANT may be among the right side variables.

RATS computes the transfer function by:

- Taking the Fourier transform of the lag polynomial \( 1 - A(L) \) in types 1 and 2 or \( C(L) \) for types 3 and 4.
- Dividing that by the Fourier transform of the moving average part \( B(L) \), if it is present.
Notes

If you are filtering in the frequency domain and are using a transfer function, remember that `CMULTIPLY` conjugates the second series as does the `*` operation in an expression such as `CSET`.

For simple transfer functions, particularly those with known coefficients, you might find it easier to use the function `%ZLAG` to construct the transfer function directly using `CSET`.

Examples

```
linreg(define=dlag) ipd  
# constant m1{0 to 8}  
frequency 1 512  
trfunc(equation=dlag) 1  
cmult 1 1  

Type 3: Distributed lag
Series 1 = Transfer function  
= Squared Gain

equation(noconstant,coeffs=| |.9,.3| |) arma y 1 1  
trfunc(equation=arma) 1
```

Defines $y_t = 0.9y_{t-1} + u_t + 0.3u_{t-1}$ and computes its transfer function.

The same transfer function as this last example can be created using the following `CSET` instruction.

```
cset 1 = (1-.9*%zlag(t,1))/(1+.3*%zlag(t,1))
```
**TVarying**

**TVARYING — Time-Varying Coefficients**

**TVARYING** is a subcommand of **SYSTEM** for adding time-varying coefficients properties to the Kalman filter. You must use it with **KFSET(VARIANCE=KNOWN)**. See Section 10.13.2 in the *User’s Guide* for details on time-varying coefficient models.

```
tvarying list of transition covariance matrices
```

**Parameters**

`list of ...` These are the state-transition covariance matrices, which are the covariance matrices of the changes in the coefficients. You need one array for each equation in the **SYSTEM**.

These are **SYMMETRIC** arrays. You do not need to **DECLARE** these in advance, but you do need to dimension and set them before you execute a **KALMAN** instruction.

**Notes**

If you want to write a single set of code which can handle **VAR**’s of various sizes, we recommend that you use a **VECTOR[SYMMETRIC]** in place of a list. For instance,

```
dec vector[symmetric] tvs(neqn)
tvarying tvs
```

makes **TVS(1)**, **TVS(2)**,...,**TVS(NEQN)** the list of matrices.

The **TVARYING** matrices, and the matrices for **KFSET** as well, have to be set before you can do a **KALMAN** instruction.

See the instruction **DLM** and Section 12.2 of the *User’s Guide* for information on using the Kalman filter for general state-space modelling. **TVARYING** is used for the specific situation where you are estimating a linear model with time-varying coefficients.

**See Also . . .**

See the **SYSTEM, KFSET, KALMAN** instructions.
TYPE — Setting Procedure Parameter Types

TYPE declares the data types of PROCEDURE or FUNCTION parameters. Use LOCAL for local variables used only within the procedure and DECLARE for global variables.

```
type   type   list of parameter names
```

Parameters

**type**

Variable type (three letters or more) you want to assign to the listed parameter names. This can be any of the basic and aggregate data types that RATS supports. By default, parameters for a PROCEDURE are type INTEGER, called by value, while those for a FUNCTION are type REAL, called by value.

**list of names**

The formal parameters of the procedure which are to have this type. Put * in front of a name if you want it to be called by address. Otherwise, it will be called by value.

Pass By Value or By Address

RATS allows you to pass data to procedures either by value or by memory address. Calls by value are the default. To make a parameter called by address, put * in front of its name. For instance, in

```
procedure bjfore depvar start end forecast
type series depvar *forecast
type integer start end
```

the parameters DEPVAR and FORECAST are type SERIES, with DEPVAR called by value and FORECAST by address. START and END are INTEGER called by value.

Use calls by value for any parameter which is input to the procedure, and will not be reset by the procedure. When you EXECUTE the procedure, the actual parameter can be any constant, variable or expression of the correct type. If, however, you use a variable name not previously declared, you will get an error.

Use a call by address for any parameter which is set by the procedure, such as the FORECAST series in BJFORE. When you EXECUTE the procedure, the actual parameter must be a variable, array or series element of the correct type. If not previously declared, it will be declared as a global variable with the required type.

Using the Parameters

Within a procedure, you can use the parameter names just like any other variable with its type. However, you cannot use parameters called by value on the left side of an equal sign—such a value is treated like a constant. For parameters passed by address, refer to the parameter by name alone, not “*name” as in the C language.
**UFORECAST — Univariate Forecasting**

**UFORECAST** computes dynamic or static forecasts for a single linear equation. To forecast a **MODEL** or a system of equations, use **FORECAST** (for dynamic forecasts) or **STEPS** (for static forecasts). These instructions also provide more flexibility than **UFORECAST**, and thus may be preferred in some single-equation forecasting situations. See Section 8.2 for more information.

```
uforecast(equation=equation, other options )  series  start  end
```

**Wizard**

You can use the Single-Equation Forecasts wizard on the Statistics menu to generate forecasts.

**Parameters**

- **series**
  
  The series into which you wish to store the forecasts. RATS will create this series if it doesn’t already exist.

- **start end**
  
  The range over which forecasts are to be computed. Omit these parameters if you use the FROM, TO, or STEPS option.

**Option**

- **equation=equation to forecast**
  
  The name or number of the equation to forecast. If you omit this option, RATS will use the most recently estimated regression.

- **from=starting period of the forecast interval**
- **to=ending period of the forecast interval**
- **steps=number of forecast steps to compute**
  
  These offer an alternative to the start and end parameters for setting the periods for which forecasts will be computed. If you have set a SMPL, these default to forecast over that range. You can use:

  - FROM and TO to set the starting and ending periods for the forecasts, or
  - FROM and STEPS to set the starting date and number of steps (periods)

- **static/[nostatic]**
  
  Use STATIC if you want static forecasts rather than dynamic forecasts (see Section 1.1.9 in the User’s Guide).
print/[noprint]

PRINT will display the forecasted and actual values in the output window.

window="Title for window" [unused]

The WINDOW option will create a (read-only) spreadsheet window showing the forecasted values. You can use File-Export... to export these to a file.

errors=series of forecast errors [unused]

stderrs=standard errors of forecast [unused]

ERRORS saves the forecast errors (differences between actual and forecasted values) into a series, while STDERRS computes and saves the standard errors of forecast. See the ERRORS instruction for technical details on the standard errors of forecast computation.

simulate/[nosimulate]

SIMULATE draws independent $N(0,\sigma^2)$ shocks over the forecast period where $\sigma^2$ is the residual variance for the equation or regression being forecast.

bootstrap/[nobootstrap]

BOOTSTRAP draws shocks over the forecast period randomly with replacement from the residuals associated with the equation or regression being forecast.
Examples

The command below forecasts an equation called \texttt{GDPEQ} for the four quarters of 2004. The forecasts are saved into \texttt{GDPFORE}.

\[ \texttt{uforecast(equation=gdpeq,from=2004:1,steps=4) gdpfore} \]

The next example, taken from page 210 of Diebold (2004), uses \texttt{UFORE} to generate forecasts from an ARIMA model. See the file \texttt{DIEBP216.PRG} for the full example.

\[ \texttt{boxjenk(regressors,ar=3,noconstant,define=ar3eq) lsales} \]
\[ \texttt{$1968:1$ $1993:12$ resid} \]
\[ \texttt{# time time2 seasons\{0 to -11\}} \]

\[ \texttt{Compute the forecasts, and their standard errors over the year 1994. Generate upper and lower 95\% confidence bands} \]

\[ \texttt{ufore(stderrs=stderrs,equation=ar3eq) fcst 1994:1 1994:12} \]
\[ \texttt{set upper 1994:1 1994:12 = fcst+1.96*stderrs} \]
\[ \texttt{set lower 1994:1 1994:12 = fcst-1.96*stderrs} \]

\[ \texttt{Create a dummy variable to be used in shading the forecast period. Since we're doing forecasts out to 1998:12 later on, we create this over that entire period.} \]

\[ \texttt{set forezone * 1998:12 = t>=1994:1} \]
\[ \texttt{graph(header="History and 12-Month-Ahead Forecast",}$\]
\[ \texttt{shading=forezone) 4} \]
\[ \texttt{# lsales 1992:1 1993:12} \]
\[ \texttt{# fcst 1994:1 1994:12} \]
\[ \texttt{# upper 1994:1 1994:12} \]
\[ \texttt{# lower 1994:1 1994:12} \]

The final example, from page 180 of Tsay (2005), uses the \texttt{UForeErrors} procedure to get benchmark forecasts using the sample mean, then compares the errors with those from an AR(1) estimation. See \texttt{TSAYP180.PRG} for the full example.

\[ \texttt{stats ibmln * 1997:12} \]
\[ \texttt{set benchfore 1998:1 1999:12 = %mean} \]
\[ \texttt{@uforeerrors ibmln benchfore} \]

\[ \texttt{AR(1) model} \]

\[ \texttt{linreg(define=ar1eq) ibmln * 1997:12} \]
\[ \texttt{# constant ibmln\{1\}} \]

\[ \texttt{Out-of-sample static (one step) forecasts for the AR(1)} \]

\[ \texttt{uforecast(equation=ar1eq,static) ar1fore 1998:1 1999:12} \]
\[ \texttt{@uforeerrors ibmln ar1fore} \]
UNTIL — Conditional Loops

UNTIL initiates a conditional loop of instructions. RATS will execute the instructions in the loop (the statement block) repeatedly until a specified condition is true. Because UNTIL tests the condition at the end of the loop, RATS will always execute the statements in the loop at least once. WHILE is similar but tests before it starts.

```plaintext
until  condition
    instruction or block of instructions to be executed until condition is true.
end until (omit if located inside a loop, procedure, or other compiled block).
```

Parameters

- **condition**: This is the expression that UNTIL tests. It can either be an integer or a real-valued expression. The condition is true if it has a non-zero value (such as 1) and false if it has value zero. Usually you construct it using logical and relational operators.

Statement Block

The statement block can be a single instruction, or a group of instructions enclosed in braces. It can also be a DO, DOFOR, WHILE, UNTIL or LOOP loop, or an IF-ELSE block. If you have any doubt about whether you have set up a block properly, just enclose the statements that you want executed in `{ and `}.

If the UNTIL instruction is the first instruction in a compiled section, you must use an END instruction after the statement block to signal the end of the compiled section.

Comments

Usually, the condition compares two variables, and it is quite common for one of these to be set only within the statement block following UNTIL. Since the condition is processed first (though not executed first), RATS will not yet have seen this variable. To get around this, you must DECLARE any variable which is introduced later in the loop.

Example

```plaintext
set smpl = 1.0
compute lastnobs=-1
until  lastnobs==%nobs {
    linreg(smpl=smpl) depvar
    # constant testv1 testv2 testv3 testv4
    set normalize = abs(%resids/sqrt(%seesq))
    set smpl = smpl.and.(normalize<3.00)
}
end until
```

Needed only if this loop isn’t part of a larger compiled section

This keeps running a regression, each time dropping all observations with residuals larger than three standard errors until no more observations are dropped.
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 RATS- DATA 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`.

---

490  *RATS Reference Manual*
USERMENU — Defining Pull-Down Menus

**USERMENU** allows you to add one or more of your own pull-down menus to the RATS menu bar. **USERMENU** is generally preferable to the older **MENU-CHOICE** combination in most situations. **MENU-CHOICE** is actually implemented as a “modal dialog box,” which may ask the user to make a choice about something she cannot see, as the dialog box is covering the entire window. **USERMENU** is implemented as a true menu. The user, rather than the dialog box, is in control of the organization of the windows, and can check back through output, look at graphs, save graphs or text, etc. before making a choice from the menu. However, as with **MENU-CHOICE**, it is your program which is in control of RATS itself.

```plaintext
usermenu(options)   pairs of id_number>>"menustring"
```

### Parameters

- **id_num>>"string"**
  - The parameter list specifies which menu items will be affected by the current **USERMENU** command. The **id_numbers** are integer values used to identify each menu item. For example, you might use 1, 2, 3 and 4, or 1000, 1001, 1002, and 1003 as **id_numbers**. The **menustrings** supply the corresponding names that will appear in the pull-down menu. They are required when you create the menu with **ACTION=DEFINE**, but are optional with **ACTION=MODIFY**.

When you define the menu, the parameter list must include all of the menu items you want in the menu. You cannot add or remove menu items after you have defined the menu.

With the **ACTION=MODIFY** option, the parameters specify the menu items you wish to modify. Use this to change the name of the item(s) (by supplying a different **menustring**), to enable or disable, and check or uncheck items.

All menu items must have a unique **id_num**, even if you are doing multiple **USERMENU**’s. See the section on “Using Multiple USERMENU’s” for more information.

### Options

- **id=menu identification number [0]**
  - If you want to use two or more **USERMENU**’s simultaneously, you must use the **ID** option with **ACTION=DEFINE** to assign a unique, non-zero integer identification number to the **USERMENU** being defined. You can then use that number to refer to the menu in later operations. Do not use the **ID** option when working with only one menu. See “Using Multiple USERMENU’s” below for details.
action=define/modify/[run]/remove

Use ACTION=DEFINE to define a menu, set its menu bar title, and the list of menu items. This is the first step. Note that the menu does not go into the menu bar until you do ACTION=RUN. This gives you time to modify its appearance after the initial definition.

Use ACTION=MODIFY to change the appearance or titles of items within a menu. See the ENABLE and CHECK options below.

ACTION=RUN actually adds your menu(s) to the menu bar (if they are not already there) and suspends the execution of the program until the user makes a selection from a menu. *Your menu is in (almost) complete control of the program.* The user has access to the other menu bar operations, but can execute no other RATS instructions until you say so. You should provide some form of a quit or abort item in your menu to allow the user to get out when she is done. When the user has selected an item from a USERMENU menu, execution continues with the line following USERMENU(ACTION=RUN) instruction. The variable %MENUCHOICE is set to the id_number of the selected item.

ACTION=REMOVE removes the menu(s) from menu bar. It is your job to do this when you no longer need the menu(s).

title="title string"

The TITLE option sets the title of the menu as it will appear in the menu bar. Use this option with ACTION=DEFINE. You should keep this short (single word) so the menu bar does not get cluttered.

enable=no/yes

With ACTION=MODIFY, you can use ENABLE=YES to enable the specified menu item(s), or ENABLE=NO to disable them. Use ENABLE=NO to prevent the user of your program from selecting items which cannot be executed yet, possibly because other steps must be completed first. All items are enabled initially.

check=no/yes

Use CHECK=YES with ACTION=MODIFY to place “check marks” in front of the specified menu item(s), and CHECK=NO to remove them. You can use this to indicate to the user that a “switch” is off or on, or that a certain operation has been completed.

The %MENUCHOICE Variable

As noted above, when the user selects an operation from a USERMENU, RATS stores the id_number of the selected menu item in the reserved variable %MENUCHOICE. You can use conditional statements to check the value of %MENUCHOICE and then control program flow accordingly. See the example below for details.
Using Multiple USERMENU's

RATS allows you to have several USERMENU’s active at one time. In general, these are handled just like single menus, except for the following:

- You define each menu with a separate USERMENU command. You must use the ID option to assign a unique non-zero id number to each menu.
- The ACTION=RUN option will always activate all currently defined menus, so you need to define all the menus before doing USERMENU(ACTION=RUN). The ID option is ignored with ACTION=RUN.
- By default, the ACTION=REMOVE option will remove all active menus from the menu bar. To remove only a particular menu, use the ID option along with ACTION=REMOVE.
- You must use unique menu item id numbers for each menu item (that is, two menu items cannot have the same id number, even if they appear on different menus).

Examples

The trivial example below creates a simple menu with three items. Selecting either of the first two items causes RATS to display a message in the output window. The third item closes the menu.

* Define a menu with three items:
  usermenu(action=define,title="Test")
  1>>"First item" 2>>"Second item" 3>>"Quit menu"

* Begin loop
  loop

  * Activate menu:
    usermenu(action=run)

  * Act on selection. If "Quit", close menu and break out of loop:
    if %menuchoice==1
      display "First item was selected"
    if %menuchoice==2
      display "Second item was selected"
    if %menuchoice==3
      { 
        usermenu(action=remove)
        break
      }
  end loop

See Section 16.4 in the User’s Guide for more examples.
**VAdd**

**VADD — Adding a Variable to An Equation**

VADD adds a variable or its lags to the equation being modified. The coefficients on the new variables are set to zero.

```
vadd( option ) newseries lags list of lags
```

**Parameters**

- **newseries**
  The series whose lag(s) you want to add to the equation. *Use the variable name %MVGAVGE to add a moving average lag.*

- **list of lags**
  The lags of newseries to add to the equation. If you only want to add the zero lag of newseries, just use VADD newseries. The added variables are given zero coefficients in the new equation.

**Option**

- **print/[noprint]**
  PRINT prints the modified equation.

**Examples**

```
linreg(define=gdpeq) gdp
# constant gdp{1} m1{0 to 3}
modify gdpeq
vadd %mvavgave lags 1 2
iterate gdpeq
```

This estimates an equation by OLS, then adds two moving average lags and re-estimates it using ITERATE.

**See Also**

- **MODIFY**
  Required to initiate modification of an equation.

- **VREPLACE**
  Replaces a variable in an equation with a transformation of a second variable.
VARIABLES — Specifying the Variables in a VAR

VARIABLES is one of the subcommands of SYSTEM for setting up a VAR. Use it to list the dependent variables of the system.

```
variables  list of endogenous variables
```

Wizard

The VAR (Setup/Estimate) wizard, located on the Statistics menu, provides an easy, dialog-driven interface for defining and estimating VAR models.

Parameters

```
list of ...  These are the dependent variables for the equations in the system.
```

If you are setting up an error-correction model using ECT, list the undifferenced variables. The restrictions are handled automatically by RATS.

Comments

If you do not use a prior, the order of subcommands VARIABLES, LAGS and DETERMINISTIC is unimportant.

Example

```
system(model=var12)
variables  tbillus  tbillcan  m1us  m1can  exrcan
lags 1 to 12
det  constant
end(system)
```

sets up a five variable VAR with twelve lags on each variable.

See Also

SYSTEM, LAGS, DETERMINISTIC, SPECIFY, ECT
VCV — Computing a Residual Covariance Matrix

VCV computes a variance-covariance matrix of a set of series, and can store the matrix for later use. It is designed primarily for use with a set of residuals. Note in particular that the default behavior is to compute the covariances without subtracting means.

```
VCV(options) start end
# list of series of residuals
```

Wizard

You can use the Covariance Matrix wizard on the Statistics menu to compute a covariance matrix. You can also compute a covariance matrix for a set of series by using Show Series Window on the Data menu, selecting the desired series, and clicking on the toolbar icon.

Parameters

```
start   end
```

Range of entries to use. If you have not set a SMPL, this defaults to the common defined range of all the listed series.

Supplementary Card

Lists the set of series for which VCV will compute the covariance matrix. Note that this is a list of series only: you cannot use regression format.

Options

```
[print]/noprint
```

Use NOPRINT to suppress the printing of the computed matrix. RATS prints the matrix with covariances on and below the diagonal and correlations above the diagonal. See the example on the next page.

```
centered/[nocentered]
```

By default, VCV does not subtract means out of the input series. If you use the option CENTERED, it does.

```
smpl=SMPL series or formula (User's Guide, Section 5.2)
spread=series of residual variances (Section 5.4)
```

With both these options, the SMPL and/or spread series applies to all listed series.

```
matrix=SYMMETRIC array
```

This stores the computed covariance matrix in the indicated array. You do not need to DECLARE or DIMENSION this array before doing VCV.
window="Title of window"

If you use the WINDOW option, the output is displayed in a (read-only) spreadsheet window. You can export the contents of this window to various file formats using File–Export....

Example

vcv(matrix=v) 1921:1 1941:1
# rcons rinv rwage

compares and prints the covariance matrix of three series, creates the 3x3 SYMMETRIC array V and saves the result there. A sample output is given below. This has the covariances on and below the diagonal and the correlations above it. The matrix V will have just the covariances.

Covariance\Correlation Matrix
RCONS     RINV       RWAGE
RCONS  0.89175982593  0.3010684027  -0.5780087046
RINV   0.41131881809  2.09304660284  0.3863244971
RWAGE -0.39361453883  0.40304589112  0.52002665162

Technical Information

VCV will, in general, give the same result as the SIGMA options on instructions such as ESTIMATE, NLSYSTEM and SUR if applied to the residuals from those. If $u_t$ is the column vector of the group of series at time $t$, then the estimate is

$$\hat{\Sigma} = \frac{1}{T} \sum_{t=1}^{T} u_t u_t'$$

Note the use of the $T$ divisor, without adjustment for “degrees of freedom.” The $T$ divisor gives the maximum likelihood estimator (in general), and will thus give a matrix which can be used in further likelihood-based analysis, such as testing and restricted modeling.

VCV will, in general, give a different result than instructions or functions which remove the means in the course of their calculations (examples are the function %COV, and the CMOM instruction when used with the CENTER or CORR options).

Also, because the standard errors of estimate in regression outputs are corrected for degrees of freedom, there will also be a scale factor difference between the SEESQ’s in the output and the diagonal elements produced by VCV.
Missing Values

Any observation which has a missing value for any of the series will be dropped from
the calculation.

Variables Defined by VCV

%LOGDET log determinant of the matrix (real)
%NOBS number of observations (integer)
%NVAR number of variables (integer)
%MEANV VECTOR of means of the variables (only if CENTER option)

See Also

The %COV(a,b) and %CORR(a,b) functions and the CMOMENT instruction compute
similar statistics (but with means subtracted for %COV and %CORR, and CMOM if used
with the CENTER or CORR options). The instruction RATIO takes two sets of compat-
ible residuals and conducts a likelihood ratio test based upon them.
VREPLACE — Substituting Variables Within an Equation

VREPLACE replaces a variable in the equation you are modifying (see MODIFY) with a transformation of another variable. You can also use VREPLACE to renormalize an equation so it has a different dependent variable.

\[
v\text{replace}(\text{option}) \quad \text{oldseries by newseries transform number}
\]

# lags in filter (only with transform=FILTER)
# coeffs of filter (only with transform=FILTER)

**Parameters**

- **oldseries**: The variable in old equation that you want to replace.
- **newseries**: The variable replacing oldseries.
- **transform**: The transformation which was done originally to newseries to produce oldseries. It should be one of the following (RATS accepts three or more letters as abbreviations):
  - **difference**: With number as the number of differences
  - **sdiff**: With number as the number of seasonal differences. See BOXJENK for information on seasonal differencing and seasonal spans.
  - **filter**: With the supplementary cards from FILTER repeated.
  - **equation**: FILTER with EQUATION = number.
  - **prewhitened**: Residuals from ARMA with EQUATION = number.
  - **lag**: With number equal to the lag.
  - **swap**: Renormalizes the equation so newseries is the new dependent variable, replacing oldseries. The zero lag of newseries should be present in the equation.

You can omit transform if you simply want to replace oldseries by newseries. You can also include both DIFF and SDIFF on a single instruction (other combinations require separate VREPLACE commands for each part).

- **number**: Depends upon the transformation as indicated above.
VReplace

Supplementary Cards

For a transform of FILTER, include the same supplementary cards which you used on the FILTER instruction.

Options

print/[noprint]

If you use the PRINT option, RATS prints the new equation.

Description

VREPLACE replaces variable oldseries by the indicated transformation(s) of variable newseries (transform=SWAP is explained on the previous page).

Examples

instruments constant dshift1 dshift2 sshift1 sshift2 sshift3
linreg(inst,frml=demandeq) price # constant quantity dshift1 dshift2
linreg(inst,define=supplyee) price # constant quantity sshift1 sshift2 sshift3
modify supplyee
vreplace price by quantity swap
frml(equation=supplyee) supplyeq
group market demandeq>>f_price supplyeq>>f_quant

estimates a supply-demand system by two-stage least squares, with PRICE as the left hand side variable in both equations. The MODIFY and VREPLACE instructions replace PRICE with QUANTITY on the left side of the supply equation, and the FRML converts the equation into a formula. The GROUP instruction groups the two formulas into a system which will determine both PRICE and QUANTITY.

boxjenk(ar=1,ma=1,define=yprewh) y / yres
boxjenk(ar=2,ma=0,define=xprewh) x / xres
boxjenk(ma=1,inputs=1,define=trfunc) yres # xres 0 1 0
modify trfunc
vreplace yres by y prewhitened yprewh
vreplace xres by x prewhitened xprewh

computes a transfer function model of Y on X. This first prewhitens Y and X by ARMA(1,1) and ARMA(2,0) models, respectively.

See Also

MODIFY, VADD

500 RATS Reference Manual
WHILE — Conditional Looping

**WHILE** sets up a conditional loop. RATS will execute the instructions in the statement block as long as the *condition* is true. **WHILE** tests the condition at the top of the loop, so it might never execute the statement block. The similar instruction **UNTIL** loops until a condition is false.

```plaintext
while  condition
    instruction or block of instructions to be executed as long as condition is true.
end while  (omit if located inside a loop, procedure, or other compiled block).
```

**Parameters**

*condition*    This is the expression that RATS tests. It can either be an integer or a real-valued expression. The *condition* is *false* if it has a value zero and *true* if it has any non-zero value. Usually, you construct it using logical and relational operators.

**Description**

When RATS encounters a **WHILE** instruction, it tests the *condition*. If it is true, RATS executes the statement or block of statements following the **WHILE** instruction, and then tests the condition again. As long as the condition is true, RATS will execute the statement(s). If the condition tests false, RATS immediately skips to the first instruction following the **WHILE** block. You can terminate the loop at any point with a **BREAK** instruction, or skip directly to the next *condition* check with **NEXT**.

**The Statement Block**

The statement(s) following the **WHILE** can be a single instruction, or a group of instructions enclosed in braces ({ }). You can also use a **DO**, **DOFOR**, **WHILE**, **UNTIL** or **LOOP** loop, or an **IF-ELSE** block. If you have any doubt about whether you have set up a block properly, just enclose the statements that you want executed in { and }.

If the **WHILE** instruction is the first instruction in a compiled section, you must use an **END** after the statement block to end the compiled section.

**Example**

```plaintext
compute count=0, i=0, sum=0.0
while count<20 {
    compute i=i+1
    if x(i)>0.0
        compute sum=sum+x(i), count=count+1
    }
end while  This is needed only if this is not in a procedure or loop.
```

computes the sum of the first 20 positive values of X.
WINDOW: Smoothing Spectral Estimates

WINDOW smooths periodograms and cross-periodograms to compute spectral estimates. Smoothing is required because the periodogram itself is an inconsistent estimate of the spectrum: the variance does not go to zero as the number of data points increases. Smoothing trades some bias for a decrease in variance.

```
window( options )  cseries  start  end  newcseries  newstart
```

Parameters

- **cseries**: Series to smooth.
- **start end**: Range of entries to smooth. This defaults to the defined range of cseries. It should always be the range used for the Fourier transform.
- **newcseries**: Series for the resulting smoothed series. By default, same as cseries.
- **newstart**: Starting entry for the smoothed series. By default, same as start.

Options

- **type=[flat]/tent/quadratic/triangular**: TYPE selects from the types of windows that RATS provides: flat, tent-shaped, or quadratic. FORM allows you to choose your own form for the window. See the next page for an explanation of these options.
- **form=VECTOR with the window form**:
- **width=window width \[0.75\sqrt{N}\]**: The width of the window. It must be odd. WINDOW will round an even value up to the next odd number.
- **mask=masking series**: Use this option when you are going to apply a mask to the bands of the spectrum. The smoothing transformation for entries near a masked-out band becomes increasingly one-sided so it gives no weight to the excluded ordinates. After WINDOW, you should multiply the smoothed series by the masking series.
Description

For a continuous spectral density there are two requirements for consistency:

- The window width must go to infinity as the number of data points increases (ensuring that the variance goes to zero).
- The window width must increase at a rate slower than the increase in the number of data points (ensuring that bias goes to zero).

RATS uses a default width based upon the square root of the number of ordinates. It offers several choices for the type of window, explained below.

RATS uses the following formula to smooth series $I$ to produce $S$:

$$S(j) = \sum_{-(n-1)}^{n-1} w_k I(j + k)$$

where $m$ is window width, $n = (m + 1)/2$ and the $w$s are the window weights. Series $I$ is considered to be periodic of period $end$—in smoothing the last entries of $I$, the window wraps around to use entries 1,2, etc. when the formula above asks for $end+1,end+2$, etc.

For a mean zero series, the ordinate for frequency zero has, perforce, a value of zero. Since most spectral analysis is done with mean zero series, WINDOW treats zero frequency as if it were a masked entry. This means that the moving averages for ordinates near this point are adjusted so they give no weight to the zero value. To include the zero frequency of cseries in the moving averages, you must use a MASK option with a masking series that has ones in all entries. CSET is the easiest way to create such a series.

The Window Options

$type=[\text{flat}]/\text{tent}/\text{quadratic}/\text{triangular}$
$form=\text{VECTOR} \text{ with the window form}$

The window weights $w_k$ are symmetric about $k=0$ and add up to one. Using the notation $m=\text{window width}, n = (m + 1)/2$, the windows provided through the TYPE option have the (unscaled) form:

$$w_k = \begin{cases} 
1 & \text{for } -(n-1) \leq k \leq (n-1) \\
\frac{n-|k|}{n} & \text{for } -(n-1) \leq k \leq (n-1) \\
\frac{n^2-k^2}{n} & \text{for } -(n-1) \leq k \leq (n-1)
\end{cases} \quad (\text{WINDOW}=\text{FLAT})$$

$$w_k = \begin{cases} 
1 & \text{for } -(n-1) \leq k \leq (n-1) \\
\frac{n-|k|}{n} & \text{for } -(n-1) \leq k \leq (n-1) \\
\frac{n^2-k^2}{n} & \text{for } -(n-1) \leq k \leq (n-1)
\end{cases} \quad (\text{WINDOW}=\text{TENT}, \text{TRIANGULAR})$$

$$w_k = \begin{cases} 
1 & \text{for } -(n-1) \leq k \leq (n-1) \\
\frac{n-|k|}{n} & \text{for } -(n-1) \leq k \leq (n-1) \\
\frac{n^2-k^2}{n} & \text{for } -(n-1) \leq k \leq (n-1)
\end{cases} \quad (\text{WINDOW}=\text{QUADRATIC})$$

The actual weights are scaled to sum to one.

Use the FORM option for a general symmetric window. The VECTOR is dimension $n$ and provides the values for $w_0,w_1,...,w_{n-1}$, that is, just one side of the window. RATS
automatically reflects this for the negative $k$'s and scales so the weights sum to one. For instance, the quartic window of width 11 could be done by:

```
declare vector mytent(6)
ewise mytent(k)=6**4-(k-1)**4
max at k=1, zero at k=7
window(form=mytent,width=11) 1 / 2
```

**Comments**

The choice of window type is largely a matter of personal preference. With relatively smooth economic time series, it makes little difference. When the spectrum is likely to have sharp features, the FLAT window, used in conjunction with a taper (instruction TAPER) is probably the best choice. If you are expecting some interesting features (such as peaks at the seasonals), it is usually a good idea to try several window widths, as very narrow peaks may get flattened by a window that is too wide.

**Variables Defined by WINDOW**

- **%EDF**: Equivalent Degrees of Freedom (real)
- **%EBW**: Equivalent Band Width (real)

The formulas for EDF and EBW are

\[
\text{EDF} = \frac{3n^4}{(2n^2 + 1)} \quad n = (m + 1)/2 \quad \text{for WINDOW = FLAT}
\]

\[
\text{EBW} = \frac{\pi \text{EDF}}{T} \quad \text{where } T \text{ is the number of ordinates}
\]

Note that for a padded series, the computed value for %EBW is correct, but %EDF must be corrected by multiplying by $N/T$ where $N$ is the number of actual data points.

**Example**

```
fft 1
cmult(scale=1.0/(2*pi%nobs)) 1 1
window(width=9) 1 / 2
window(width=15,type=triangular) 1 / 3
```

produces series 2 as the periodogram 1 smoothed with flat window of width 9 and series 3 as 1 smoothed with a tent, width 15.
WRITE — Printing Matrices

The principal use of WRITE is to print the contents of arrays to the screen or to a file. You can also use it to print simple variables and expressions, but DISPLAY is a superior choice for such situations.

WRITE is the companion instruction of READ, which reads data into arrays and variables. The REPORT instruction is probably your best choice if you need to collect and display a set of estimation results, particularly if you need to export those results into another application.

```
write( options ) arrays, variables, expressions
```

Parameters

arrays,... These are the objects to be displayed. WRITE will put each on a separate line (or set of lines for a matrix).

Options

```
unit=[output]/copy/other unit
```
The unit to which data are written.

```
format=[free]/binary/"(FORTRAN format)"
```
The format in which the data are to be written. With either FORMAT=FREE or FORMAT="(format)", WRITE prints the matrices by row. A FORTRAN format should be appropriate for displaying a row of an array, not the entire array.

```
[skip]/noskip
```
With FORMAT=FREE or FORMAT="(format)", SKIP (the default) puts two blank lines after each array printed. You can suppress these line skips by using NOSKIP.

Example

```
declare rectangular e(3,2)
input e
  1.5,2.0,0.0,3.5,2.7,4.6
write "E" e
```

```
E
  1.500000  2.000000
  0.000000  3.500000
  2.700000  4.600000
```
Write

Comments

WRITE does not label the arrays that it prints. You may find it helpful to include a descriptive string on the WRITE instruction as in the example.

WRITE always writes out arrays one at a time. DISPLAY, used within a loop, can produce output unavailable through WRITE:

```fortran
do i=1,n
    display(unit=copy) x1(i) x2(i) x3(i)
end do i
```

You can also use REPORT to display arrays. For instance, this will display x1, x2 and x3 in three columns, put into a common format that takes up 10 character positions.

```fortran
report(action=define,hlabels=||"X1","X2","X3"||)
report(atrow=1,atcol=1,fillby=cols) x1
report(atrow=1,atcol=2,fillby=cols) x2
report(atrow=1,atcol=3,fillby=cols) x3
report(action=format,width=10)
report(action=show)
```

See Also . . .

DISPLAY, REPORT, READ
X11 — X11 Seasonal Adjustment

X11 performs a seasonal adjustment on a series using the Census X11 method. The X11 instruction is only available in the Professional version of RATS—the command will have no effect in the standard versions. Please contact Estima if you are interested in upgrading to the Professional version.

**X11( options )  series  start  end  adjusted  factors**

**Wizard**
You can use the X11 wizard on the Data menu to seasonally adjust series.

**Overview**
X11 can only be applied to monthly or quarterly series containing at least three years of data. Note that the X11 method will not necessarily work well with some types of series:

- It assumes that the seasonal component changes, if at all, in a fairly smooth fashion.
- It assumes that the seasonality is primarily a function of the calendar.

If, instead, the series shows a fairly sharp break in its seasonal pattern, or if the seasonal is more a function of some other variable (e.g. weather-driven electric demand), X11 is likely to leave some obvious seasonality in all, or at least part, of the data. One simple way to check how well X11 is doing on a new series is to run it first on the first 3/4 of the data, then on the last 3/4. Compare the overlapping range.

**Parameters**

- **series**  Series to be seasonally adjusted.
- **start  end**  Range to use in X11 seasonal adjustment. If you have not set a SMPL, this defaults to the largest range of series.
- **adjusted**  Supply a series name for this parameter if you want to save the seasonally adjusted data in a series.

Two additional options, adjusted and factors, used in earlier versions have been replaced by options of the same names. Version 7 will still recognize these parameters, but we recommend that you use the option in any new programs.
Options

**multiplicative/[nomultiplicative]**

Chooses between a multiplicative or an additive decomposition of the series. Additive is the default. With the following notation:

- \( S \) = seasonal component
- \( TC \) = trend-cycle
- \( I \) = irregular
- \( TD \) = trading day
- \( H \) = holiday

The multiplicative decomposition is \( X = TC \times S \times I \times TD \times H \). Multiplicative is chosen when the magnitude of the seasonal fluctuations increases with the magnitude of the series itself. Almost without fail this will be the choice for a trending series.

The additive decomposition is \( X = TC + S + I + TD \). Note, that there is no holiday adjustment. This is the only possibility if the series has any non-positive elements. It will usually be the choice for a non-trending series.

**print= [none]/short/standard/long/full**

The **print** option controls the amount of output displayed by the \texttt{X11} instruction:

- **NONE** This produces no output—used for production runs where you don’t need to examine the output.
- **SHORT** This choice produces the minimum amount of printed output—only the initial series and final component estimates.
- **STANDARD** This is the standard level of X11 output—still primarily the final estimates.
- **LONG** This setting includes most of the important intermediate steps in addition to the standard output.
- **FULL** This reports on every step in the main adjustment process.

See the *X11 Manual Supplement* PDF file for details on which tables will be produced by each of the settings for **PRINT**.

**adjusted=series for the seasonal adjusted data**

**factors=series for the seasonal factors**

Use these if you want to save the seasonally adjusted series and/or the series of seasonal factors computed by \texttt{X11}. The seasonal factors series incorporates the seasonal component, plus trading day and holiday components, if any, and includes forecasts for three years beyond the adjustment period.
[graduate] / nograduate

lower=Lower limit standard errors from the mean [1.5]
upper=Upper limit standard errors from the mean [2.5]

Graduating extremes reduces the effect of outliers on the estimates of the seasonal factors. This only makes a great deal of difference if a series has a good deal of non-seasonal variability. The actual process used in graduating extreme values is fairly complicated, but basically, it drops data points which are more than “Upper Limit” standard errors from the mean (within a certain calculation), leaves unchanged those closer to the mean than “Lower Limit” standard errors and makes a smooth transition from inclusion to exclusion for those in between. The default values for the limits 1.5 and 2.5.

tradeday=[none] / compute / apply / signif

tdsigma=outlier limit standard errors from the mean [2.5]

These options are only applicable to monthly data. They allow you to apply a “trading day adjustment” for variation due to the number of Mondays, Tuesdays, etc., in a month. A typical series which would benefit from trading day adjustment is a total retail sales series, where there would be considerable predictable variation among the days of the week.

NONE The trading day option is not applied.

COMPUTE Computes the trading day factors and displays them in the X11 output (with PRINT=SHORT,...,FULL), but does not use them in the final adjustment. If, after examining the computed factors, you want to use the trading day adjustment, you can simply re-run X11 with TRADEDAY=APPLY or TRADEDAY=SIGNIF.

APPLY Computes the trading day factors and applies them to the final adjustment.

SIGNIF Computes the trading day factors and applies them to the final adjustment only if they are significant.

TDSIGMA sets the cutoff limit (in standard errors) used in the trading day calculation. Data points which exceed this limit are dropped completely from the trading day regression.

easter/ [noeaster]
laborday/ [nolaborday]
thanksgiving/ [nothanksgiving]

These are only available for multiplicative decompositions of monthly data. They adjust for calendar variation due to the placement of (U.S.) holidays. For instance, tourist trade in September might be relatively higher in months when
Labor Day comes late than when it comes early. As with trading day variation, series such as total retail sales are the most likely candidates for holiday adjustment. Labor Day adjustment affects August and September, Thanksgiving affects November and December, and Easter affects March and April. The extra adjustment factors for any pair are constrained to sum to 200%. The three choices can be used in any combination.

Holiday adjustment doubles the computation time, as two passes through the X11 process are required, and the payoff tends to be minimal. Labor Day and Thanksgiving Adjustments, in particular, rarely make any noticeable difference in the adjusted series, even when their regressions are statistically significant.

Examples
This seasonally adjusts DEUIP (German Industrial Production) by applying the multiplicative decomposition, and graduated extremes with the default lower and upper limits of 1.5 and 2.5. The adjusted series and the seasonal factors are saved into DEUADJIP and DEUFACT, respectively.

```
calendar(m) 1961:12
allocate 1988:1
open data examx11.rat
data(format=rats) / deuip
x11(multiplicative,factors=deufact) deuip / deuadjip
```

The following seasonally adjusts the series SALES, using the multiplicative decomposition, graduated extremes, and the three holiday adjustments. The trading day adjustment will also be applied if the trading day factors are significant:

```
x11(easter,laborday,thanks,tradeday=signif,print=standard) $ sales 1980:1 1993:12 salessa
```

The following seasonally adjusts RATE using an additive decomposition, without graduated extremes:

```
x11(nomult,nograduate,print=short) rate / adj_rate
```

Technical Information
For technical details, please see the X11 Manual Supplement PDF file (included with the Professional version of RATS).
Section 2: The RATS Functions

Section 2 of the Reference Manual provides a detailed listing of every built-in instruction. Functions differ from instructions in that they are not stand-alone executable objects—they can only be called as part of an expression in a RATS instruction. They are most commonly used on the right-hand-side expressions of \texttt{COMPUTE}, \texttt{SET}, and \texttt{FRML} instructions, although they can be used in many other circumstances. For example, functions can appear in conditional expressions on \texttt{IF}, \texttt{UNTIL}, and \texttt{WHILE} instructions, or in \texttt{DISPLAY} instructions to format or generate output. They can also be used in expressions on instruction parameters, such as the start or end parameters on a \texttt{LINREG} instruction. You invoke a function with the syntax:

\begin{verbatim}
function( argument1, argument2, ...)
\end{verbatim}

Separate the arguments with commas. Note that the left parenthesis must come immediately after the function name, with no blank spaces in between.

For example, suppose you want to compute the transpose of matrix A. You can do that using the \texttt{TR()} function in a \texttt{COMPUTE} instruction:

\begin{verbatim}
compute atrans = tr(a)
\end{verbatim}

For more information on Functions, see Sections 1.1.10 and 1.8 of the User's Guide.

Organization of this Section

Section 2 begins with a quick reference to the available functions grouped by type or purpose, which begins on the next page. This is followed by an alphabetical listing of all functions with details on usage.
### Mathematical Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>abs(x)</code></td>
<td>Absolute value of a scalar or matrix</td>
</tr>
<tr>
<td><code>%besselj(n,x)</code></td>
<td>Bessel function of the first kind</td>
</tr>
<tr>
<td><code>%binomial(x,n)</code></td>
<td>Binomial coefficient</td>
</tr>
<tr>
<td><code>%boxcox(x,y)</code></td>
<td>Box-Cox transformation</td>
</tr>
<tr>
<td><code>exp(x)</code></td>
<td>Exponentiation function</td>
</tr>
<tr>
<td><code>%factorial(x)</code></td>
<td>Factorial function</td>
</tr>
<tr>
<td><code>fix(x)</code></td>
<td>Convert real to integer by truncation</td>
</tr>
<tr>
<td><code>float(n)</code></td>
<td>Convert integer to real</td>
</tr>
<tr>
<td><code>%frac(x)</code></td>
<td>Fraction of a real number</td>
</tr>
<tr>
<td><code>%isimpson(x,f)</code></td>
<td>Computes numerical integral by Simpson’s rule</td>
</tr>
<tr>
<td><code>%itrapezoid(x,f)</code></td>
<td>Computes numerical integral by trapezoidal rule</td>
</tr>
<tr>
<td><code>log(x)</code></td>
<td>Natural log function</td>
</tr>
<tr>
<td><code>%max(x,y)</code></td>
<td>Maximum of two reals</td>
</tr>
<tr>
<td><code>%min(x,y)</code></td>
<td>Minimum of two reals</td>
</tr>
<tr>
<td><code>%noprec(x)</code></td>
<td>Tests for loss of precision</td>
</tr>
<tr>
<td><code>%round(x,n)</code></td>
<td>Rounding</td>
</tr>
<tr>
<td><code>sign(x)</code></td>
<td>Sign function</td>
</tr>
<tr>
<td><code>sqrt(x)</code></td>
<td>Square root function</td>
</tr>
<tr>
<td><code>%tsign(x,y)</code></td>
<td>Sign transfer</td>
</tr>
<tr>
<td><code>%valid(x)</code></td>
<td>Check for valid (non-missing) value</td>
</tr>
</tbody>
</table>

### Trigonometric Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%acos(x)</code></td>
<td>Arc (inverse) cosine</td>
</tr>
<tr>
<td><code>%asin(x)</code></td>
<td>Arc (inverse) sine</td>
</tr>
<tr>
<td><code>%atan(x)</code></td>
<td>Arc (inverse) tangent</td>
</tr>
<tr>
<td><code>cos(x)</code></td>
<td>Cosine function</td>
</tr>
<tr>
<td><code>%cosh(x)</code></td>
<td>Hyperbolic cosine</td>
</tr>
<tr>
<td><code>%cot(x)</code></td>
<td>Cotangent function</td>
</tr>
<tr>
<td><code>%csc(x)</code></td>
<td>Cosecant function</td>
</tr>
<tr>
<td><code>%sec(x)</code></td>
<td>Secant function</td>
</tr>
<tr>
<td><code>sin(x)</code></td>
<td>Sine function</td>
</tr>
<tr>
<td><code>%sinh(x)</code></td>
<td>Hyperbolic sine</td>
</tr>
<tr>
<td><code>tan(x)</code></td>
<td>Tangent function</td>
</tr>
<tr>
<td><code>%tanh(x)</code></td>
<td>Hyperbolic tangent</td>
</tr>
</tbody>
</table>

### Matrix Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%abs(A)</code></td>
<td>Elementwise absolute value of a matrix</td>
</tr>
<tr>
<td><code>%avg(A)</code></td>
<td>Average of values</td>
</tr>
<tr>
<td><code>%blockdiag(VR)</code></td>
<td>Creates a block diagonal matrix from <code>VECT[RECTANG]</code></td>
</tr>
<tr>
<td><code>%blockglue(G)</code></td>
<td>Concatenation of matrices</td>
</tr>
<tr>
<td><code>%blocksplit(A,I)</code></td>
<td>Partitioning of matrix</td>
</tr>
<tr>
<td><code>%bqfactor(S,LS)</code></td>
<td>Blanchard-Quah factorization</td>
</tr>
<tr>
<td><code>%cols(A)</code></td>
<td>Number of columns of a matrix</td>
</tr>
<tr>
<td><code>%compress(A,v)</code></td>
<td>Compress out rows of A where elements of V are zero</td>
</tr>
<tr>
<td><code>%const(x)</code></td>
<td>Fill matrix with a constant value</td>
</tr>
<tr>
<td><code>%corr(A,B)</code></td>
<td>Correlation coefficient</td>
</tr>
<tr>
<td><code>%corrtocv(C,V)</code></td>
<td>Convert correlation matrix to covariance matrix</td>
</tr>
<tr>
<td><code>%cov(A,B)</code></td>
<td>Covariance of two arrays</td>
</tr>
</tbody>
</table>
%cvtocorr(V)\hspace{1cm} 
Converts a covariance matrix to correlations

%cxadj(z)\hspace{1cm} 
Complex matrix adjoint (conjugate transpose)

%cxeigdecomp(z)\hspace{1cm} 
Complex Eigen decomposition

%cxinv(z)\hspace{1cm} 
Complex matrix inverse

%cxsvd(Z)\hspace{1cm} 
Complex singular value decomposition

%decomp(A)\hspace{1cm} 
Choleski decomposition

%det(A)\hspace{1cm} 
Determinant of an array

%diag(A)\hspace{1cm} 
Diagonal matrix from a 1-dimensional array

%dot(A,B)\hspace{1cm} 
Dot product of two arrays

%eigdecomp(S)\hspace{1cm} 
Eigen decomposition of symmetric array

%exp(x)\hspace{1cm} 
Elementwise exponentiation function

%fill(r,c,v)\hspace{1cm} 
Create matrix filled with a single value

%fractiles(A,F)\hspace{1cm} 
Fractiles of an array

%ginv(A)\hspace{1cm} 
Generalized inverse of a matrix

%gsortho(A)\hspace{1cm} 
Gram-Schmidt orthonormalization

%identity(size)\hspace{1cm} 
Create identity matrix

%index(V)\hspace{1cm} 
Sorting index for a vector

%innerxx(A)\hspace{1cm} 
Matrix Inner cross product

%inv(A)\hspace{1cm} 
Inverse of a matrix

%kroneker(A,B)\hspace{1cm} 
Kroneker product of two arrays

%kronid(A,B)\hspace{1cm} 
Kroneker product with identity, post multiplied

%kronmult(A,B,C)\hspace{1cm} 
Kroneker product, post multiplied

%log(A)\hspace{1cm} 
Elementwise natural log function

%logdetxx(s)\hspace{1cm} 
Returns the log determinant of a symmetric matrix

%ltinv(L)\hspace{1cm} 
Inverse of packed triangular matrix

%ltouterxx(A)\hspace{1cm} 
Outer matrix product of packed triangular matrix

%matpeek(A,coords)\hspace{1cm} 
Extracting entries from a sparse matrix

%matpoke(A,coords,v)\hspace{1cm} 
Filling entries in a sparse matrix

%maxindex(A)\hspace{1cm} 
Location of maximum value of an array

%maxvalue(A)\hspace{1cm} 
Get maximum value of an array

%minindex(A)\hspace{1cm} 
Location of minimum value of an array

%minvalue(A)\hspace{1cm} 
Get minimum value of an array

%mqform(A,B)\hspace{1cm} 
Matrix quadratic form operation

%mqformdiag(A,B)\hspace{1cm} 
Diagonal of matrix quadratic form

%mscalar(x)\hspace{1cm} 
Create a scalar matrix

%normsqr(A)\hspace{1cm} 
Sum of the squared elements of a matrix or vector

%nullspace(A)\hspace{1cm} 
Column null space

%outerxx(A)\hspace{1cm} 
Matrix outer cross product

%parmspeek(P)\hspace{1cm} 
Extract parameters from a PARMSET

%parmspoke(P,v)\hspace{1cm} 
Put vector into a PARMSET

%patchmat(A,B)\hspace{1cm} 
Replaces NA's in A with entries from B

%patchzero(A)\hspace{1cm} 
Replaces NA's in A with zeros

%perp(A)\hspace{1cm} 
Returns a matrix forming basis for null space of A

%psddiag(A,B)\hspace{1cm} 
Diagonalizer matrix for a symmetric

%psdfactor(A,I)\hspace{1cm} 
General Choleski factorization

%psdinit(A,W)\hspace{1cm} 
Stationary solution for initializing state space model

%psubmat(A,r,c,B)\hspace{1cm} 
Copy values from one array into another

%psubvec(VA,p,VB)\hspace{1cm} 
Copy values from one vector into another

%pt(v,t,x)\hspace{1cm} 
Puts information from an array into an array of series

%qform(A,B)\hspace{1cm} 
Quadratic form
%qformd(A,B) Diagonal quadratic form
%qforminv(S,V) Quadratic form using an inverted matrix.
%qrdecomp(A) QR decomposition
%ran(x) Random normal draw
%ranks(A) Ranks the elements of a matrix
%ranmat(M,N) Rectangular of random normal draws
%ranmvnormal(F) Random multivariate normal draw
%ranwishart(n,r) Random Wishart matrix
%ranwishartf(F,r) Random Wishart given a covariance matrix
%ranwisharti(F,r) Random inverse Wishart matrix
%rows(A) Number of rows in a matrix
%rsscmom(C,B) Residual sum of squares from cross-product matrix
%scalar(A) First element of a matrix
%seqa(start,incr,n) Creates sequence of real numbers
%sigmacmom(C,B) Sum of squared multivariate error matrix from CMOM
%solve(A,B) Solve linear equations
%sort(A) Sorts an array
%sortc(A,c) Sorts an array based on the values in a column
%sortcl(A,list) Sorts an array based on values in multiple columns
%sqrt(A) Elementwise square root
%stereo(V) Stereo projection
%sum(A) Sum of array elements
%svdecomp(A) Singular value decomposition
%sweep(A,k) Sweep function of a matrix
%sweeplist(A,list) Sweep function over a list of pivots
%sweeptop(A,k) Sweep function over pivots in top rows
%symmcol(n) Returns column number for packed position n
%symmpos(r,c) Returns position in “packed” symmetric of element (r,c)
%symmrow(r) Returns row number for packed position n
%testdiff(base,test) Test difference of two arrays
%trace(A) Transpose of a matrix
%trace(A) Trace of a matrix
%unitv(n,i) Creates unit vector
%vec(A) Vectorizes an array
%vectorect(V,r) Reshapes Vector to Rectangular
%vectosymm(V,r) Reshapes Vector to Symmetric
%wfractiles(A,W,F) Weighted fractiles
%xcol(A,i) Extracts a column of a matrix
%xdiag(A) Extracts diagonal from a matrix
%xrow(A,i) Extracts a row of a matrix
%xsubmat(A,m,n,o,p) Extracts a rectangular block from a matrix
%xsubvec(V,sr,er) Extracts a block from a vector
%xt(v,t) Extracts information from an array of series
%zeros(rows,cols) Returns a matrix of zeros
Distribution/Probability Functions

%betainc(x,a,b)  Incomplete beta function
%cdf(x)  Cumulative density function of standard Normal
%chisqr(x,r)  Chi-squared tail probability
%chisqrdensity(x,r)  Density of the chi-square
%chisqrcdf(x,r,nc)  Non-central chi-squared CDF
%chisqrncdensity(x,r,nc)  Density of the non-central chi-square
%density(x)  Standard Normal density function
%digamma(x)  Digamma function (derivative of the log gamma)
%dmills(x)  Derivative of the inverse Mills’ ratio
%ftest(x,n,m)  F-test tail probability
%gcd(x)  Gamma function
%gammainc(x,n,m)  Incomplete gamma function
%gedcdf(x,s)  CDF of GED distribution
%gevcdf(x,k,m,s)  CDF of generalized extreme value distribution
%gpcdf(x,k,m,s)  CDF of generalized Pareto distribution
%invchisqr(p,r)  Inverse chi-square test
%invchisqrcdf(p,r)  Inverse non-central chi-square CDF
%invftest(p,n,m)  Inverse F-test
%invgevdensity(x,c)  Inverse GED CDF
%invgevdensity(p,k,m,s)  Inverse Generalized Extreme Value CDF
%invgev(p,k,m,s)  Inverse Generalized Pareto CDF
%invgpdf(p,k,m,s)  Inverse t CDF
%invtcdf(p,d)  Inverse t test
%lnbeta(a,b)  Natural log of Beta function
%lngamma(x)  Natural log of the Gamma function
%logbetadensity(x,a,b)  Log density of Beta distribution
%logcdf(v,x)  Log of the logistic CDF
%logconcdf(S,v)  Log of the logistic CDF
%logdensity(A,v)  Log of the Normal PDF
%logdenstycdfv(S1,S2,n)  Log PDF of the multivariate Normal
%loggeddensity(x,c,v)  Log GED density
%logistic(x)  Logistic function
%lognegbin(x,r,p)  Log of the negative binomial density
%logtdensity(V,U,nu)  Log multivariate t density
%mills(x)  Inverse Mills’ ratio
%negbin(x,r,p)  Negative binomial density
%nyblomtest(x,p)  Nyblom fluctuations test significance level
%poisson(mean,k)  Poisson function
%qformpdf(A,x)  CDF of quadratic form
%qformpdf(D,x)  CDF of diagonal quadratic form
%tcdfnc(x,r)  CDF for non-central t density
%tdensity(x,r)  t density
%tdensitync(x,r)  Non-central t density
%trigamma(x)  Trigamma function (second derivative of log gamma)
%ttest(x,n)  Two-tailed t test
%ztest(x)  Two-tailed standard normal probability
Functions

Random Number Generation Functions

%ran(x)             Random normal draw
%ranbeta(a,b)       Random beta draw
%ranbranch(P)       Random selection of a branch
%ranchisqr(n,k)     Random chi-squared draw
%rancombo(n,k)      Random combination
%rangamma(r)        Random gammas
%rangrid(Vx,Vf)     Random draw from distribution approximated by grid
%raninteger(l,u)    Random integer
%ranlogkernel()     Log kernel density from most recent random draw
%ranmvkron(S,X)     Rectangular of random normal draws
%ranmvkroncmom(C,r,p,x) Draw from multivariate Normal regression
%ranmvnormal(F)     Draw from MV Normal regression, CMOM variant
%ranmvpost(p1,m1,p2,m2) Random draw from multivariate normal posterior
%ranmvpostcmom(C,r,p,x) Multivariate normal posterior draw, CMOM variant
%ranpermute(n)      Random permutation
%ransphere(n)       Random draw on unit sphere
%rant(d)            Random draw from Student $t$ distribution
%ranwishart(n,r)    Random Wishart matrix
%ranwishartf(F,r)   Random Wishart given a covariance matrix
%ranwisharti(F,r)   Random inverse Wishart matrix
%uniform(low,high) Random draw from a uniform distribution

String, Label, and Output Formatting Functions

%bestrep(A,n)       “Best” representation
%concat(first,last) Concatenate two labels
%datelabel(t)       Date label for a given date/entry number
%l(series)          Get label of a series
%label(variable)     Get label of a series or other variable
%left(s,n)          Left substring
%mid(s,m,n)         Middle substring
%minimalrep(A,n)    Determines “minimal” representation
%right(s,n)         Right substring
%s(label)           Get (or create) series with a given label
%strcmp(s1,s2)      Compare two strings (case-sensitive comparison)
%strcmpnc(s1,s2)    Compare two strings disregarding case
%string(n)          Convert integer value to a string
%strlen(s)          Returns the length of a string
%strlower(s)        Convert string to lower case
%strrep(s,n )       Repeats a string
%strupper(s)        Convert string to upper case
%strval(x,f)        String showing a numerical value
%value(s)           Get numeric value from a string
Functions

Date, Entry Number Functions

- `%allocend()` - Default series length set by `ALLOCATE`
- `%cal(year, period)` - Entry number of specified date
- `%calendar()` - Returns the current `CALENDAR` setting
- `%closestdate(y, m, d, dow)` - Observance date for y:m:d (closest day of week)
- `%closestweekday(y, m, d)` - Observance date for y:m:d (closest Monday–Friday)
- `%dateandtime()` - Current date and time
- `%datelabel(t)` - Date label for a given date/entry number
- `%day(t)` - Day of month for entry T
- `%daycount(t)` - Number of days in entry T
- `%dow(y, m, d)` - Day of week for given date
- `%easter(y, m, d)` - Dates after March 23rd for (Western) Easter
- `%floatingdate(y, m, dow, n)` - Day of month for floating holiday
- `%indiv(t)` - Individual within a panel set
- `%julian(t)` - Number of days from Jan 1, 1901 or Jan 1, 0001
- `%month(t)` - Month number (1–12) of entry T
- `%panelobs()` - Size of panel data time dimension
- `%period(t)` - Period within a panel set
- `%regend()` - Ending entry of most recent regression
- `%regstart()` - Starting entry of most recent regression
- `%today()` - Today’s entry
- `%tradeday(t, d)` - Number of occurrences of day D in period T
- `%weekday(t)` - Day of week of entry T
- `%year(t)` - Year number

Financial Functions

- `%annuity(p, r, n)` - Present value of an annuity
- `%payment(a, r, n)` - Required payments for an annuity

Complex-Valued and Related Functions

- `%arg(z)` - Argument of a complex number
- `%cabs(z)` - Complex absolute value
- `%cexp(z)` - Complex $e^z$
- `%clog(z)` - Complex natural log
- `%cmplx(real, imag)` - Complex number from real and imaginary parts
- `%conjg(z)` - Complex conjugate
- `%csqrt(z)` - Complex Square root
- `%cxadj(z)` - Complex matrix adjoint (conjugate transpose)
- `%cxeigdecomp(z)` - Complex Eigen decomposition
- `%cxinv(z)` - Complex matrix inverse
- `%cxsvd(Z)` - Complex singular value decomposition
- `%freqend()` - Number of ordinates used by `FREQUENCY`
- `%freqsize(n)` - Recommended number of complex ordinates
- `%imag(z)` - Imaginary part of complex number
- `%real(z)` - Real part of complex number
- `%unit(x)` - Unit circle value $\exp(ix)$
- `%unit2(n, T)` - Unit circle value for $\exp(2\pi i (n - 1)/T)$
- `%zlag(t, x)` - Unit circle value for $2\pi(t - 1)x / N$
## Functions

### Equation/Regression List Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%eqncoeffs(eqn)</code></td>
<td>Coefficients of an equation</td>
</tr>
<tr>
<td><code>%eqndepvar(eqn)</code></td>
<td>Dependent variable of an equation</td>
</tr>
<tr>
<td><code>%eqngetidentity(eqn)</code></td>
<td>“Identity” tag for an equation</td>
</tr>
<tr>
<td><code>%eqnhandle(eqn)</code></td>
<td>Get “handle” of an equation</td>
</tr>
<tr>
<td><code>%eqnlag(eqn, lag)</code></td>
<td>Lags the right hand side of an equation</td>
</tr>
<tr>
<td><code>%eqnlagpoly(eqn, s)</code></td>
<td>Extract lag polynomial from an equation</td>
</tr>
<tr>
<td><code>%eqnprj(eqn, t)</code></td>
<td>Fitted value of an equation for a given entry</td>
</tr>
<tr>
<td><code>%eqnregcoeffs(eqn)</code></td>
<td>Gets labels for regressors of an equation</td>
</tr>
<tr>
<td><code>%eqnsetcoeffs(eqn, c)</code></td>
<td>Sets the coefficients of an equation</td>
</tr>
<tr>
<td><code>%eqnsetidentity(eqn, n)</code></td>
<td>Sets the “identity” tag for an equation</td>
</tr>
<tr>
<td><code>%eqnsetresids(eqn, r)</code></td>
<td>Sets the residuals associated with an equation</td>
</tr>
<tr>
<td><code>%eqnsize(eqn)</code></td>
<td>Sets the residual variance of an equation</td>
</tr>
<tr>
<td><code>%eqntable(number)</code></td>
<td>Number of explanatory variables in equation</td>
</tr>
<tr>
<td><code>%eqnvariance(eqn)</code></td>
<td>List of variables/lags in an equation</td>
</tr>
<tr>
<td><code>%eqnxvector(eqn, t)</code></td>
<td>Variance associated with an equation</td>
</tr>
<tr>
<td><code>%instlist()</code></td>
<td>Extract an X(t) vector for an equation</td>
</tr>
<tr>
<td><code>%insttable()</code></td>
<td>Get current instruments as a regressor list</td>
</tr>
<tr>
<td><code>%instxvector(entry)</code></td>
<td>Get current instruments as a table</td>
</tr>
<tr>
<td><code>%reglist()</code></td>
<td>Extract an X(t) for the current instrument set</td>
</tr>
<tr>
<td><code>%regload()</code></td>
<td>Get regressor list from last regression</td>
</tr>
<tr>
<td><code>%regsave()</code></td>
<td>Reload regression saved with <code>%REGSAVE()</code></td>
</tr>
<tr>
<td><code>%rladdlag(R,S,L)</code></td>
<td>Save a regression to a VECTOR of INTEGERS</td>
</tr>
<tr>
<td><code>%rladdlaglist(R,S,L)</code></td>
<td>Add a lagged regressor to a regressor list</td>
</tr>
<tr>
<td><code>%rladdone(R,S)</code></td>
<td>Add list of lags to a regressor list</td>
</tr>
<tr>
<td><code>%rlconcat(R1,R1)</code></td>
<td>Add a regressor to a regressor list</td>
</tr>
<tr>
<td><code>%rlcount(R)</code></td>
<td>Concatenate two regressor lists</td>
</tr>
<tr>
<td><code>%rlempty()</code></td>
<td>Number of regressors in list</td>
</tr>
<tr>
<td><code>%rlfromtable(Table)</code></td>
<td>Returns an empty regressor list</td>
</tr>
<tr>
<td><code>%rlverify(R)</code></td>
<td>Convert a regression table to a regressor list</td>
</tr>
<tr>
<td><code>%tableextract(T1,list)</code></td>
<td>Verify a regressor list</td>
</tr>
<tr>
<td><code>%tablefind(T1,S,L)</code></td>
<td>Extracts subtable from a table</td>
</tr>
<tr>
<td><code>%tablefindall(T1,T2)</code></td>
<td>Locate a single (variable,lag) pair in a table</td>
</tr>
<tr>
<td><code>%tablefindlags(T1,S)</code></td>
<td>Locate variables from one table in another</td>
</tr>
<tr>
<td><code>%tablefindmiss(T1,T2)</code></td>
<td>Locate occurrences of a single variable in a table</td>
</tr>
<tr>
<td><code>%tablefromrl(R)</code></td>
<td>Locate variables missing from a table</td>
</tr>
<tr>
<td><code>%tablemerge(T1,T2)</code></td>
<td>Create table from regressor list</td>
</tr>
</tbody>
</table>

### Model Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%modelcompanion(mdl)</code></td>
<td>Companion matrix for a model</td>
</tr>
<tr>
<td><code>%modeldepvars(mdl)</code></td>
<td>Get list of dependent variables from a model</td>
</tr>
<tr>
<td><code>%modeleqn(mdl, n)</code></td>
<td>Get copy of equation from a model</td>
</tr>
<tr>
<td><code>%modelfind(mdl, y)</code></td>
<td>Locates an equation in a model by dependent variable</td>
</tr>
<tr>
<td><code>%modelgetcoeffs(mdl)</code></td>
<td>Get coefficient matrix for a model</td>
</tr>
<tr>
<td><code>%modelgetvcv(mdl)</code></td>
<td>Get covariance matrix for a model</td>
</tr>
<tr>
<td><code>%modellagsums(mdl)</code></td>
<td>Compute VAR lag sums for a model</td>
</tr>
<tr>
<td><code>%modellabel(mdl, n)</code></td>
<td>Label of dependent variable in an equation in a model</td>
</tr>
</tbody>
</table>
%modelpoke(mdl,n,eqn) Replaces an equation in a model
%modelsetcoeffs(m,c) Set coefficient matrix for a model
%modelsetcvv(mdl,S) Set covariance matrix for a model
%modelsze(model) Number of equations in a model
%modelsubstect(model) Substitute out error-correction terms

**Polynomial Functions**

%polyadd(V1,V2) Add two polynomials
%polycxroots(V) Complex roots of a polynomial
%polydiv(V1,V2,d) Divide two polynomials
%polymult(V1,V2) Multiply two polynomials
%polyroots(V) (Real) roots of a polynomial
%polysub(V1,V2) Subtract two polynomials
%polyvalue(V1,x) Evaluate a polynomial at a given value for \( x \)

**Utility Functions**

%(expressions) Evaluates complex/multiple expressions
%allocend() Default series length set by ALLOCATE
%bin(x,V) Assigns value to correct “bin”
%clock(m,n) Modified modular division
%cputime() CPU timer (in seconds)
%defined(name) Status of procedure parameter or option
%do(i,n,m,expr) Internal do loop
%iabs(n) Integer absolute value
%idiv(m,n) Integer division (integer/integer = integer result)
%if(x,y,z) Evaluate a conditional expression
%imax(m,n) Maximum of two integers
%imin(m,n) Minimum of two integers
%l(series) Get label of a series
%mod(m,n) Modular division
%ovcheck(x) Checks for potential overflow
%parmspeek(P) Extract parameters from a PARMSET
%parmspoke(P,v) Put vector into a PARMSET
%pt(v,t,x) Puts information from an array into an array of series
%ratsversion() Get RATS version number
%regend() Ending entry of most recent regression
%regstart() Starting entry of most recent regression
%s(label) Get (or create) series with a given label
%selseries() Return series selected in the Series Window
%seq(m,n) Creates an integer sequence
%seriesentry(ser,t) For panel data references outside current individual
%unitfn(s) Get file name for I/O unit
%unitfnext(s) Get extension of file name for I/O unit
%unitfnpath(s) Get path of file for I/O unit
%unitfnroot(s) Get root name of file for I/O unit
%valid(x) Check for valid (non-missing) value
%xt(v,t) Extracts information from an array of series
Alphabetical Listing of RATS Functions

This section 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, \( \text{ABS} \) returns the absolute value of the scalar. For matrix argument \( A \), \%\text{ABS} \) returns a matrix \( B \) such that: \( B(i,j) = |A(i,j)| \)

Argument type: Real for \( \text{ABS} \), array of reals for \%\text{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 \( |x| \leq 1 \), 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 \text{ALLOCATE} or first \text{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| ≤ 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

%bin( x, V ) — Determine appropriate “bin” for a value
This returns an integer indicating which “bin” the value x would fall in given a a vector V of boundary values defining the bins. For example, %bin(2.5, |1.0, 2.0, 3.0, 4.0|) returns the 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

%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
Functions

%binomial( x, n ) — Binomial coefficient
Returns the binomial coefficient $x(x-1)...(x-n+1)/n! x(x-1)...(x-n+1)/n!$

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

%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

%blockspl( A, I ) — Matrix partitioning
Returns a RECTANGULAR of RECTANGULARs 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: $(x^y - 1)/y$, 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:

$L(L^{-1}SL^{-1})^{1/2}$

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 \( F(x) \), where \( F \) 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 \( \chi^2 \) 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 noncentrality $nc$. 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 of the
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

%clock( m, n ) — Modified modular division
Does a modified form of modular division, returning $[(m-1) \text{ modulo } n] + 1$. 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)
```


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
%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 (Nx1 and 1xN 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.

Argument type: Real
Returns type: Real

%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: \( \frac{1}{NM} \sum_{j}(A_{ij} - \bar{A})(B_{ij} - \bar{B}) \)

Argument types: Real arrays. A and B must have the same dimensions (RATS considers Nx1 and 1xN to be equivalent).
Returns type: Real

%cputime() — CPU Timer
Returns the CPU’s internal time in seconds. Useful for determining 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 $V$:
$$C_{ij} = \frac{V_{ii} V_{jj}}{\sqrt{V_{ii} V_{jj}}}$$
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

%cxeigdecomp( Z ) — Complex eigen decomposition
Returns the eigen decomposition of a Hermitian complex matrix. The return is a vector of complex matrices,
$$Z = U \text{diag}(W) V^*$$
where $*$ denotes the conjugate transpose.
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 matrix.
Argument type: Complex matrix (must be N x 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 $Z = U \text{diag}(W) V^*$ (where $*$ denotes the conjugate transpose).
$U$ and $V$ are column-orthonormal matrices, that is, $U^* U = I$, $V^* V = I$. 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(2002:11) is 30. With quarterly data, %DAYCOUNT(2002:2) is 91.

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

%decomp( S ) — Choleski decomposition
Computes the Choleski decomposition of a SYMMETRIC matrix. For a matrix S, this returns a lower-triangular RECTANGULAR array F, such that F′F = 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 Section 16.2.3 of 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×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 Nx1 or 1xN RECTANGULAR array.

Argument type: Vector or Nx1 or 1xN 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

%dills( x ) — Derivative of the inverse Mills’ ratio

Returns the derivative evaluated at \( x \) of the inverse Mills’ ratio for the normal \( \frac{\phi}{\Phi} \). The inverse Mills’ ratio itself is computed by the %MILLS function.

Argument type: Real
Returns type: Real

%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 Nx1 and 1xN 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×1 matrix of eigenvalues, and the second is the N×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 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 4.7.2 of the User’s Guide 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.

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 for the regressors (explanatory variables) in an equation. These are formatted as they are shown in regression output, with, for instance, series(lag) used 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

%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 x 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: 2xN Rectangular array of integers

%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 , entry ) — Extract an X(t) vector for an equation
Returns a VECTOR with the “X” variables from the equation for the indicated entry. Use 0 for equation to pull the explanatory variables from the most recent regression.

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

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 \( B(i,j) = e^{A(i,j)} \).

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

%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
**Functions**

**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 Section 1.8 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 nth 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(2007,11,4,4)` is the fourth Thursday of November, 2007, while `%floatingdate(2007,11,4,-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

**%freqend() — Number of frequency ordinates**
Returns the number of ordinates set by the FREQUENCY.

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 data points.

Argument type: Integer
Returns type: Integer
%ftest( x, n, m ) — F-test tail probability
Returns the tail probability of $F(n,m)$.

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) for the value $x$ and shape $s$.

Argument types: Real
Returns type: Real

%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 $A^\dagger$ which solves $AA^\dagger A = A$ and $A^\dagger AA^\dagger = A^\dagger$. $A$ does not have to be square; $A^\dagger$ will have the same dimensions as $A$.

Argument type: Rectangular array ($m \times n$)
Returns type: Rectangular array ($n \times m$)

%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

%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, y, and z can be real or integer values). 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 = Real
Returns type: Real

%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 Nx1 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 A’A for the matrix 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 x n array.

Argument types: None
Returns type: Vector of Integers

%insttable() — Table of instruments
Returns the table (a 2 x 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 A. A can be SYMMETRIC or NxN 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, \( \text{chisqr}(\%\text{invchisqr}(p,r),r) = p \)

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, \( \text{chisqrnc}(\%\text{invchisqrnc}(p,r,nc),r,nc) = p \)

Argument types: \( p \) and \( nc \) Real, \( r \) an Integer. With \( 0 \leq p \leq 1 \) and \( r > 0 \)
Returns type: Real

%invftest( p, n, m ) — Inverse F-test tail probability
Returns the value \( x \) which for which the tail probability of an \( F \) with \( n \) and \( m \) degrees of freedom is \( p \). In particular, \( \text{ftest}(\%\text{invftest}(p,n,m),n,m) = p \).

Argument types: Real, with \( 0 \leq p \leq 1 \). \( n \) 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 \( \exp\left(-\frac{|x|^{2/c}}{2}\right) \).

Argument type: Real. \( 0 \leq p \leq 1 \), \( c \) 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, \( \text{cdf}(\%\text{invnormal}(p)) = p \)

Argument type: Real. \( 0 \leq p \leq 1 \)
Returns type: Real
%invtdf( 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. 0 ≤ p ≤ 1. 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( Vx, Vf ) — 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

%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 × nq matrix C = 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 \((A \otimes B)C\) where \(A \otimes B\) 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 \(\log(\exp(x)/(1+\exp(x)))\), 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
Functions

%log( A ) — Elementwise natural log function
Elementwise log function. This returns an array B such that $B_{ij} = \log_e(A_{ij})$.

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

%logbetadensity( x, α, β ) — Log Density of Beta
Returns the log density of the beta distribution for x, given α and β.

Argument type: Real
Returns type: Real

%logcdf( v, x ) — Natural log of the normal CDF
Returns $\log(\Phi(x/\sqrt{v}))$ where v is the variance, and $\Phi$ is the normal CDF.

Argument type: Reals
Returns type: Real

%logconcdensity( ˆΣ, T ) — Log concentrated MV normal density
Returns the log concentrated density for a (multivariate) normal with $\hat{\Sigma}$ and number of observations T:

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

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:

$$-(1/2) (N \log 2\pi + \log|A| + v'A^{-1}v)$$

Argument types: A is NxN Symmetric, v is a Vector (or Nx1 or 1xN Rectangular). Both can be scalars as well.
Returns type: Real

%logdensitycv( Σ, ˆΣ, T ) — Log multivariate normal density
Returns the log of multivariate normal density as a function of Σ, $\hat{\Sigma}$, and the number of observations (T):

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

Argument types: Σ and $\hat{\Sigma}$ are NxN Symmetric (or scalar Reals), T is an integer.
Returns type: Real.
%logdetxx( S ) — Log determinant of a symmetric matrix

Returns \( \log |S| \) for a symmetric matrix S.

Argument type: Symmetric
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: \( \frac{1}{1 + b e^{-x}} \). 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: None

%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 this 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.
%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 inverse
For a packed matrix L, this returns $LL'$ of the lower-triangular portion.

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 ≤ 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
Functions

%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

%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

%modellagsums( M ) — Lag sums for a VAR model
Computes the lag sums for a VAR model M. This is the matrix $I - \sum_{s=1}^{l} \Phi_s$ 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

%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

%modelsze( 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 $B'A'B$.

Argument types: $A$ must be $N\times N$ Rectangular or Symmetric; $B$ can be any array that allows the $B'A'B$ 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 $B'A'B$.

Argument types: Same as for %mqform(), 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

%negbin( x, r, p ) — Negative binomial density
Returns the negative binomial density $\binom{r+x-1}{x} \left( \frac{p}{1-p} \right)^x$. 

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 $A^\perp$ 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 $A A'$ for the matrix $A$. Use $\%INNERXX(A)$ to get $A'A$.

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 dimension
Returns the number of periods per individual for panel data as was set on the \texttt{CALENDAR} instruction.

Argument type: None
Returns type: Integer

%parmspeek( P ) — Extract parameters from a PARMSET
Returns the parameters from a \texttt{PARMSET} in vector form.

Argument type: Parameter set
Returns type: Vector

%parmspoke( P, v ) — Put vector into a PARMSET
Resets the parameters in \texttt{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
This patches over (replaces) any NA's in the \texttt{Base} matrix with the corresponding entries from the \texttt{Source} matrix. \texttt{Base} and \texttt{Source} must have conforming dimensions.

Argument types: Any matrix (both must have same dimensions)
Returns type: Matrix of same type/dimensions as \texttt{Base}

%patchzero( Base ) — Replace NA's in a matrix with zeros
This patches over (replaces) any NA's in the \texttt{Base} matrix with zeros.

Argument types: Any matrix
Returns type: Matrix of same type/dimensions as \texttt{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 \( A^\bot \) which forms a basis for the null space of \( A \). If \( m > n \) (that is, \( A \) has more rows than columns), this is a basis for the left null space, so \( A^\bot A = 0 \). \( A^\bot \) is \( m \times (m - n) \). If \( m < n \), it will be a basis for the right null space, \( A^\bot A = 0 \), with \( A^\bot \) being \( n \times (n - m) \).

Argument types: Rectangular array
Returns type: Rectangular array

%poisson( mean, k ) — Poisson function
Returns \( P(x \leq k \mid m = \text{mean}) \) for a Poisson process with the mean given by \( \text{mean} \).

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 \( V \) represents an \( n \)-degree polynomial, in the form \( V(1) + V(2)x^1 + \ldots + V(n+1)x^n \).

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 ) — Add 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
Multiplies two polynomials, represented by the vectors V1 and V2.

Argument types: Vectors
Returns type: Vector

%polyvalue( V1, 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: Vector

%psddiag( S , B ) — Diagonalizing a symmetric matrix
For the positive semi-definite SYMMETRIC S, returns a lower triangular matrix L such that 
$L S L' \text{ is diagonal with 1's and 0's on the diagonal. } B \text{ is used to determine when a calculation has produced a zero (to machine precision). If S is full rank, } \%\text{PSDDIAG}( S , B ) \text{ will give } 
L S L' = 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, \ldots, 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 $A A' + W = V$. This is the stationary covariance
matrix for a state-space model.

Argument types: A is NxN Rectangular; W is NxN 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. The copied
values will overwrite any existing values in the affected entries of A. Note that A must be
sufficiently large to allow all the entries in B to be copied into the 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 $B'AB$, where B is either a VECTOR or an Nx1 or 1xN RECTANGULAR array.

Argument types: Vector or Rectangular array
Returns type: Real

%qformd( A, B ) — Evaluate a quadratic form with a diagonal matrix

Returns $\sum A_i B_i^2$ where A and B are either conforming VECTORS or Nx1 or 1XN RECTANGULAR arrays.

Argument types: Vector or Rectangular array
Returns type: Real

%qformdpdf( D, x ) — PDF of a diagonal quadratic form

Returns $P(e'De < x)$ where 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 $v'S^{-1}v$ where S is positive semi-definite symmetric.

Argument types: Symmetric, Vector
Returns type: Real

%qformpdf( A, x ) — PDF of a quadratic form

Computes $P(e' Ae < x)$ for a quadratic form in Normal(0,1) variables, where A is SYMMETRIC and e is i.i.d. This can be used for ratios of quadratic forms as well, since $P(e' Ae/e' Be < x) = P(e'(A - xB)e < 0)$.

Argument types: Symmetric, Real
Returns type: Real
%qrdecomp( A ) — QR decomposition
Returns a VECTOR [RECTANGULAR] with two elements, the Q and R, respectively, of \( QR = A \),
where Q is orthonormal (\( QQ' = I \)) and 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 \( \{1, \ldots, \text{dim}(P)\} \) 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 \( \{1, \ldots, n\} \).

Argument types: \( n \) and \( k \) are integers, both positive
Returns type: Vector of Integers with dimension \( k \)
%rangamma( r ) — Random gammas
Returns either a single draw or an array of draws from a gamma distribution with shape
parameter \( r \).

Argument type: \( \text{Real. } r > 0 \)
Returns type: \( \text{Real, unless in the specific form } A=\text{RANGAMMA}(r) \), where \( A \) is a real
array with dimensions already set, in which case independent draws are
placed into the elements of \( A \).

%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 \( A \). The 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 \( m \times n \) rectangular matrix filled with 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 \((X'X)^{-1}\) matrix (FX).

Argument types: Rectangular
Returns type: Rectangular (dim FX x 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 $FF'$, 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.

%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 into the elements of A.
%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 \( FF' \) and shape parameter \( r \).

Argument types: \( F \) is Rectangular, \( r \) is a positive real. \( r \) must be greater than \( \text{dim}(F) \).
Returns type: Symmetric

%ranwisharti( F, r ) — Random inverse Wishart

Returns a draw from an inverse Wishart distribution with covariance matrix \( FF' \) and shape parameter \( r \).

Argument types: \( F \) is Rectangular, \( r \) is a positive real number greater than \( \text{dim}(F) \).
Returns type: Symmetric

%ratsversion() — Get version number of RATS

This returns the version number of the copy of RATS you are using (for example, “7.00”).

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 %RLADDREG and related functions to modify the regressorlist.

Argument type: None
Returns type: Vector of Integers

%regload() — Reload saved regression

This loads into memory a regression that was saved using %REGSAVE(). Subsequent instructions such as TEST, other hypothesis testing operations, and PRJ will apply to this regression.
Functions

Argument type: Regression (Vector of Integers created using %REGSAVE(1))
Returns type: Vector of Integers

%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

%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. Use %RLADDONLE( ) to add a series without
any lag terms.

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
that 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, R1 ) — 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)
Functions

%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

%rlempthy() — 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

%rlfromtable( Table ) — Convert a regression table to a regressor list
Converts a regression table (such as generated by %EQNTABLE or %INSTTABLE) into a regressor list.

Argument type: Table is a regression table—a 2 x 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 structures) 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

%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

%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.

Argument type: Real
Returns type: Real

%sinh( x ) — Hyperbolic sine
Returns the hyperbolic sine of its argument: \((\exp(x) - \exp(-x))/2\)

Argument type: Real
Returns type: Real

%solve( A, B ) — Solves linear equations
Returns the solution \(x\) of \(Ax = B\), where \(A\) is an \(N\times N\) real matrix, \(B\) 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 based upon the values of the columns in the list. The list
should be a VECTOR of INTEGERS, or a “literal matrix” of the form \(\text{||col1,...,coln||}\). 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.
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 \( B_j = \sqrt{A_j} \)

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

%stereo( V ) — Stereo projection

Returns the conversion of a general \( n - 1 \) 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 \( s1 < s2 \).

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 \( s1 < s2 \).

Argument types: 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

%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

%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 current default range (either the range defined by CALENDAR and ALLOCATE or by an SMPL instruction).
  Argument type: Real array or series
  Returns type: Real

%svdecomp( A ) — Singular Value Decomposition
Returns a VECTOR[RECTANGULAR] with three elements, the U, W and V, respectively, of A = U \left[ \text{diag}(W) \right] 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 RECTANGULAR or SYMMETRIC array; k is any integer from 1 to N, where A is an NxN matrix. B = %sweep(A, k) returns:
\[ B(k,k) = 1.0/A(k,k) \]
\[ B(i,k) = -A(i,k)/A(k,k) \quad i \neq j \]
\[ B(k,j) = A(k,j)/A(k,k) \quad j \neq k \]
\[ B(i,j) = A(i,j) - A(i,k)A(k,i)/A(k,k) \quad \text{otherwise} \]

Note that \%SWEEP(\%SWEEP(A, k), k) = A. See Section 4.7.1 of the User’s Guide 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)

\%sweepn( A, k ) — Sweep function of a matrix
Returns the result of sweeping matrix A on all pivots 1,...,k. The sweep function is described with \%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 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 \( (row \text{ and } column \geq 1) \)
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 Rectangular 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}) 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.

Argument type: Real
Returns type: Real

%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

%tcdfnc( x, d, nc ) — CDF for non-central \( t \) distribution
Returns the CDF of \( x \) for a non-central \( t \) distribution, with degrees of freedom \( d \), and non-centrality parameter \( nc \).

Argument type: Real
Returns type: Real

%tdensity( x, r ) — \( t \) density
Returns the density function at \( x \) for a \( t \) distribution with \( r \) degrees of freedom.

Argument type: Real. \( r \) must be positive.
Return type: Real

%tdensitync( x, d, nc ) — Non-central \( t \) distribution
Returns the density function of \( x \) 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:

\[
\max_i \left\{ \min \left[ \left| \frac{test_i - base_i}{base_i} \right|, \left| test_i - base_i \right| \right] \right\}
\]

Argument types: Real vectors of same dimension
Returns type: Real
Functions

%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 NxN 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 2004 started on a Thursday, and so has five of each of Thursday, Friday and Saturday, and four of the other days. With a monthly calendar, %TRADEDAY(2004:1,1) is 4 while %TRADEDAY(2004:1,5) 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)
    Return 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 t with r degrees of freedom, that is, the probability that a t-distributed random variable with r degrees of freedom exceeds x.
    Argument types: Real
    Return 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 $\exp(\imath x)$.

Argument type: Real
Returns type: Complex

%unit2( n, T ) — Unit circle value for $2\pi(n-1)/T$
Returns the unit circle value $\exp\left(2\pi\imath \left(n-1\right)/T\right)$.

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 the function $\%unitfn(\text{"data"})$ 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 $\%unitfnext(\text{"data"})$ 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 $\%unitfnpath(\text{"data"})$ 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”, %unitfnroot("data") would return “mydata”.

Argument type: String (unit name)
Returns type: String

%unity( 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 $x$ is not the system missing value (%NA). In all other cases, it returns a zero. Use it to test for missing values in a series, etc.

Argument type: Real
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 which has the values of $A$ arranged as a VECTOR. If $A$ is RECTANGULAR, $A$ is stripped out by columns, if SYMMETRIC, 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 one of the rectangular array, elements $r+1$ through $2r$ go into column two, 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 \textit{t}.

Argument type:  
Integer

Returns type:  
Integer

%wfractiles(A, W, F)  —  Compute fractiles
Returns a \texttt{VECTOR} with fractiles of the elements of the array \textit{A} with (unnormalized) weights provided by \textit{W}. \textit{W} needs to have the same shape as \textit{A}. The list of desired fractiles are provided by the \texttt{VECTOR} \textit{F}.

Argument type:  
\textit{A} and \textit{W} are real arrays, \textit{F} is a Vector

Returns type:  
\texttt{Vector}

%xcol(A, i)  —  Extract column of a matrix
Extracts column \textit{i} from matrix \textit{A}.

Argument type:  
Real matrix

Returns type:  
Real matrix (\textit{N}×1)

%xdiag(A)  —  Extract diagonal from a matrix
Extracts the diagonal from matrix \textit{A}.

Argument type:  
Real matrix

Returns type:  
Real matrix (\textit{N}×1)

%xrow(A, i)  —  Extract row
Extracts row \textit{i} from matrix \textit{A}.

Argument type:  
Real matrix

Returns type:  
Real matrix (1×\textit{N})

%xsubmat(A, startrow, endrow, startcol, endcol)  —  Extract a matrix
This extracts an \textit{M}×\textit{N} rectangular matrix from matrix \textit{A}. For example, to extract the upper-left 4×4 elements of \textit{A}, you would use \%xsubmat(A,1,4,1,4).

Argument type:  
\textit{A} matrix, and integer-valued row and column numbers

Returns type:  
Real matrix

%xsubvec(V, startrow, endrow)  —  Extract a vector
Extracts elements \textit{startrow} through \textit{endrow} of vector \textit{V}.

Argument type:  
\textit{V} is a Vector, \textit{startrow} and \textit{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

%zeros( rows, cols ) — Create a zero matrix
Returns a matrix with dimensions \( \text{rows} \times \text{cols} \), 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 \texttt{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
Appendix A: Expression Syntax and Operators

This appendix describes the operators supported in RATS and the basic elements of expression syntax. See Section 1.8 of the User’s Guide for more information on expressions, variables, and operators.

Arithmetic and Logical Operators

The table below lists the arithmetic and logical operators available in RATS, in order of precedence, from lowest to highest.

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>−</td>
<td>Negation (prefixing number or expression)</td>
</tr>
<tr>
<td>2</td>
<td>** or ^</td>
<td>Exponentiation (scalars or matrices)</td>
</tr>
<tr>
<td>2</td>
<td>.*</td>
<td>Elementwise exponentiation (matrix by scalar)</td>
</tr>
<tr>
<td>3</td>
<td>*/</td>
<td>Multiplication(*), division (/)</td>
</tr>
<tr>
<td>3</td>
<td>.*,./</td>
<td>Elementwise multiplication (.*), division (. /)</td>
</tr>
<tr>
<td>4</td>
<td>+,−</td>
<td>Addition(+), subtraction(− between expressions)</td>
</tr>
<tr>
<td>4</td>
<td>.+,.−</td>
<td>Elementwise addition(. +), subtraction (. −)</td>
</tr>
<tr>
<td>5</td>
<td>&gt;= or .ge.</td>
<td>Is greater than or equal to</td>
</tr>
<tr>
<td>5</td>
<td>&gt; or .gt.</td>
<td>Is greater than</td>
</tr>
<tr>
<td>5</td>
<td>&lt;= or .le.</td>
<td>Is less than or equal to</td>
</tr>
<tr>
<td>5</td>
<td>&lt; or .lt.</td>
<td>Is less than</td>
</tr>
<tr>
<td>5</td>
<td>== or .eq.</td>
<td>Is equal to (note distinction between the == and = operators)</td>
</tr>
<tr>
<td>5</td>
<td>!=, &lt;&gt; or .ne.</td>
<td>Is not equal to</td>
</tr>
<tr>
<td>6</td>
<td>.not.</td>
<td>Logical negation</td>
</tr>
<tr>
<td>7</td>
<td>.and.</td>
<td>Logical “and”</td>
</tr>
<tr>
<td>8</td>
<td>.or.</td>
<td>Logical “or”</td>
</tr>
<tr>
<td>9</td>
<td>~</td>
<td>Horizontal concatenation of two arrays</td>
</tr>
<tr>
<td>9</td>
<td>~~</td>
<td>Vertical concatenation of two arrays</td>
</tr>
<tr>
<td>9</td>
<td>~\</td>
<td>Diagonal concatenation of two arrays</td>
</tr>
<tr>
<td>10</td>
<td>=</td>
<td>Assignment (copy object on right to object on left)</td>
</tr>
<tr>
<td>10</td>
<td>*=</td>
<td>Multiply and assign (a <em>= b same as a = a</em>b)</td>
</tr>
<tr>
<td>10</td>
<td>/=</td>
<td>Divide and assign (a /= b same as “a = a/b”)</td>
</tr>
<tr>
<td>10</td>
<td>+=</td>
<td>Increment and assign (a += b same as “a= a+b”)</td>
</tr>
<tr>
<td>10</td>
<td>-=</td>
<td>Decrement and assign (a -= b same as “a = a−b”)</td>
</tr>
</tbody>
</table>
Appendix A: Expression Syntax

Operator Precedence

“Precedence” is the order in which RATS evaluates the operators in an expression. For example, consider the following instruction.

```
compute c = 100–5*5
```

Multiplication has a higher precedence than subtraction, so RATS first multiplies 5 times 5. Then, it subtracts this result from 100, returning a final result of 75.

When operators have the same precedence (the same value in the Precedence column in the table on the previous page), RATS evaluates them from left to right, with two exceptions:

- The exponentiation operators (**, ^, .^) are evaluated from right to left. For example, RATS will handle A**B**C as A**(B**C)
- In the absence of parentheses, RATS will evaluate the rightmost = sign (assignment operator) first, then move left.

Exponentiation takes precedence over negation in expressions of the form –A**B. For example, –A**2 is interpreted as –(A^2), rather than (–A)^2.

Using Parentheses

You can use parentheses to control the order in which RATS evaluates an expression. RATS will evaluate an expression contained in parentheses first, then move to operations outside the parentheses. If an expression includes nested parentheses (one set inside another), RATS evaluates the expression in the innermost set of parentheses first, then moves outward. For example:

```
1+2*3+4*5        = 27
(1+2)*3+4*5      = 29
1+((2*3)+4)*5    = 51
```

Arithmetic Operators and Complex Values

Some of the arithmetic operators function differently when applied to complex values:

- The +, -, =, and / operators function in the usual way.
- The * operator multiplies the first number by the conjugate of the second number. If A and B are complex, A*B=AB.
- The ** (and ^) operator functions as expected. However, you cannot raise a negative real number (zero imaginary part) to a power.
Relational Operators and Complex Values

You can use the == and <> operators to test the equality of two complex numbers. You cannot, however, use any of the other relational operators with complex numbers.

You can use logical operators in conjunction with the %REAL and %IMAG functions to compare the real or imaginary parts of a complex number. For example, if A and B are complex:

\[
\%\text{real}(a) \geq \%\text{real}(b)
\]

is a legal expression.

Label and String Operators

With LABEL and STRING variables, you can only use the +, =, ==, and <> operators. The + operator concatenates two labels or strings. For example, if LABEL1 and LABEL2 are LABEL variables, LABEL1+LABEL2 adds to the end of LABEL1 as much of LABEL2 as will fit, truncated at its last non-blank position:

```plaintext
compute [label] label1 = "Time"
compute [label] label2 = "Series"
compute [label] label3 = label1+label2
display label3
```

produces the output:

TimeSeries

You can also use a literal string of text (in quote marks) as part of a LABEL or STRING expression. For example:

```plaintext
compute [string] name = "Bob " + "Smith"
```
Appendix B: Reserved Names

Appendix B: List of Reserved Variable Names

This appendix lists the variable names reserved for use by RATS. You can also access a categorized list of these variables from within RATS, by selecting RATS Variables from the Wizards menu.

There are two types of reserved variables: accessible variables and special purpose variables:

Accessible variables are set by various RATS instructions. Examples include \%RSS (residual sum of squares) and \%BETA (vector of coefficients), which are defined by estimation instructions such as LINREG. You can use these variables in expressions just like any other variables. You cannot change the values of the scalar variables (such as \%RSS), but you can change the values of the array variables (such as \%BETA). Note that all of these variables begin with the \% character.

Special purpose variables serve a particular purpose. These are: the I and J integer variables used as EWISE subscripts; the integer T reserved for use as the time subscript in SET or FRML instructions; the CONSTANT series used for regressions with a constant term; \%MVGAVGE, used for adding a moving average lag to an equation; \%GARCHV, used for including a variance term in the mean equation in an ARCH or GARCH model; the constant \%PI; and the word TO, used for lists on loops, series lists, and in lag notation.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%BETA</td>
<td>VECTOR</td>
<td>Coefficient vector defined by estimation instructions.</td>
</tr>
<tr>
<td>%BETASYS</td>
<td>VECTOR</td>
<td>Full coefficient vector, defined by AR1 and others</td>
</tr>
<tr>
<td>%CDSTAT</td>
<td>REAL</td>
<td>Test statistic (hypothesis tests, CDF, STATISTICS).</td>
</tr>
<tr>
<td>%CMOM</td>
<td>SYMMETRIC</td>
<td>Cross-moment matrix defined by CMOM, MCOV.</td>
</tr>
<tr>
<td>%COEFF</td>
<td>RECTANGULAR</td>
<td>Matrix of equation coefficients defined by ALLOCATE.</td>
</tr>
<tr>
<td>CONSTANT</td>
<td>Special</td>
<td>Series name for constant term in regressions.</td>
</tr>
<tr>
<td>%CONVCRIT</td>
<td>REAL</td>
<td>Final convergence criterion. Will be set to zero if the subiterations limit was hit (non-linear estimations).</td>
</tr>
<tr>
<td>%CONVERGED</td>
<td>INTEGER</td>
<td>Variable set to 1 if a process converged, and 0 otherwise (non-linear estimations).</td>
</tr>
<tr>
<td>%CVCRIT</td>
<td>REAL</td>
<td>Final convergence criterion. Will be set to zero if the subiterations limit was hit (non-linear estimations).</td>
</tr>
<tr>
<td>%DURBIN</td>
<td>REAL</td>
<td>Durbin-Watson statistics. Computed by regressions.</td>
</tr>
<tr>
<td>%EBW</td>
<td>REAL</td>
<td>Equivalent band width (WINDOW, NPREG, DENSITY).</td>
</tr>
<tr>
<td>%EDF</td>
<td>REAL</td>
<td>Equivalent degrees of freedom (WINDOW).</td>
</tr>
<tr>
<td>%ERRCODE</td>
<td>INTEGER</td>
<td>The error code of the last error.</td>
</tr>
<tr>
<td>%ESALPHA</td>
<td>REAL</td>
<td>Alpha parameter estimated by ESMOOTH.</td>
</tr>
<tr>
<td>%ESDELTA</td>
<td>REAL</td>
<td>Delta parameter estimated by ESMOOTH.</td>
</tr>
<tr>
<td>%ESGAMMA</td>
<td>REAL</td>
<td>Gamma parameter estimated by ESMOOTH.</td>
</tr>
<tr>
<td>%FRACT01</td>
<td>REAL</td>
<td>1st percentile of a series (STATISTICS).</td>
</tr>
<tr>
<td>%FRACT05</td>
<td>REAL</td>
<td>5th percentile of a series (STATISTICS).</td>
</tr>
<tr>
<td>Variable</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>%FRACT10</td>
<td>REAL</td>
<td>10th percentile of a series (STATISTICS).</td>
</tr>
<tr>
<td>%FRACT25</td>
<td>REAL</td>
<td>25th percentile of a series (STATISTICS).</td>
</tr>
<tr>
<td>%FRACT75</td>
<td>REAL</td>
<td>75th percentile of a series (STATISTICS).</td>
</tr>
<tr>
<td>%FRACT90</td>
<td>REAL</td>
<td>90th percentile of a series (STATISTICS).</td>
</tr>
<tr>
<td>%FRACT95</td>
<td>REAL</td>
<td>95th percentile of a series (STATISTICS).</td>
</tr>
<tr>
<td>%FRACT99</td>
<td>REAL</td>
<td>99th percentile of a series (STATISTICS).</td>
</tr>
<tr>
<td>%FSIGNIF</td>
<td>REAL</td>
<td>Significance level of F statistic (estimation, testing)</td>
</tr>
<tr>
<td>%FSTAT</td>
<td>REAL</td>
<td>F statistic (estimation, testing instructions)</td>
</tr>
<tr>
<td>%FUNCVAL</td>
<td>REAL</td>
<td>Final function value (estimation instructions)</td>
</tr>
<tr>
<td>%GARCHV</td>
<td>SERIES</td>
<td>Within GARCH only, represents the series of variances.</td>
</tr>
<tr>
<td>%INTERACTIVE</td>
<td>INTEGER</td>
<td>0,1 indicator for interactive or batch mode</td>
</tr>
<tr>
<td>%ITERS</td>
<td>INTEGER</td>
<td>Number of iterations completed</td>
</tr>
<tr>
<td>I</td>
<td>INTEGER</td>
<td>I and J are used as index variables for EWISE. They can be used elsewhere, as in a DO loop index.</td>
</tr>
<tr>
<td>J</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>%JBSIGNIF</td>
<td>REAL</td>
<td>Jarque–Bera test significance level (STATISTICS).</td>
</tr>
<tr>
<td>%JBSTAT</td>
<td>REAL</td>
<td>Jarque–Bera test statistic (STATISTICS).</td>
</tr>
<tr>
<td>%JSIGNIF</td>
<td>REAL</td>
<td>Significance level for %JSTAT.</td>
</tr>
<tr>
<td>%JSTAT</td>
<td>REAL</td>
<td>Test statistic for Hansen J-test of overidentifying restrictions (instrumental variables estimations).</td>
</tr>
<tr>
<td>%KAPPA</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>%KURTOSIS</td>
<td>REAL</td>
<td>Excess kurtosis of a series (STATISTICS).</td>
</tr>
<tr>
<td>%LAGRANGE</td>
<td>VECTOR</td>
<td>VECTOR of Lagrange multipliers (non-linear estimations with constraints).</td>
</tr>
<tr>
<td>%LOGDET</td>
<td>REAL</td>
<td>Log determinant of the variance-covariance matrix (ESTIMATE, NLSYSTEM, SUR, VCV).</td>
</tr>
<tr>
<td>%LOGL</td>
<td>REAL</td>
<td>Value of the log-likelihood (most estimation).</td>
</tr>
<tr>
<td>%MAXENT</td>
<td>INTEGER</td>
<td>Entry containing max. value (EXTREMUM, CXT).</td>
</tr>
<tr>
<td>%MAXIMUM</td>
<td>REAL</td>
<td>Max. value of a series (EXTREMUM, STAT, CXT).</td>
</tr>
<tr>
<td>%MEAN</td>
<td>REAL</td>
<td>Mean value of a series (STATISTICS, regressions).</td>
</tr>
<tr>
<td>%MEANV</td>
<td>VECTOR</td>
<td>Vector of means of the explanatory variables (PRJ)</td>
</tr>
<tr>
<td>%MEDIAN</td>
<td>REAL</td>
<td>Median value of a series (STATISTICS).</td>
</tr>
<tr>
<td>%MENUCHOICE</td>
<td>INTEGER</td>
<td>Number of the menu item selected in a USERMENU.</td>
</tr>
<tr>
<td>%MENUSTATUS</td>
<td>INTEGER</td>
<td>0-1 status variable (0=Cancel) on a MENU/CHOICE.</td>
</tr>
<tr>
<td>%MINENT</td>
<td>INTEGER</td>
<td>Entry containing the min. value (EXTREMUM, CXT).</td>
</tr>
<tr>
<td>%MINIMUM</td>
<td>REAL</td>
<td>Minimum value of a series (EXTREM, STAT, CXT).</td>
</tr>
<tr>
<td>%MVGAVGE</td>
<td>Special</td>
<td>Special variable name used to add moving average terms to an equation (VADD, GARCH).</td>
</tr>
<tr>
<td>%NA</td>
<td>REAL</td>
<td>RATS code for a missing value.</td>
</tr>
<tr>
<td>%NARMA</td>
<td>INTEGER</td>
<td>Number of ARMA terms estimated (BOXJENK).</td>
</tr>
<tr>
<td>%NCMOM</td>
<td>INTEGER</td>
<td>Number of variables (CMOMENT, MCOV).</td>
</tr>
<tr>
<td>%NCORRECT</td>
<td>INTEGER</td>
<td>Number of cases correct (DDV).</td>
</tr>
<tr>
<td>%NDF</td>
<td>INTEGER</td>
<td>Num. of degrees of freedom (estimation instructions).</td>
</tr>
<tr>
<td>%NDFJ</td>
<td>INTEGER</td>
<td>Num. of degrees of freedom of J statistic.</td>
</tr>
<tr>
<td>%NDFQ</td>
<td>INTEGER</td>
<td>Num. of degrees of freedom of Q statistic.</td>
</tr>
</tbody>
</table>
### Appendix B: Reserved Names

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%NDFTEST</td>
<td>INTEGER</td>
<td>Num. of degrees of freedom of the test statistic.</td>
</tr>
<tr>
<td>%NFREE</td>
<td>INTEGER</td>
<td>Number of freely estimated parameters (estimation instructions).</td>
</tr>
<tr>
<td>%NMISS</td>
<td>INTEGER</td>
<td>Num. of skipped/missing observations (estimation instructions; others).</td>
</tr>
<tr>
<td>%NOBS</td>
<td>INTEGER</td>
<td>Num. of observations (estimation instruct., others).</td>
</tr>
<tr>
<td>%NREG</td>
<td>INTEGER</td>
<td>Num. of regressors (estimation instructions).</td>
</tr>
<tr>
<td>%NREGMEAN</td>
<td>INTEGER</td>
<td>Num. of regressors in mean model (GARCH).</td>
</tr>
<tr>
<td>%NREGSYSTEM</td>
<td>INTEGER</td>
<td>Num. of regressors in entire model (ESTIMATE).</td>
</tr>
<tr>
<td>%NVAR</td>
<td>INTEGER</td>
<td>Num. of equations estimated (ESTIMATE, GARCH).</td>
</tr>
<tr>
<td>%PI</td>
<td>REAL</td>
<td>The constant π.</td>
</tr>
<tr>
<td>%PRIOR</td>
<td>RECTANGULAR</td>
<td>The prior matrix for a VAR (SPECIFY, END(SYSTEM))</td>
</tr>
<tr>
<td>%PRJCDF</td>
<td>REAL</td>
<td>Predicted probability from ATMEAN or XVECTOR (PRJ)</td>
</tr>
<tr>
<td>%PRJDENSITY</td>
<td>REAL</td>
<td>Predicted density from ATMEAN or XVECTOR (PRJ)</td>
</tr>
<tr>
<td>%PRJFIT</td>
<td>REAL</td>
<td>Fitted value from ATMEAN or XVECTOR options (PRJ)</td>
</tr>
<tr>
<td>%QSIGNIF</td>
<td>REAL</td>
<td>Significance level of %QSTAT (estimation instructions).</td>
</tr>
<tr>
<td>%QSTAT</td>
<td>REAL</td>
<td>Ljung–Box Q-statistic (estimation instructions).</td>
</tr>
<tr>
<td>%RBARSQ</td>
<td>REAL</td>
<td>Adjusted R-squared statistic (regression instructions).</td>
</tr>
<tr>
<td>%REPORTCOL</td>
<td>INTEGER</td>
<td>Last column filled in a REPORT.</td>
</tr>
<tr>
<td>%REPORTROW</td>
<td>INTEGER</td>
<td>Last row filled in a REPORT.</td>
</tr>
<tr>
<td>%REPORTTAGCOL</td>
<td>INTEGER</td>
<td>Last column tagged in a REPORT.</td>
</tr>
<tr>
<td>%REPORTTAGROW</td>
<td>INTEGER</td>
<td>Last row tagged in a REPORT.</td>
</tr>
<tr>
<td>%RESIDS</td>
<td>SERIES</td>
<td>Series of residuals (RREG)</td>
</tr>
<tr>
<td>%RHO</td>
<td>REAL</td>
<td>First lag autocorrelation coefficient (all single-equation estimation instructions).</td>
</tr>
<tr>
<td>%RSQUARED</td>
<td>REAL</td>
<td>Centered R-squared statistic (regressions).</td>
</tr>
<tr>
<td>%RSS</td>
<td>REAL</td>
<td>Residual sum of squares (regressions).</td>
</tr>
<tr>
<td>%SCALETAP</td>
<td>REAL</td>
<td>Scale factor for spectral estimators (TAPER).</td>
</tr>
<tr>
<td>%SEESQ</td>
<td>REAL</td>
<td>Standard error of estimate squared (regressions).</td>
</tr>
<tr>
<td>%SIGMA</td>
<td>SYMMETRIC</td>
<td>Covariance matrix of resids (ESTIMATE, SUR, NLSYS).</td>
</tr>
<tr>
<td>%SIGMASQ</td>
<td>REAL</td>
<td>Maximum likelihood estimate of variance (many estimation instructions).</td>
</tr>
<tr>
<td>%SIGNIF</td>
<td>REAL</td>
<td>Significance level of %CDSTAT (hypothesis testing instructions, STATISTICS, CDF).</td>
</tr>
<tr>
<td>%SKEWNESS</td>
<td>REAL</td>
<td>Skewness of a series (STATISTICS).</td>
</tr>
<tr>
<td>%STDERRS</td>
<td>VECTOR</td>
<td>Vector of standard errors (estimation instructions).</td>
</tr>
<tr>
<td>%SUMLC</td>
<td>REAL</td>
<td>Value of linear combination (SUMMARIZE)</td>
</tr>
<tr>
<td>T</td>
<td>INTEGER</td>
<td>Time subscript used in SET and FRML instructions. In general, should not be used for other purposes.</td>
</tr>
<tr>
<td>%THEIL</td>
<td>VECT[RECT]</td>
<td>Vector of rectangular arrays containing the forecast statistics produced by the THEIL instruction.</td>
</tr>
<tr>
<td>TO</td>
<td>Special</td>
<td>Reserved word used in lists of consecutive integers (e.g., list of lags such as {1 to 4}, or a list of consecutively-numbered series.)</td>
</tr>
</tbody>
</table>
### Appendix B: Reserved Names

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%TRSQ</td>
<td>REAL</td>
<td>Number of observations times the raw (uncentered) R-squared statistic (regressions).</td>
</tr>
<tr>
<td>%TRSQUARED</td>
<td>REAL</td>
<td>Number of observations times the centered R-squared statistic (regressions).</td>
</tr>
<tr>
<td>%TSTATS</td>
<td>VECTOR</td>
<td>Vector of coefficient t-stats (estimation instructions).</td>
</tr>
<tr>
<td>%UZWZU</td>
<td>REAL</td>
<td>$u^\prime Z W Z u$ for instrumental variables (LINREG, NLLS, NLSYS, SUR).</td>
</tr>
<tr>
<td>%VARIANCE</td>
<td>REAL</td>
<td>Sample variance of a series (STATISTICS, regressions)</td>
</tr>
<tr>
<td>%VARLAGSUMS</td>
<td>RECTANGULAR</td>
<td>Matrix containing the sums of the lag coefficients for a VAR (ESTIMATE).</td>
</tr>
<tr>
<td>%VARLC</td>
<td>REAL</td>
<td>Variance of linear combination of coeffs (SUMMARIZE).</td>
</tr>
<tr>
<td>%VINDIV</td>
<td>REAL</td>
<td>Variance of individual component (PSTATS, PREG).</td>
</tr>
<tr>
<td>%VRANDOM</td>
<td>REAL</td>
<td>Variance of random component (PSTATS, PREG).</td>
</tr>
<tr>
<td>%VTIME</td>
<td>REAL</td>
<td>Variance of time component (PSTATS, PREG).</td>
</tr>
<tr>
<td>%WMATRIX</td>
<td>SYMMETRIC</td>
<td>Final weight matrix for GMM (LINREG, NLLS)</td>
</tr>
<tr>
<td>%X</td>
<td>RECTANGULAR</td>
<td>Array of real series defined by ALLOCATE.</td>
</tr>
<tr>
<td>%XX</td>
<td>SYMMETRIC</td>
<td>Covariance matrix of coefficients, or $X'X^{-1}$ for linear regressions (estimation instructions).</td>
</tr>
<tr>
<td>%XXSYS</td>
<td>SYMMETRIC</td>
<td>Full covariance matrix (AR1, others)</td>
</tr>
<tr>
<td>%Z</td>
<td>CMATRIX</td>
<td>Array of complex series defined by FREQUENCY.</td>
</tr>
</tbody>
</table>
Appendix C: 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 shown 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
Appendix C: Error Messages

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)”.


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?
Appendix C: Error Messages

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 Unex-
   pected.
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 "<<num-
   ber>>"
Appendix C: Error Messages

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 SOURCEing 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>>
Appendix C: Error Messages

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)
Appendix C: Error Messages

**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 FRMLs, 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
Appendix C: Error Messages

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 <=)

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.
Appendix C: Error Messages

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 Proce-
       dure?
CP22  <<name>> may not be used in a compiled section
CP23  Option abbreviation <<abbrev>> already in use in procedure <<name>>

Miscellaneous Errors Relating to Specific Instructions

X1    Syntax is CALENDAR(options) year month day with this fre-
       quency
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"
Appendix C: Error Messages

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]
Appendix C: Error Messages

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.
Appendix C: Error Messages

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.
Appendix D: Instructions Listed by Task

In this appendix, we’ve listed all of the RATS instructions, grouped by their function. Some instructions are listed in more than one category.

Regression and Estimation Instructions

**AR1**
Estimates regressions with first-order autocorrelation correction

**BOXJENK**
Estimates ARIMA, transfer function, and intervention models

**CMOMENT**
Computes cross-moment and correlation matrices

**CVMODEL**
Models a covariance matrix

**DDV**
Performs discrete dependent variable estimation (logit, probit)

**DLM**
Includes techniques for analyzing dynamic linear models

**FIND**
General maximization or minimization

**GARCH**
Estimates univariate and multivariate ARCH and GARCH

**INSTRUMENTS**
Sets the list of instrumental variables

**LDV**
Performs limited dependent variable estimation

**LINREG**
Estimates linear regressions

**LQPROG**
Solves linear and quadratic programming problems.

**MAXIMIZE**
General maximum-likelihood estimation

**MCOV**
Computes consistent covariance matrices

**NLLS**
Estimates single-equation non-linear least squares regressions

**NLPAR**
Controls the parameters of non-linear estimations

**NLSYSTEM**
Estimates non-linear systems of equations

**NNLEARN**
Fits neural network models

**NNTTEST**
Generates output from neural network models

**NONLIN**
Sets the free parameters for non-linear estimation.

**NPREG**
Performs non-parametric regressions.

**PREGRESS**
Panel data regressions, such as fixed and random effects

**RLS**
Estimates by recursive least squares.

**RREG**
Estimates by robust regression (LAD, quantile)

**STWISE**
Estimates stepwise regressions

**SUR**
Estimates linear systems of equations

**Sweep**
Regresses a set of “targets” on a set of “instruments”.

Forecasting

**BOOT**
Randomization for bootstrapping

**ERRORS**
Decomposition of forecast variance

**ESMOOTH**
Exponential smoothing

**FORECAST**
Computes dynamic forecasts

**HISTORY**
Historical decompositions

**PRJ**
Computes fitted values and normal distribution statistics

**SIMULATE**
Generates random simulations of a model

**STEPS**
Computes static forecasts

**THEIL**
Computes forecast performance statistics

**UFORECAST**
Computes forecasts or simulations for univariate model—generally more convenient to use than **FORECAST** or **STEPS**
Appendix D: Instructions Listed by Task

Box-Jenkins Forecasting

- **BOXJENK** Estimates ARIMA, transfer function, and intervention models
- **CORRELATE** Computes the autocorrelations of a series
- **INITIAL** Solves Yule–Walker equations (initial estimates of ARMA models)

Vector Autoregressions

- **CVMODEL** Covariance matrix modelling
- **DETERMINISTIC** Specifies the “other” variables in a VAR
- **ECT** Specifies structure for error correction models
- **ERRORS** Computes a decomposition of forecast variance
- **ESTIMATE** Estimates a VAR system
- **HISTORY** Computes historical decompositions
- **IMPULSE** Computes impulse response functions
- **KALMAN** Kalman filtering
- **KFSET** Sets up the Kalman filter
- **LAGS** Specifies the lags for a VAR
- **SPECIFY** Specifies Bayesian priors for vector autoregressions
- **SYSTEM** Sets up a VAR system
- **TVARYING** Sets up time-varying coefficients for the Kalman filter
- **VARIABLES** Specifies the variables in a VAR

Equations, Formulas, and Models

- **ASSOCIATE** Assigns coefficients to equations
- **EQUATION** Defines linear equations
- **FRML** Defines a (possibly non-linear) formula
- **GROUP** Groups equations and formulas to form a model.
- **MODIFY** Modifies a single equation
- **NONLIN** Sets the free parameters for non-linear estimations
- **VADD** Adds a variable to an equation (used with MODIFY)
- **VREPLACE** Replaces a variable in an equation (used with MODIFY)

Hypothesis Testing and Restricted Regressions

- **ENCODE** Initial step for estimating restricted regressions
- **EXCLUDE** Tests exclusion restrictions
- **MRESTRICT** Tests general restrictions using matrices for input
- **PSTATS** Performs panel data analysis of variance tests
- **RATIO** Tests in multiple equation systems
- **RESTRICT** Tests or imposes general linear restrictions
- **SUMMARIZE** Computes coefficient sums
- **TEST** Tests specific coefficient values
Appendix D: Instructions Listed by Task

Panel Data

**PANEL**
Does standard panel data transformations

**PFORM**
Converts data into the panel format supported by RATS.

**PREGRESS**
Performs panel data regressions, such as fixed and random effects

**PSTATS**
Does analysis of variance tests and variance decompositions

Statistics

**CDF**
Computes marginal significance levels

**CROSS**
Computes cross-correlations

**DENSITY**
Estimates the density function of a series

**MVSTATS**
Computes moving statistics and fractiles

**PRJ**
Computes fitted values/normal distribution statistics

**PSTATS**
Computes analysis of variance statistics on panel data

**SSTATS**
Computes statistics accumulated over the entries of series

**STATISTICS**
Computes basic statistics for a single series

**TABLE**
Computes a table of statistics for a set of series

**VCV**
Computes a residual covariance matrix

Working with Data Series

**ACCUMULATE**
Computes partial sums

**ALLOCATE**
Sets default series length

**CALENDAR**
Sets the date and frequency information

**CLEAR**
Clears an existing data series or creates a new series

**DATA**
Reads data series from I/O unit

**DIFFERENCE**
Performs general time series differencing

**EQV**
Attaches names to numbered series

**EXP**
Takes the anti-log of a series

**EXTREMUM**
Locates the extreme values of a series

**FILTER**
Does general linear filtering

**GSET**
Does element-wise operations on series whose elements aren’t real or complex numbers.

**LABELS**
Sets output labels for series

**LOG**
Takes the log of a series

**ORDER**
Sorts or ranks data series

**PANEL**
Transformations of panel data series

**PFORM**
Converts data into the panel format supported by RATS.

**PRINT**
Prints data series to the output unit

**SAMPLE**
Extracts data from and changing frequency of data series

**SEASONAL**
Creates seasonal dummies

**SET**
Does general data transformations

**SMP**
Sets the default entry range

**X11**
X11 seasonal adjustment procedure (Pro version only)
Appendix D: Instructions Listed by Task

Matrix Operations and Calculations

- **CMOMENT**: Computes cross-moment and correlation matrices
- **COMPUTE**: Primary instructions for matrix calculations and operations
- **EIGEN**: Eigen decompositions of symmetric or rectangular matrices
- **GSET**: Element-wise operations on series whose elements are matrices
- **MAKE**: Creates an array from data series
- **QZ**: Computes QZ (generalized Schur) decomposition
- **VCV**: Computes a residual covariance matrix

Scalar and Array Variables

- **COMPUTE**: Performs general scalar and array computations
- **DECLARE**: Sets the data type of variables
- **DIMENSION**: Sets the size of arrays
- **EWISE**: Performs elementwise operations on matrices
- **FMATRIX**: Creates a matrix from a filter
- **INPUT**: Free-format input for variables
- **MAKE**: Creates an array from data series
- **READ**: General information input

RATS Format Files

- **CATALOG**: Lists the contents of a RATS format file
- **DATA**: Reads data series from a file
- **DEDIT**: Opens a RATS format file for editing
- **DELETE**: Removes series from a RATS format file
- **EDIT**: Edits a data series on screen
- **ENVIRONMENT**: Opens a RATS format file for reading and writing
- **INCLUDE**: Adds series to a RATS format file
- **PRTDATA**: Prints series stored on an open RATS format file
- **QUIT**: Aborts editing of a RATS format file
- **RENAME**: Renames series on a RATS format file
- **SAVE**: Saves changes to a RATS format file
- **STORE**: Stores data series on a RATS format file
- **UPDATE**: Alters data on a RATS format file

General File Operations

- **CHANGE**: Switches I/O units
- **CLOSE**: Closes an open I/O unit
- **OPEN**: Opens files for input or output
- **REWIND**: Rewinds to the beginning of a file
- **SOURCE**: Opens another file of RATS instructions. Used primarily with procedures.
Appendix D: Instructions Listed by Task

Displaying Information
- **COPY** Copies data series to screen, file, or printer
- **DBOX** Allows definition of custom dialog boxes
- **DISPLAY** Displays text, variables, and expression results
- **INFOBOX** Displays informational dialog boxes and progress bars
- **MEDIT** Allows spreadsheet-style viewing and editing of matrices
- **MESSAGEBOX** Displays simple informational dialog boxes
- **PRINT** Prints data series to the output unit
- **REPORT** Generates formatted reports as text or in a spreadsheet window.
- **WRITE** Prints matrices and other variables

Graphics
- **GCONTOUR** Generates contour plots
- **GRAPH** Generates time series graphs
- **GRPARM** Sets parameters for time-series and scatter graphs
- **GRTEXT** Adds text strings to a graph
- **SCATTER** Generates x-y scatter plots
- **SPGRAPH** Controls special graphs functions, including putting multiple graphs on a single page

Looping and Program Control
- **BRANCH** Branches to a specific location
- **BREAK** Breaks out of a loop
- **DBOX** Allows definition of custom dialog boxes
- **DO** Loops over an index
- **DOFOR** Loops over a list
- **END** Ends a RATS program
- **HALT** Stops execution from within a compiled section
- **IF** and **ELSE** Executes one or more instructions under specified conditions
- **INFOBOX** Displays informational dialog boxes and progress bars
- **LOOP** Loops indefinitely
- **MEDIT** Allows spreadsheet-style viewing and editing of matrices
- **MENU, CHOICE** Creates simple user-defined menus
- **MESSAGEBOX** Displays informational dialog box and waits for user response
- **NEXT** Moves to top of loop block
- **SELECT** Allows selection from a list of series, strings or integers
- **UNTIL** Conditional looping
- **USERMENU** Creates pull-down menus that control program execution
- **WHILE** Conditional looping
Appendix D: Instructions Listed by Task

User-Defined Procedures and Functions

- **ENTER**: Creates user-defined supplementary cards for procedures
- **EXECUTE**: Executes (calls) a RATS procedure
- **FIXED**: Creates and sets fixed-value arrays in a procedure
- **FUNCTION**: Creates a user-defined function
- **LOCAL**: Declares local variables in a procedure or function
- **OPTION**: Defines options for a RATS procedure
- **PROCEDURE**: Defines a RATS procedure
- **RETURN**: Returns from a procedure (ends the procedure)
- **SOURCE**: Opens a file containing one or more RATS procedures
- **TYPE**: Sets the data type of procedure or function parameters

Frequency Domain Analysis

- **CACCUMULATE**: Computes partial sums of complex series
- **CADD**: Adds two complex series
- **CDIVIDE**: Divides two complex series
- **CEXP**: Takes the anti-log of a complex series
- **CLABELS**: Assigns labels to complex series
- **CLN**: Takes the log of a complex series
- **CMASK**: Creates a complex seasonal mask
- **CMULTIPLY**: Multiplies one complex series by the complex conjugate of another
- **CMVE**: Copies entries of complex series
- **CONJUGATE**: Conjugates a complex series
- **CPRINT**: Prints complex series
- **CSAMPLE**: Reduces the number of frequencies of a complex series
- **CSCALE**: Scales a complex series by a constant
- **CSET**: Does general transformations of complex series
- **CSQRT**: Takes the square root of a complex series
- **CSUBTRACT**: Subtracts two complex series
- **CTOR**: Copies (one part) of a complex series into a real-valued series
- **CXTREMUM**: Finds the extreme values of a complex series
- **FFT**: Computes Fourier transforms
- **FOLD**: Folds a spectrum (computes implied spectral density function)
- **FREQUENCY**: Initializes frequency domain analysis (required before working with complex series)
- **IFT**: Computes inverse Fourier transforms
- **POLAR**: Computes the polar decomposition of a series
- **RTOC**: Copies real-valued series to complex series
- **TAPER**: Tapers complex series
- **TRFUNCTION**: Computes the transfer function of a lag polynomial
- **WINDOW**: Smooths spectral estimates
## Miscellaneous Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNTRL</td>
<td>Controls interactive procedures</td>
</tr>
<tr>
<td>DEBUG</td>
<td>Tools for debugging procedures and RATS itself</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>Changes default option settings for RATS instructions</td>
</tr>
<tr>
<td>ENVIRONMENT</td>
<td>Sets error modes and other options</td>
</tr>
<tr>
<td>INQUIRE</td>
<td>Returns information from RATS (including length of series, dimensions of arrays, etc.)</td>
</tr>
<tr>
<td>LIST, CARDS</td>
<td>Simplifies use of multiple supplementary cards</td>
</tr>
<tr>
<td>QUERY</td>
<td>Requests input from the user</td>
</tr>
<tr>
<td>RELEASE</td>
<td>Releases segments of memory</td>
</tr>
<tr>
<td>SEED</td>
<td>Initializes the random number generator</td>
</tr>
<tr>
<td>SHOW</td>
<td>Displays amount of free memory, a list of series in memory, or a list of files</td>
</tr>
</tbody>
</table>
Bibliography


Bibliography


used in multinomial logit, UG528. used with SET, UG51.
%IMAG function, UG558.
%INDIV function, UG544.
%INNERXX function, UG161, UG164.
%JDF variable, UG280.
%JSSIGNIF variable, UG280.
%JSTAT variable, UG280.
%JULIAN function, UG168.
%KRONEKER function, UG161, UG164.
%KRONID function, UG161, UG164.
%KRONMULT function, UG161, UG164.
%L function, UG118, UG151, UG155.
%LABEL function, UG155.
%LEFT function, UG154.
%LOG function, UG160.
%LOGDET variable, UG347.
%MAXENT variable, RM170.
%MAXIMUM variable, RM170.
%MINIMUM variable, RM170.
%MODELAGSUMS function, UG358.
%MONTH function, UG168.
%MQFORM function, UG161, UG164, UG211.
%MQFORMDIAG function, UG161, UG164.
%MSCALAR function, UG160.
%MVECTOR function, UG155.
%NA missing value, UG52.
%NA variable, UG52.
%NA testing for, UG52.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
%OBS FUNCTION, UG555.
Abbreviating instruction names, UG60.
option names, UG61.
AFLAGS procedure, UG555.
Absolute value function, RM520.
complex numbers, RM523.
integers, RM535.
ACCUMULATE instruction, RM2.
ADAPTIVE.PRG example, UG462.
Add factors, UG414.
Adjusted R-squared, UG177.
AIC. See Akaike criterion.
Akaike criterion, UG203, UG320.
BJEST procedure, UG332.
example, UG204.
for VARs, UG344, UG348.
Akaike, H., UG203.
AKAIKE.PRG example, UG204.
Alert
dialog box, RM304.
Algorithms
BFPS, UG261, UG286.
BHII, UG261, UG286.
Cochrane-Orcutt, UG404.
Gauss-Seidel, UG411.
generic, UG263, UG286.
Hildreth-Lu, UG404.
Levenberg-Marquardt, RM319.
Newton-Raphson, UG258.
random numbers, RM417.
simplex, UG262, UG286, UG293.
ALLOCATE instruction, RM4, UG8.
for cross-sectional data, UG524.
for panel data, UG543.
function returning length, RM520.
using series parameter, UG152.

Almon lags, UG202.
Analysis of variance (panel data), RM366, UG548.
Andrews,D., UG576.
Annual data, RM28.
Annuity functions
payments, RM547.
present value, RM520.
Appending data to a file, RM339.
AR1
fitted values, RM10.
forecasts, RM10, UG301.
AR1 instruction, RM6, UG43, UG189.
FRML option, UG409.
instrumental variables, RM9,
RM257.
with panel data, UG550.
simultaneous equation
models, UG404.
AR1.PRG example, UG191.
Arc cosine, RM520.
Arc sine, RM521.
Arc tangent, RM521.
ARCH models, UG417.
example, UG427.
GARCH instruction, RM206.
Markov switching, UG458.
testing for, UG227.
ARCH/GARCH Wizard, RM206.
Archiving data, RM364.
ARCHTEST procedure, UG227.
Arellano-Bond estimator, UG555.
ARELLANO.PRG example, UG555.
ARFIMA models, UG576.
Argument function
of complex number, RM520.
ARIMA models, RM17, UG314, UG319.
with BOXJENK, RM17.
defining with EQUATION, RM142.
example, UG320.
forecasting, UG314, UG319.
initial estimates using
INITIAL, RM250.
intervention terms, UG336.
non-consecutive lags, RM18.
procedures for, UG329.
residual autocorrelations, UG332.
selecting, UG319.
ARIMA.PRG example, UG320.
Arithmetic
expressions, RM569, UG52.
operators, RM569.
complex values, UG558.
ARMAX models, UG328.
Arrays, UG149. See also Matrix.
ASSOCIATE instruction, RM12.
with EQUATION, RM142.
Autocorrelations, RM58, UG319.
correction
higher-order, RM11.
with AR1, RM6, UG43.
graphing, UG129.
inverse, RM58, RM60.
Complex numbers
- absolute value of, RM523.
- argument of, RM520.
- conjugate, RM525.
- creating, RM525.
- exponentiation, RM523.
- expressions using, UG558.
- imaginary part of, RM535.
- log function, RM524.
- matrix adjoint, RM527.
- matrix inverse, RM527.
- operators on, RM571.
- real part of, RM554.
- singular value decomposition, RM527.
- square root, RM526.

Complex series, UG559.
- %Z variable, RM71.
- converting from real, RM400.
- converting to real, RM73.
- creating, RM71, RM400.
- displaying, UG564.
- Fourier transforms, RM171.
- graphing, RM73, UG564.
- labeling, RM39.
- padding, UG563.
- preparation, UG562.
- printing, RM64.
- reading in, RM401.

COMPUTE instruction, RM50, UG36, UG136, UG143.
- assigning data types, UG137.
- alternative to INPUT, RM252.
- with FORTRAN format, UG95.
- with expressions, UG55.
- non-linear, UG264, UG271.
- taper, RM471.
- of strings, RM525, UG118, UG154.
- of matrices, RM522, UG158.
- of series, RM66.
- of arrays, RM526.
- auto, RM58.
- correlations from, RM527.
- Covariance matrix modelling, RM75.
- of coefficients, RM276, UG211.
- consistent estimate of, RM295.
- of series, RM496.
- of series, RM496.
- VAR (via KFSET), RM263.
- Covariance Matrix wizard, RM496.
- Covariance/correlation matrix, UG179.
- Cox, D. R., RM294.
- CPRINT instruction, RM64.
- CREATE option
  - with LINREG, UG212.
  - with RESTRICT and MRESTRICT, UG206.
- Cross Correlations wizard, RM66.
- CROSS instruction, RM66.
- Cross sectional data, UG524.
- Cross-correlations, RM66.
- Cross-equation restrictions, RM457.
- Cross-moment matrix computing, RM44.
- Cross-periodograms, RM502.
- Cross-spectral analysis, RM352, UG566.
- CROSSPEC procedure, UG568.
- CSAMPLE Instruction, RM70.
- SCALE instruction, RM41.
- CSET instruction, RM71, UG559.
- CSQRT instruction, RM41.
- CSUBTRACT instruction, RM48.

CTOR Instruction, RM73.
- CTO instruction, RM73.
- CUMPDGM procedure, UG231.
- Cumulated periodogram test, RM26, UG231.
- CUSUM test, RM393, UG238.
- example, UG125.
- CVMODEL instruction, RM75, UG355, UG360.
- CVMODEL.PRG example, UG370.
- DAILY instruction, RM26, RM79.

D
- Daily data, RM28.
- removing gaps, RM402.
- DAMP option, UG185.
- Data. See also Series.
- archiving with PRTDATA, RM364.
- changing frequency of, RM32.
- editing, RM130.
- interpolating, RM32.
- opening files containing, UG10.
- reading from a file, UG10, UG70.
- setting frequency of, RM27.
- typing in, UG72.
- writing to a file, UG70.
- Data (RATS Format) wizard, UG81.
- DATA I/O unit, RM339.
- DATA instruction, RM80, UG10, UG39, UG70.
- binary data, UG103.
- changing data frequencies, UG76.
- CITIBASE format, UG99.
- DRI format, UG99.
- FORTRAN (text) format, UG96.
- free-format (text) files, UG92.
- HAVER format, UG99.
- with holidays omitted, UG80.
- PORTABLE format files, UG102.
- spreadsheet files, UG87.
- text files, UG93.
- troubleshooting, UG89.
- Data transformations, RM420, UG17.
- Data types, UG136, UG143.
- basic, UG146.
- declaring variables, RM99.
- Data wizards, RM80, UG12.
- Date/time stamp, RM527.
- Dates. See also Calendar functions.
- arithmetic using, UG137.
- of entries, RM528.
- entry number of, RM523.
- in expressions, UG137.
- functions, RM524.
- label function, RM528.
- specifying, RM29.
- on spreadsheet files, UG86, UG89.
- Davidson-MacKinnon test, UG241.
- Davis, R. A., UG319.
- Day of week, RM529, RM567.
- dBase files, UG73, UG90.
- DBOX instruction, RM83, UG145.
Index

DDV instruction, RM92, UG525, UG528, UG532, UG539.
DEBUG Instruction, RM98.
DECLARE instruction, RM99, UG144, UG145.
Decomposition
Blanchard-Quah, RM522, UG358.
Choleski, RM528, UG352.
eigen, RM133, RM530.
complex matrix, RM527.
error components, RM366.
historical, RM233, UG372.
QR, RM551.
QZ, RM371.
Schur, RM371.
singular value, RM560, UG162.
complex matrix, RM527.
structural, UG353.
of variance, RM147, RM152, UG385.
DEDIT instruction, RM101, UG82.
with old RATS format files, UG84.
saving changes after, RM404.
Default
entry range, RM427, UG11, UG64.
options, RM103.
DEFAULT instruction, RM103.
DEFINE option, UG47.
Degrees of freedom
correcting for, UG217.
VAR with a prior, UG382.
DELETE instruction, RM104, UG82.
Delta method
with SUMMARIZE, RM455.
DeLurgio, S., UG319, UG337.
Density
beta distribution, RM540.
chi-squared, RM523.
non-central, RM524.
empirical, RM105.
GED distribution, RM541.
generalized Pareto, RM541.
GEV distribution, RM541.
negative binomial, RM541, RM546.
normal, RM528.
multivariate, RM540.
at random draw, RM552.
t distribution, RM563.
multivariate, RM541.
non-central, RM563.
DENSITY instruction, RM105.
UG460.
Dent, W., UG232.
Dependent variables
of equation, RM142, RM530.
of MODEL, RM544.
in a VAR, RM495.
DESEASON.PRG example, UG571.
Determinant, RM528, UG160.
DETERMINISTIC
instruction, RM108, UG342, UG387.
DPC option, UG217.
DFUNIT procedure, UG249, UG254.
Diagonal matrix, RM529.
Dialog boxes
INFOBOX, RM248.
MESSAGEBOX, RM304.
user-defined, UG598.

Dikey, D., UG249.
Dickey-Fuller unit root test, UG249.
Diebold-Mariano test, UG309.
DIEBP210.PRG example, RM488.
DIEBP228.PRG example, UG125.
DIF files, UG85.
DIFFERENCE instruction, RM109, UG320.
Differencing, RM109, UG50.
fractional, RM109, UG576.
matrix operator, RM185.
 prior to forecasting, UG315.
 prior to VAR models, UG343, UG387.
seasonal, RM109.
Differencing wizard, RM109, UG19.
Digamma function, RM529.

DIMENSION instruction, RM112, UG145, UG148.
Dimensioning arrays, RM112.
Discrete dependent variables, RM92, UG525.
DISPLAY instruction, RM113, UG36, UG140, UG145.
Displaying
arrays, RM505.
calculations, UG140.
complex series, UG564.
data series, RM356.
series, UG16.
DISTRIB procedure, RM32, UG77.
Distributed lags, UG201.
graphing, UG201.
lag length, UG203.
polynomial, RM135, UG207.
sums of coefficients, RM453.
Distributions
chi-squared, RM36.
extreme value, RM93, RM360.
UG525, UG526.

F, RM36.
GED (generalized error), RM206.
logistic, RM360.
logit, RM93.
normal, RM36.
prob, RM93.
t, RM36, UG419.
Wishart, UG492.

Division
modulo, RM543.

DLM, RM116, UG436.
forecasting with, UG314.
stationary covariance matrix, RM549.

DLM instruction, RM116.
DMARIANO procedure, UG309.
DO instruction, RM124, UG56, UG581.
DO loops
EWISE as alternative to, RM163.
Doan, T., RM264, UG343, UG392, UG393.
DOFOR instruction, RM126, UG56, UG581.
Dot product, RM529.
DRI database, UG74, UG99.
Dummy observation prior, UG396.
Dummy variables, UG49.
Chow tests, UG237.
panel data, UG545.
seasonal, RM413.
for subsamples, RM427, UG181.
Duration analysis, UG540.
Durbin h statistic, UG230.
Durbin, J., RM26, UG230.
Durbin-Watson statistic, UG27, UG178, UG220.
%DURBIN variable, RM276.
with missing observations, UG182.
Dynamic forecasts, RM188, UG34, UG299.
Dynamic Linear Models. See DLM.

Easter
date of, RM529.
Echoing of input lines, RM140.
ECT instruction, UG374.
ECT.PRG example, UG376.
EDIT instruction, RM130, UG82.
EGTEST procedure, UG253, UG254.
EIGEN instruction, RM133, UG157.
Eigenvalues/vectors, RM133, RM530.
ELSE instruction, RM237.
ENCODER instruction, RM135, UG202, UG205.
END instruction, RM137, UG57.
with needed, UG582.
END(SYSYSTEM) instruction, UG382.
Enders, W., UG252, UG319, UG341, UG353, UG374, UG416.
Endogenous variables
in a VAR, RM267, RM495.
swapping, UG410.
Engle, R., UG227, UG252, UG416.
UG421, UG428, UG486, UG573.
Engle-Granger test, UG253.
ENTER instruction, RM138, UG145.
ENTRIES option, UG217.
Enter numbers, UG9.
using dates, RM29.
in expressions, UG137.
Enter ranges, UG8, UG11, UG64.
setting default, RM427.
slipping observations within, UG180.
specifying with SMPL option, UG180.
ENVIRONMENT instruction, RM140.
Index

EQUATION data type, UG143, UG147.
EQUATION instruction, RM142, UG145.
Equation/FRML wizard, RM142, RM196.
Equations, RM142, UG47, UG147.
adding variables to, RM494.
coefficients, RM530.
assigning to, RM12.
reading from a file, RM13.
setting, RM531.
combining with COMPUTE, RM53.
converting to FRMLs, RM196.
dependent variable of, RM530.
estimating systems of, RM457.
estimating with LINREG, RM272.
fitting values, RM530.
forecasting, UG301.
grouping into a MODEL, RM224, UG410.
identity, RM531, UG410.
lag polynomial, RM530.
listing variables in, RM306.
modifying, RM306.
non-linear, UG148.
regressors, RM532.
residuals
setting, RM12, RM531.
size of, RM532.
solving systems of, RM558.
substituting variables
within, RM499.
system of, RM108.
table, RM532.
variable labels, RM531.
variance, RM12, RM531, RM532.
EQV instruction, RM146.
Error correction models, RM545, UG374.
Error location, UG580.
Error messages, RM577, UG66.
suppressing, RM140.
ERRORS instruction, RM147, UG346, UG364.
ESMOOTH instruction, RM153, UG317.
example, UG318.
ESTIMATE instruction, RM158, UG345.
CMOM option, UG382.
DUMMY option, UG382.
with prior, UG382.
Estimation. See also Regression.
RATS instructions for, RM589.
Estimators
ARCH, RM206.
Arellano-Bond, UG555.
ARMA (alternative to BOX/ENK), RM250.
Bayesian, RM44.
censored samples, UG533.
Cochrane-Orcutt, UG189.
convergence criterion, RM318.
discrete dependent variables, RM92.
distributed lags, UG201, UG207.
fixed effects, RM347, UG549, UG552.
GARCH, RM206.
HAC, RM295.
Hannan efficient, UG572.
Hec Kit, UG537.
heteroscedasticity consistent, RM295.
Hildreth-Lu, UG189.
iterated least squares, UG537.
Kalman filter, RM259.
least absolute deviations (LAD), RM396, UG214.
limit dependent variable, RM268.
linear regression, RM272, UG176.
with linear restrictions, RM315, RM307, RM385, UG223.
logit, RM92, UG525.
maximum likelihood, RM290, UG286.
minimum absolute deviations (MAD), RM396.
mixed, UG208.
multinomial logit, UG528.
multivariate non-linear least squares, RM320.
network, RM328, UG469.
non-linear, UG45.
non-linear least squares, RM313, UG276, UG278.
non-linear systems, UG282.
non-parametric, RM336.
ordinary least squares (OLS), UG25, UG43, UG183.
panel data, RM353, UG548.
parameters for non-linear, RM334.
probit, RM92, UG525.
quantile regression, RM396.
random effects, UG549, UG552.
recursive least squares, UG392.
radius, RM44, UG208.
r robust, UG214, UG215.
seemingly unrelated regressions (SUR), RM457, UG197, UG200.
simultaneous equation models, UG405.
stepwise regression, RM449.
systems of equations, RM457.
three-stage least squares, RM457, UG197, UG405.
three-stage non-linear least squares, RM320, UG282, UG405.
tobit, UG533.
truncated samples, UG533.
two-stage least squares, RM257, RM313.
Estrella, A., RM97.
EWISE instruction, RM162, UG145, UG156.
Exact line search method, RM318.
Excel files, UG39, UG73, UG85.
EXCLUDE instruction, RM164, UG31, UG220.
Exclusion restrictions, UG31, UG220.
EXECUTE instruction, RM166, UG583.
@ shorthand for, UG59.
Execution order. See Program control.
Exogenous variables in a VAR, RM108.
in forecasts, RM188, UG397.
EXP instruction, RM169.
Exponential smoothing, RM153, UG317.
forecasting, UG314.
technical information, RM156.
Exponential Smoothing wizard, RM153.
Exponentiation, RM532.
of a complex number, RM523.
Exponents, RM569.
complex, RM41.
Export menu command, RM364.
Exporting data, RM54, RM357, UG71.
Expressions, UG52.
complex numbers, UG558.
displaying results, RM113, RM505.
evaluating with COMPUTE, RM50.
logical, UG53.
looping within, RM529.
scalar, UG136.
syntax, RM569.
with series, RM420.
EXPSSMO1.PRG example, UG318.
EXPSSMO2.PRG example, UG318.
Extreme Value distribution. See GEV distribution.
Extreme values across multiple series, RM470.
of a complex series, RM79.
of a series, RM170.
EXTREMUM instruction, RM170.

F
F distribution
inverse tail probability, RM537.
tail probability, RM534.
F-tests, RM366, UG27, UG31, UG142, UG178.
CDF instruction, RM36.
for zero coefficients, RM164.
Factor analysis, RM75, UG357.
FACTOR option, UG362.
Factorial function, RM532.
Fair, R. C., RM6, UG377, UG404.
Fair’s procedure for AR1, RM6, UG404.
FAME format, UG74.
Fan chart, UG134.
FE estimator. See Fixed effects.
Index

FFT instruction, RM171, UG561, UG566.
FGLS estimator. See Generalized least squares.
Filename
of I/O Unit, RM565.
Files
appending data to, RM339.
binary, UG103.
closing, RM42.
closing RATS format, RM370.
dBase, UG73.
FAME format, UG74.
formats, UG11, UG39.
FORTRAN formats, RM56.
free-format text, RM56, UG92.
graphics, UG110.
list of available, RM422.
opening, RM339, UG10.
operations on, RM592.
organization of data on, UG11.
outputting scalars and arrays to, RM113.
PORTABLE format, UG102.
PRN format, UG85.
reading data from, RM80, RM374.
reading strings from, UG118.
rewinding, RM391.
spreadsheet, UG73, UG85.
temporary, RM42.
text (ASCII), UG74, UG95.
translating data formats, UG71.
PORTABLE, UG102.
PRN, UG85.
RATS, UG81.
WKS, UG85.
writing series to, RM54, RM356.
FILTER instruction, RM173.
Filtering, RM173.
frequency domain, UG569.
Henderson, RM173.
Hodrick-Prescott, RM173, UG446.
Spencer, RM173.
FIND instruction, RM179, UG293.
Fitted values, RM358, UG34, UG43.
FIX function, UG137.
Fixed effects estimators, RM347, UG549, UG552.
with PREGRESS, RM353.
Flannery, B. P., RM60.
Fluctuations test
distribution, RM546.
FOLD Instruction, RM187.
Fonts
for graphs, RM225, UG119.
FORCEDFACTOR procedure, UG361.
FORECAST instruction, RM188, UG34, UG300, UG320, UG482.
saving results, UG303.
Forecasts, UG47, UG298.
ARIMA models, RM17, UG314, UG319.
example, UG320.
Box-Jenkins models, UG314, UG319.
building models, RM224.
conditional, RM188, UG397, UG400.
dynamic, RM188, UG34, UG299.
error statistics, RM443.
exogenous variables in, RM188.
exponential smoothing, RM153,
UG314, UG317.
graphing, UG34, UG132, UG304.
identities, UG302.
instructions for, RM589.
of MODELS, UG409.
out-of-sample, UG412.
performance statistics, RM476, UG307, UG308.
rolling regressions, UG305.
with shocks, RM188.
spectral methods, UG314, UG334, UG574.
standard errors of, RM147, UG307.
static, RM358, RM443, UG34, UG299.
time series methods, UG299, UG314.
troubleshooting problems, UG310.
univariate, RM486.
with VARs, UG377, UG385.
FORM=CHISQUARED, UG211, UG213.
FORMAT option, RM80, UG11, UG39.
DBF, UG90.
DIF, UG85.
FORTRAN formats, UG95.
FREE, RM56, UG92.
PORTABLE, UG102.
PRN, UG85.
RATS, UG81.
WKS, UG85.
Formatting output
%BESTREP function, RM521.
%MINIMAL function, RM543.
Formulas, RM190, RM196, RM198, UG47, UG143, UG148, UG272, UG409.
saving results, UG137.
with PREGRESS, RM353.
in non-linear estimation, UG289.
Functions
calendar/date, RM32.
complex/spectral, UG558.
list of, RM511.
random draws, UG483.
using, RM511, UG36, UG53, UG138.
Functions wizard, UG138.
G
Gain
cross spectral, RM483.
Gamma distribution
random draws, RM552, UG483.
Gamma function, RM534.
digamma, RM529.
log of, RM539.
trigamma, RM564.
GARCH instruction, RM206.
GARCH models, RM206, RM290, UG417.
Bayesian analysis, UG502, UG512.
bootstrapping, UG518.
example, UG427.
multivariate, UG428, UG434.
GARCHBOOT.PRG example, UG518.
GARCHGIBBS.PRG example, UG512.
Index

GARCHIMPORT.PRG example, UG502.
GARCHMV.PRG example, UG434.
GARCHUV.PRG example, UG427.
Gardner, E. S., RM156.
Gauss-Newton algorithm, UG260.
used by NLLS, RM313.
Gauss-Seidel algorithm, UG410.
used by INITIAL, RM250.

Graph wizard, RM214, UG23.

GRAPH instruction, RM214, UG20.
Granger-Sims causality tests, UG232.
Granger, C. W. J., UG232, UG252.
Goodness of fit, UG26.
Goldfeld-Quandt test, UG228.
Godfrey-Breusch test, UG230.
Godfrey, L. G., UG230.
Glosten, L., UG420.
Global Insight database, UG74.
GIV.PRG example, UG281.
GIBBS.PRG example, UG507.
Gibbs sampling, UG263.
Gibbs sampling, UG507.
Gibbs, E. S., RM156.
GARCHUV.PRG example, UG427.
GARCHMV.PRG example, UG434.
GARCHIMPORT.PRG example, UG502.
GARCHUV.PRG example, UG427.
Gardner, E. S., RM156.
Gauss-Newton algorithm, UG260.
used by NLLS, RM313.
Gauss-Seidel algorithm, UG410.
used by INITIAL, RM250.

Graph wizard, RM214, UG23.
Graphs, RM214, UG20.
adding text to, RM228.
appearance options, RM225.
autocorrelations, UG129.
contour plots, RM212.
copying and pasting, UG113.
displaying, RM140, UG109.
exporting, UG113.
fan charts, UG134.
fonts on, UG119.
high-low-close, UG130.
highlighting entries, UG131.
impulse response, UG365.
instructions for, UG108.
keys, UG118.
labeling, RM225, RM228, UG117.
multiple per page, RM434, UG123.
opening, UG110.
orientation of, RM225, UG114.
overlay/two-scale, RM220, UG121.
preparing for publication, UG113.
printing, RM140, UG112.
saving, RM140, RM339, UG110.
scatter (X-Y) plots, RM405, UG21.
shading, UG131.
sizes of, RM225.
special, RM434.
Griddy Gibbs, UG515.
Griffiths, W. E., UG209.
Ghanem, H. A., UG189.
GARCHIMPORT.PRG example, UG247.

H

HAC covariance matrix, RM295.
UG185, UG190.
Hall, B. H., UG259, UG261.
Hall, R. E., UG259, UG261.
HALT instruction, RM232.
Hamilton, J., UG250, UG252, UG267.
UG341, UG353, UG374, UG416.
UG449, UG455.
Hamilton switching model, UG449.
HAMILTON.PRG example, UG455.
Hanhan, E. J., UG572.
Hannan efficient, UG572.
Hannan-Quinn, UG203.
HANNAH.PRG example, UG572.
Hansen, B., UG234, UG238, UG245.
UG246.
Hansen, L. P., RM324, RM460.
UG183, UG242, UG281.
Hansen specification test, UG245.
Hansens stability test, UG234, UG238.
HANSEN.PRG example, UG246.
Harvey-Collier test, UG196.
UG259, UG261, UG552.

Hausman specification test, UG243.
UG247.
HAUSMAN.PRG example, UG247.
Haver Analytics data, UG73.
Haver Analytics database, UG99.
Hazard models, UG540.
Heckit models, UG533, UG537.
Heckman, J., UG211, UG357.
Henderson filter, RM173.
Hessian matrix, UG258, UG261.
UG266.
HETERO.PRG example, UG188.
Heteroscedasticity, UG186.
consistent covariance matrix, RM295.
testing for, UG226, UG227, UG229.
HETTEST.PRG example, UG229.
Hildreth-Lu, RM6, UG189, UG404.
Hill, R. C., UG209.
Hill-climbing methods, UG259.
Histograms, RM105.
Historical decompositions, RM233.
UG372.
Historical simulations, UG413.
HISTORY instruction, RM233.
UG346, UG372.
HISTORY.PRG example, UG373.
Hodrick, R., UG446.
Hodrick-Prescott filter, RM173.
UG446.
Hoerl, A. E., UG208.
Holidays, RM31.
on data file, UG80.
Easter, RM529.
finding date of, RM524, RM533.
Holt-Winters model, RM154.
Hosking, J. R. M., UG576.
HPFILTER.PRG example, UG446.
Hsiao, C., UG548.
Hyperbolic trig functions

H

HAC covariance matrix, RM295.
UG185, UG190.
Hall, B. H., UG259, UG261.
Hall, R. E., UG259, UG261.
HALT instruction, RM232.
Hamilton, J., UG250, UG252, UG267.
UG341, UG353, UG374, UG416.
UG449, UG455.
Hamilton switching model, UG449.
HAMILTON.PRG example, UG455.
Hannan, E. J., UG572.
Hannan efficient, UG572.
Hannan-Quinn, UG203.
HANNAH.PRG example, UG572.
Hansen, B., UG234, UG238, UG245.
UG246.
Hansen, L. P., RM324, RM460.
UG183, UG242, UG281.
Hansen specification test, UG245.
Hansens stability test, UG234, UG238.
HANSEN.PRG example, UG246.
Harvey-Collier test, UG196.
UG259, UG261, UG552.

Hausman specification test, UG243.
UG247.
HAUSMAN.PRG example, UG247.
Haver Analytics data, UG73.
Haver Analytics database, UG99.
Hazard models, UG540.
Heckit models, UG533, UG537.
Heckman, J., UG211, UG357.
Henderson filter, RM173.
Hessian matrix, UG258, UG261.
UG266.
HETERO.PRG example, UG188.
Heteroscedasticity, UG186.
consistent covariance matrix, RM295.
testing for, UG226, UG227, UG229.
HETTEST.PRG example, UG229.
Hildreth-Lu, RM6, UG189, UG404.
Hill, R. C., UG209.
Hill-climbing methods, UG259.
Histograms, RM105.
Historical decompositions, RM233.
UG372.
Historical simulations, UG413.
HISTORY instruction, RM233.
UG346, UG372.
HISTORY.PRG example, UG373.
Hodrick, R., UG446.
Hodrick-Prescott filter, RM173.
UG446.
Hoerl, A. E., UG208.
Holidays, RM31.
on data file, UG80.
Easter, RM529.
finding date of, RM524, RM533.
Holt-Winters model, RM154.
Hosking, J. R. M., UG576.
HPFILTER.PRG example, UG446.
Hsiao, C., UG548.
Hyperbolic trig functions
cosine, RM526.
sine, RM558.
tangent, RM563.
Hypothesis tests, RM307, UG31.
applicability, UG222.
with BOXJENK, RM22.
coefficient values, RM473.
with consistent covariance matrix, RM297.
exclusion restrictions, RM164.
P-test, UG27.
formulas for, UG223.
instructions for, RM590, UG220.
linear restrictions, RM385.
for logit and probit, RM97.
with NLLS, UG277.
with SUR, UG199.
variables defined, UG222.
with VARs, RM372, UG347.
wizard for, UG33.
I/O units, RM339.
closing, RM42.
filename functions, RM565, RM566.
reserved, RM339.
standard, RM37.
switching, RM37.
tips on, UG104.
user-defined, RM339.

Identity, RM224.
adding to a model, UG410.
Identity matrix, RM535.
IF instruction, RM237.
%IF function as alternative to, RM237.
IFT instruction, RM171, UG561.
Imaginary part, RM525, RM535.
Importance sampling, UG500.
Importing data, UG71.
IMPULSE instruction, RM240, UG346, UG364.
alternative to, RM148.
Impulse responses, RM240, UG364, UG368.
cumulative, UG365.
fractiles, UG497.
graphing, UG365.
standard errors of, UG497.
IMPULSES.PRG example, UG368.
INCLUDE instruction, RM246, UG82.
alternative to, RM446.
Individual effects, RM366, UG546.
means, RM346.
Inequality constraints
linear/quadratic programming, RM283.
non-linear estimation, RM334, UG264.
INFOBOX instruction, RM248, UG598.
example using, UG478.
Information criteria, UG203. See also Akaike criterion.
INITIAL instruction, RM250.
Innovations process, UG351.
Input
changing source of, RM37.
echoing of, RM140.
SOURCE instruction, RM429.
INPUT I/O unit, RM339.
INPUT instruction, RM251, UG145.
with complex series, RM401.
Inputting
arrays, RM251.
variables, RM251.
INQUIRE instruction, RM254, UG591.
Instructions
changing default options, RM103.
continuing on another line, UG55.
echoing, RM140.
execting, UG2.
infinite input files of, RM339.
listed by general function, RM589.
locations in compiled code, UG580.
multiple per line, UG63.
secondary input files, RM429.
splitting up long, UG63.
syntax of, UG60.

INSTRUMENT.PRG example, UG194.
Instrumental variables, RM257, RM464, UG192.
generalized, UG281.
with LINREG, RM272.
non-linear least squares, UG277, RM313.
simultaneous equation models, UG404.
INSTRUMENTS instruction, RM257, UG192, UG404.
Integer
absolute value, RM534.
from real, RM533.
to real, RM535.
to string, RM559.
values in expressions, UG137.
INTEGER data type, UG137, UG143, UG146.
Integral
Simpson's Rule, RM538.
trapezoidal rule, RM538.
Interactive mode, UG2.
Interactive procedures, UG598.
dialog boxes, RM248, RM304.
UG598.
input from user with QUERY, RM368.
menus, RM303, UG497.
menus with USERMENU, RM497.
selecting from a list, RM497.
using PROCEDURE, RM362.
viewing/editing arrays, RM300.
INTERPOL procedure, RM32, UG77.
Interpolating data, RM32, UG77.
Interval estimation, UG535.
INTERVEN.PRG example, UG337.
Intervention models, RM17, UG336.
Intraday data, RM29.
INV function, UG159.
Inverse
autocorrelations, RM60.
generalized, RM534, UG159.
matrix, RM536, UG159.
Iterated least squares, UG537.
Iterations
non-linear estimation, UG256, UG262.

J
Jacquier, E., UG489.
Jarque, C. M., RM439.
Jarque-Bera Normality test, RM439.
Jenkins, G. M., RM250, UG319.
Johansen cointegration test, UG253.
JOHMLE procedure, UG253, UG254.
JOHNP121.PRG example, RM394.
Joyce, R., UG576.
JROBUST option, UG280.
Judge, G.G., UG209.
Julian date, RM538.

K
Kalman filter, RM259.
equation variances, RM265.
estimating parameters, RM179.
initial values, RM158, UG389.
likelihood function, RM264.
recursive residuals, UG390.
sequential estimation, UG389.
setting up, RM263, RM466.
state-space models, RM116, UG439.
time-varying coefficients, RM484.
UG391, UG394.
updating formula, UG388.
VAR models, UG388.
KALMAN instruction, RM259.
Kalman smoothing, RM116, UG439, UG446.
Kendall, M. G., RM441.
Kennard, R. W., UG208.
Kernel methods
computing density, RM105.
density estimation, UG460.
non-parametric regression, RM336, UG460.
KEYBOARD I/O unit, RM339.
KFSET instruction, RM263, UG391.
Kim, S., UG489.
KLEIN.PRG example, UG192.
Koopman, S.J., UG441.
Koopmans, L. H., UG185, UG574.
Kroneker product, RM538, RM539, UG161, UG164.
Kroner, K. F., UG428.
Kurtosis, RM439, UG13.

L
LABEL data type, UG143, UG146.
Labels, UG154, UG596.
of complex series, RM39.
creating series from, RM557.
on data files, UG86.
expressions using, RM52, RM571.
for graphs, UG117.
series, RM266.
of variables, RM539.
LABELS instruction, RM266.
vs. EQV instruction, RM146.
LAD estimator, UG214.
Lag length
methods for choosing, UG203.
VAR
choice of, UG379.
testing, UG348.
Lagrange multiplier
%LAGRANGE variable, RM326.
constrained optimization, RM335, UG264.
tests, UG224, UG226, UG230, UG278.  
OPG (outer product of gradient), RM299.

Lags  
(,) notation in expressions, UG17.  
in regressor lists, UG42.  
non-consecutive, with BOXJENK, RM18, 
with panel data, UG545.  
LAGS instruction, RM267, UG342.  
LAGS option for serial correlation, UG189.  
Lancaster, T., UG540.  
Landscape mode, RM225, UG114.  
LDV.  
See Limited dependent variables.  
LDV instruction, RM268, UG533, UG534, UG539.  
Leamer, E., UG554.  
Least Absolute Deviations.  See LAD estimator.  
Least squares iterated, UG537.  
with LINREG, RM272.  
non-linear, RM313.  
recursive, RM392.  
three-stage, RM457, UG405.  
two-stage, UG406.  
Lee, T-C., UG209.  
Levenberg-Marquardt algorithm, RM319.  
LGT instruction, UG525.  
Likelihood function hazard models, UG540.  
Kalman filter, UG393.  
limited dependent variables, UG534.  
maximum likelihood estimation, UG286.  
multivariate, UG288.  
partial, UG540.  
probit and logit, UG525, UG530.  
structural VAR, UG353.  
Likelihood ratio test, RM372, UG220, UG347.  
Lilien, D. M., UG421.  
Limited dependent variables, RM268, UG533.  
Limited information maximum likelihood, UG405.  
Limited/Discrete Dependent Variables wizard, RM92, RM268.  
Line numbers of errors, UG580.  
Linear filters, RM173.  
probability model, UG531, UG532.  
programming, RM283, UG464.  
restrictions, RM135, RM385.  
LINREG instruction, RM272, UG25.  
CMOMENT option, RM44, UG208.  
CREATE option, UG212.  
FRML option, UG409.  
instrumental variables, RM257.  
SPREAD option with panel data, RM347.  
UNRAVEL option, UG202.  
with VAR's, UG345.  
LIST instruction, RM278, UG363.  
example, RM149, RM244.  
Litterman, R., RM264, UG343, UG367, UG392, UG393.  
Ljung, G. M., RM59, UG231.  
Ljung-Box Q, RM58, RM66, RM276, UG230.  
Lo, A., RM201, UG416.  
LOCAL instruction, RM280, UG588.  
Local optima, UG257.  
Local variables, UG583, UG588.  
LOCALDLM procedure, UG446.  
Log determinant, RM541.  
Log function, RM539.  
complex, RM524.  
elementwise, RM540.  
LOG instruction, RM281.  
Log likelihood %LOGL reserved variable, RM276.  
Logical expressions, RM569.  
with %IF function, RM535.  
with IF instruction, RM237.  
Logical operators, RM569, UG53.  
creating dummies using, UG49.  
with complex values, UG558.  
Logistic distribution CDF, RM539.  
Logistic function, RM541.  
Logit models, RM92, UG525, UG531, UG532.  
fitted values for, RM358.  
samples with repetition, UG530.  
Long memory models estimation, UG576.  
Long-run restriction, UG358.  
LOOP instruction, RM282, UG56, UG581.  
Looping instructions, RM593.  
DO, RM124.  
DOFOR, RM126.  
LOOP, RM282.  
NEXT, RM312.  
UNTIL, RM489.  
WHILE, RM501.  
Loops, UG580.  
branching out of, RM24.  
breaking out of, RM25.  
conditional, RM489, RM501.  
DO, RM124.  
ending, RM137.  
exiting, RM24, RM25.  
indeterminate, RM282.  
introduction to, UG56.  
jumping to next pass, RM312.  
line number of error in, UG580.  
nested, RM124.  
quitting RATS from within, RM232.  
using %DO function, RM529.  
Lower case converting string to, RM560.  
LOWESS.PRG example, RM338.  
LQPROG instruction, RM283.  
Luenberger, D. A., UG466.  
LWINDOW option, UG185.  

M  
MacKinlay, A. C., RM201, UG416.  
MacKinnon, J.  
MacKinnon-White-Davidson test, UG241.  
Maital, S., UG190.  
MAKE instruction, RM287, UG157.  
Marginal significance levels, RM36, UG179, UG220, UG222.  
Markov switching models, UG449.  
ARCH, UG458.  
MARKOV.SRC procedure, UG449.  
Marquardt algorithm, RM319.  
Matrix  
[ ... | ] notation, UG157.  
absolute value of, RM520.  
average value of, RM521.  
banded, RM185.  
block diagonal, RM522.  
calculations, RM50.  
Choleski factorization, RM528, RM549, UG162.  
computations, UG36.  
congenation, RM522.  
copying data to series from, RM289.  
correlation, RM44, RM525.  
creating, UG144.  
from data series, RM287.  
cross-moment, RM44.  
declaring, RM99.  
defined by RATS instructions, UG156.  
determinant, RM528, UG160.  
diagonal, RM529.  
diagonalizing, RM549.  
dimensioning, RM112, RM287.  
displaying, RM113, RM505, UG145.  
DOFOR lists, RM126.  
dot product, RM529.  
eigen decomposition, RM133.  
elementwise operations, RM162.  
exponentiation, RM532.  
expressions, UG156.  
extracting columns, RM567.  
diagonal from, RM567.  
first element, RM557.  
rows, RM567.  
from series, RM568.  
submatrix, RM567, UG161.  
subvector, RM567.  
extreme values, UG160.  
factiles of, RM533, RM567.  
identity, RM535, UG159.  
inner product, RM536.  
instructions, UG156.  

614 RATS Reference Manual
inverse, RM536, UG159.
generalized, UG159.
Kronecker product, RM538, RM539, UG161.
using in lists, UG149.
maximum value of, RM542.
minimum value of, RM543.
number of columns, RM525.
number of rows, RM556.
one-dimensional, UG148.
optimizing computations, UG164.
outer product, RM546.
partitioning, RM522.
printing, RM505.
QR decomposition, RM551.
quadratic form, RM545.
QZ decomposition, RM371.
ranks of elements, RM552.
RATS instructions for, RM592.
reading data into, RM251, RM374.
reading/writing complex series as, RM401.
rectangular, UG143.
RECTANGULAR data type, UG148.
reshaping, RM566.
sparse, RM542.
submatrix setting, RM549.
subvector setting, RM550.
summing, RM560.
SV decomposition of, RM560.
sweep function, RM278, UG145.
sorting, RM536, RM558, UG162.
symmetric, UG143.
SYMMETRIC data type, UG148.
trace, RM564.
transpose, RM564, UG159.
unit vector, RM566.
vector, UG143.
VECTOR data type, UG148.
vectorizing, RM566, UG159.
viewing/editing with MEDIT, RM300, UG145, UG598, UG601, UG606.
zero, RM568.
Maximization using FIND, RM179, UG293.
MAXIMIZE instruction, RM290, UG286.
for censored and truncated samples, UG533.
formulas for, RM196.
for logit and probit, UG530.
parameters for, RM334.
Maximum likelihood, RM290.
AR1 models, RM6.
ARMA models, RM18.
censored and truncated samples, UG533, UG534.
estimation, UG286.
full information, UG405.
limited information, UG405.
tobit models, UG533.
Maximum value of array, RM542.
of complex series, RM79.
of integers, RM535.
moving window, RM309.
of reals, RM542.
of a series, RM170, RM439, RM468.
McCurdy, T. H., UG423.
MCOV instruction, RM295, UG157, UG211, UG215, UG279.
Mean individual (panel data), UG46.
moving window, RM309.
of a series, RM439, RM468.
MEANGROUP.SRC procedure, UG554.
Median. See also Fractiles.
moving window, RM309.
of a series, RM439.
MEMORY available, RM422.
clearing, RM137.
freeing, RM377.
MENU instruction, RM303, UG410.
Menus (user-defined), RM303, UG599.
pull-down, RM491.
USERMENU, UG598.
MESSAGEBOX instruction, RM304, UG598.
Metropolis-Hastings method, UG510.
Mills ratio, RM359, RM543, UG533, UG537.
derivative of inverse, RM529.
Minimization linear/quadratic programming, RM283, UG464.
using FIND, RM179, UG293.
Minimum value of array, RM543.
of complex series, RM79.
of integers, RM535.
moving window, RM309.
of reals, RM543.
of a series, RM170, RM439, RM468.
Missing values, RM31, UG17, UG50, UG51.
on data files, UG78.
in expressions, UG55.
in forecasts, UG310.
removing from a series, RM402.
testing for, RM566.
Mixed estimation, UG208.
MIXED procedure, UG209.
MIXVAR procedure, UG386.
MODEL data type, UG143, UG147, UG409.
Model selection stepwise regressions, RM449.
Models building with GROUP, RM224.
coefficients of, RM544.
combining with +, RM53.
companion matrix, RM543.
defining with SYSTEM, RM466.
dependent variables, RM544.
dynamic linear, RM116.
equations extracting, RM544.
locating, RM544.
replacing, RM544.
forecasting, RM188, UG409.
instructions for, RM590.
simulations, RM423.
size of, RM545.
state-space, RM116.
MODIFY instruction, RM306, UG410.
Modular arithmetic %CLOCK, RM524.
%MOD, RM543.
Monte Carlo antithetic acceleration, UG496.
bookkeeping for, UG478.
importance sampling, UG500.
impulse response standard errors, UG497.
tests, RM423, UG486.
MONTARCH.PRG example, UG486.
MONTEVAR.PRG example, UG497.
Montly data, RM28.
Morgan, I. G., UG423.
Moving average, UG18.
adding lags to equation, RM494.
alternative to, RM309.
representation, RM240, UG351.
correction, RM295.
Moving Window Statistics wizard, RM309.
MRESTRICT instruction, RM307, UG206, UG220.
MSVARSETUP procedure, UG454.
Multipliers, UG413.
MVSTATS instruction, RM309.
N Names for numbered series, RM146, RM266.
legal form, UG4, UG144.
Negative binomial distribution density, RM541, RM546.
Nelson, C. R., UG344.
Neural networks, UG469.
convergence of, UG473.
fitted values/predictions, RM333.
limit values of, UG474.
tips on fitting, UG472.
training/fitting, RM328.
Index

NEURAL.PRG example, UG476.
New RATSADATA menu
command, RM246.
Newbold, P., UG319.
Newey, W., UG185.
Newey-West estimator, RM297, UG185.
Newton-Raphson, UG258, UG260.
NEXT instruction, RM312.
NLLS instruction, RM313, UG276.
FRMLs for, RM196.
instrumental variables, RM257.
technical details, UG268.
NLLS.PRG example, UG278.
NLPAR instruction, RM318.
NLSYSTEM instruction, RM320, UG282.
NLPAR instruction, RM318.
NLLS.PRG example, UG278.
NLLS instruction, RM313, UG276.
NEXT instruction, RM312.
Newton-Raphson, UG258, UG260.
Newey-West estimator, RM297,
Newey, W., UG185.

NONLIN instruction, RM336, RM313,
Non-parametric regressions, RM336.
Non-linear systems
Non-linear least squares, RM313,
Non-linear estimation, RM290, UG45.
Non-differentiable functions
optimizing, UG293.
Non-linear estimation, RM290, UG45.
constrained, UG264.
convergence, RM318, UG256.
forecasting, UG302.
formulas for, RM196.
initial values, UG257.
introduction to, UG256.
maximum likelihood, UG286.
method of moments, UG279.
parameters for, RM334, UG270.
restrictions, UG264.
user-defined functions, UG289.
Non-linear least squares, RM313, UG276.
example of, UG278.
formulas for, UG272.
multivariate, RM320.
parameters for, UG270.
Non-linear systems
estimation, UG282.
solving, UG409.
Non-parametric regressions, RM336.
example, UG461.
NONLIN instruction, RM290, RM313, RM334, UG145, UG270.
Normal distribution, RM359. See also
Distributions.
CDF, RM523.
CDF instruction, RM36.
density, RM539, RM528.
inverse CDF, RM537.
log CDF, RM540.
mills ratio, RM543.
multivariate density, RM540.
concentrated, RM540.
quadratic form PDF, RM550.
random draws, RM551, RM552.
U160, UG478, UG483.
multivariate, RM552, RM553, UG484, UG491.
statistics, RM358.
test for, RM439.
two-tailed tail probability, RM568.
Normalizing series, UG50.
NPREG instruction, RM336, UG460.
NPREG.PRG example, UG461.
Null space, RM546, RM548.
Numbered series
labels for, RM39.
naming, RM266.
Numbers, UG62.
“best” representation for
displaying, RM521.
Nyblom distribution, RM546.
OK/Cancel dialog box, RM304.
OLS. See LINREG instruction.
OLSMENU.PRG example, UG599.
OPEN instruction, RM57, RM82, UG276.
RM336, UG460.
Open RATSADATA menu
command, RM246, RM364, RM378, RM446, UG83.
Operators
arithmetic, RM569.
label and string, RM571.
logical, RM569.
precedence, RM570, UG52.
OPG test, RM299.
Optimal control
linear-quadratic, RM116, UG437, UG442.
OPTIMALWEIGHTS option, UG279.
Optimization
linear/quadratic
programming, RM285.
using FIND, RM179.
OPTION instruction, RM341, UG589.
Options, UG10, UG60.
changing defaults, RM103.
defining in procedures, RM341.
field, UG60.
procedure, RM166, UG583, UG589.
types of, UG61.
using expressions with, UG589.
ORDER instruction, RM344.
Orderings
choosing for VAR, UG367.
Ordinary Least Squares. See LINREG
instruction.
ORGANIZATION option, RM80, UG11, UG40
with FORTRAN formats, UG96.
with free-format, UG92.
with spreadsheet files, UG85.
Orthogonalization, RM240, UG351.
Choleski factorization, UG352.
Outer product of the gradient
test. See OPG test.
Outliers, UG182.
Output
changing destination of, RM37.
data series, RM364.
displaying variables, RM113.
formatted reports, RM379.
formatting, RM113, RM505, UG141.
%bestrep, RM521.
%minimalrep, RM543.
with PRINT, UG16.
opening file for, RM339.
regression, UG176.
in table form, UG169.
VCV option, UG179.
width, RM140.
OUTPUT unit, RM339.
Overflow, UG256.
check for numeric, RM47.
Overidentifying restrictions, UG242, UG355, UG357.
Overlay graphs, RM220, UG121.
OVERLAY instruction, UG209.

P
Padding complex series, UG563.
Page break
inserting in output, RM49.
Panel data, UG542.
analysis of variance tests, RM366.
AR1 regression, UG550.
CALENDAR for, RM29, UG542, UG543.
date expressions, RM29.
dummy variables, UG545.
fixed/random effects estimation
using PREGRESS, RM353.
forming series into, RM348.
individual number, RM536.
instructions for, RM591.
lags in, UG545.
organization of, UG542.
random coefficients model, UG545.
reading from disk files, UG544.
regressions, RM346, RM533, UG548.
seasonal dummies, RM415.
series entry reference, UG557.
subsamples, UG181.
Swamy’s random coefficients
model, UG554.
time period within, RM548.
transformations, RM346, UG546.
unbalanced, RM348.
variance components, RM354.
Panel Data Regressions
wizard, RM353.
 PANEL instruction, RM346, UG546, UG553.
NPREG.PRG example, UG553.
Parameters
instruction, UG7, UG61.
non-linear estimation, RM334, UG45.
procedure, RM167, RM485, UG583.
%DEFINED function, RM343.
by address vs by value, UG587.
formal vs actual, UG586.
using INQUIRE with, RM255.
Pareto distribution. See Generalized Pareto distribution.
PARMSETS, RM334, UG143, UG147, UG270.
pulling into vector, RM547.
setting from a vector, RM547.
Parsimony
Partial autocorrelations, RM58.
Partial likelihood
hazard models, UG541.
Partial sums
of complex series, RM26.
of series, RM2.
Partitioning matrices, RM522.
PDL.PRG example, UG207.
Periodograms, UG566.
smoothing, RM502.
Permutation
random, RM553.
Perron, P., UG249.
PFORM instruction, RM348, UG542, UG546.
Phase lead, RM351, UG568.
Phillips, P. C. B., UG249.
Phills-Perron unit root test, UG249.
PICT format, RM124.
Picture codes, RM113, UG141.
on PRINT, RM356, UG16.
Piecewise linear
Poisson
Polynomial functions, RM548, UG542, UG546.
Polynomial, RM553.
Porter-Hudak, S., UG576.
PostScript format, RM140, UG110.
PPUNIT procedure, UG249.
Prag-Winsten, RM6.
PPUNIT procedure, UG249.
PostScript format, RM140, UG110.
Position codes, RM114.
PostScript format, RM140, UG110.
PPUNIT procedure, UG249.
Precedence
of operators, RM569, UG52.
Precision of numbers, UG256.
test for loss of, RM546.
Predetermined variables, RM257.
Prediction. See Forecasts.
Preferences menu command, UG585.
PREGRESS instruction, RM353, UG552, UG553.
Prescott, E., UG446.
Present value, RM520.
Press, W. H., RM60, UG261, UG262.
PRINT instruction, RM356, UG16.
using COPY instead, RM54.
PRINTER I/O unit, RM339.
Printing
complex series, RM64, UG564.
data from a RATS file, RM364.
data series, RM356.
graphs, UG112.
variables, RM113.
Prior
VAR, RM430, UG378, UG380, UG382, UG383.
mean, UG379.
tightness, UG380.
PRJ instruction, RM358, UG34.
censored/truncated data, UG537.
forecasting with, UG34.
PRN files, RM80, UG39, UG85.
Probit models, RM92, UG525, UG531, UG532.
Probit, RM92, UG525.
Probit models
fitted values for, RM358.
samples with repetitions, UG530.
sequential, UG530.
PROBIT.PRG example, UG531.
PROCEDURE instruction, RM362, UG583.
Procedures
RM362, UG58, UG580, UG583.
%DEFINED function, RM343, UG590.
development tools, RM98.
END statement, RM107.
executing, RM166, UG59.
instruction order, UG538.
instruction order, UG538.
instructions for, RM594.
library of, UG585.
local variables in, RM280.
nested, RM167.
options, RM166, RM341, RM528.
parameters, RM548, RM528.
reading from a separate file, RM429.
returning from, RM390.
search directory, UG585.
setting from a RATS file, RM364.
stoart in separate files, UG584.
 supplementary cards, RM138.
Programming
menus, RM303.
user input, RM368.
Progress bars, RM248, UG478.
Proportional hazard models, UG540.
PRTDATA instruction, RM364, UG82, UG102.
PSTATS instruction, RM366, UG548.
Q
Q test
cross correlation, RM66.
serial correlation, RM61.
QPROG.PRG example, UG467.
QR decomposition, RM551.
Quadratic form
CDF of, RM550.
computing, RM550.
Quadratic programming, RM283, UG464.
Quah, D., UG358.
Quartile regression, RM396.
Quantiles. See Fractiles.
Quarterly data, RM28.
Quattro spreadsheet files, UG85.
QUERY instruction, RM368, UG145, UG598, UG601.
QUIT instruction, RM370, UG82.
QZ instruction, RM371, UG157.
R
R-squared statistics, RM276, UG26, UG177.
Ramsay, J., UG240.
Random coefficients model
panel data, UG554.
Random effects estimators, RM353, UG549, UG552.
Random numbers, UG478.
 algorithms, RM417.
 beta, RM551.
 chi-squared, RM551.
 combination of integers, RM551.
 element in set, RM551.
 functions for, UG483.
 gamma, RM552.
 generating series of, RM423.
 initializing, RM416.
 integers, RM552.
 multivariate distributions, UG491.
 multivariate normal, RM552, RM553.
 normal, RM551, RM552, UG160.
 reproducing, RM416.
 t distribution, RM553.
 techniques for, UG487.
 uniform, RM565, UG160.
 uniform integers, RM15.
 on unit sphere, RM553.
 Wishart distribution, RM554.
Randomization tests, UG520.
RANDOMIZE.PRG example, UG520.
Range of entries
changing default, RM427.
Rank correlation, RM344.
RATIO instruction, RM372, UG220, UG347.
RATS
compiler, UG580.
version number, RM554.
RATS format files, UG39, UG72, UG81.
Index

abort editing/close file, RM370.
adding/replacing data, RM490.
deleting series from, RM104.
editing series on, RM130.
editing/creating, RM101.
finding series length, RM254.
instructions for, UG82.
listing contents of, RM35.
older versions of, UG84.
panel data, UG544.
printing series, RM364.
RATS instructions for, RM592.
finding series length, RM254.
editing series on, RM130.
deleting series from, RM104.
adding/replacing data, RM490.
abort editing/close file, RM370.
with panel data, UG550.
with AR1 errors, RM6.
band spectrum, UG573.
F statistic, UG178.
fitted values, RM358.
forecasting, UG298.
instructions for, RM589.
with LINREG, UG25.
non-parametric, RM336.
ordinary least squares, RM272.
output, UG25, UG176.
panel data, RM353.
quantile, RM396.
recursive least squares, RM392.
reporting results, UG169.
restricted, RM135, RM307, RM385, UG205, UG220.
ridge, UG208.
robust, RM396.
rolling, UG138.
seemingly unrelated, RM457, UG197.
standard Normal linear model, UG183.
statistics accessing, UG142.
defined by LINREG, RM276.
stepwise, RM449.
testing restrictions, RM164.
using SWEEP, RM464.
Regressions wizard, RM6, RM272, UG220.
Regression tables, RM562.
Regression format, UG42.
REGRESET procedure, UG240.
REGCRITS procedure, UG203.
REGCORRS procedure, UG231.
Recursive residuals, RM272.
using SWEEP, RM464.
recursive, UG390.
generalized, UG516.
Resampling, UG516.
Residuals
generalized, UG535.
recursive, UG390.
sum of squares, RM276, UG178.
for forecasts, RM476.
Roots
using FIND, RM179, UG293.
Rossi, P., UG489.
Rounding function, RM556.
RREG instruction, RM396.
Running regressions, RM379, RM402.
with complex series, RM564.
with supplementary cards, RM278.
REPORT instruction, RM379, UG145, UG169, UG281.
Reports
generating, RM379, UG169.
Resampling, UG516.
Reserved variables
list of, RM573.
RESET test, RM394, UG240.
RESET.SRC procedure, RM394.
Residuals
generalized, UG535.
recursive, UG390.
sum of squares, RM276, UG178.
for forecasts, RM476.
BREATHTEST procedure, RM405, UG226.
BJEST procedure, UG332.
for VARs, UG344, UG348.
Schwarz criterion, UG203, UG204, UG320.
for VArS, UG344, UG348.
Schwarz, G., UG203.
Schwert, G., UG249.
SCREEN I/O unit, RM339.
Seasonal differencing, RM109.
dummies, RM413, UG49.
forecasting models, UG316.
mask, RM43.
removing with linear regression, RM173.
Seasonal adjustment
Census X11, RM507.
exponential smoothing, RM153.
frequency domain, UG571.
SEASONAL instruction, RM413.
Serial correlation
Sequential probit, UG530.
Sequential selectivity bias, UG533.
SELECT instruction, RM418, UG598.
Sequence
of integers, RM557.
of reals, RM557.
Sequential probit, UG530.
Serial correlation
AR1 example, UG191.
consistent covariance
matrix, RM295.
Durbin-Watson statistic, UG178.
estimation with, RM6, RM199,
UG43.
estimation with high order, RM173,
UG572.
estimation with panel data, UG550.
periodogram test for, RM26.
regressions with, UG189.
tests for, RM58, UG27, UG230,
UG231.
Series
of arrays, RM231.
arrays of, RM287, RM550, UG151,
UG595.
clearing, RM40.
compacting frequency, UG76.
compressing, RM402.
creating, RM40, RM130, RM420,
UG17.
creating from array, RM289.
creating from label (%S), RM557,
UG596.
cross-sectional, UG524.
data type, UG148, UG150.
aggregate, UG143, UG148.
default length, RM4, UG8.
deleting from a RATS format
file, RM104.
displaying, RM54, UG16.
dummy variables, UG49.
editing on a RATS format
file, RM130.
editing a RATS format
file, RM130.
empirical density function, RM105.
expanding frequency, UG77.
periodogram test for, RM26.
regressions with, UG189.
tests for, RM58, UG27, UG230,
UG231.
Series Window
generating statistics, UG13.
SET instruction, RM420, UG17,
UG48.
Shephard, N., UG489.
SHILLER.PRG example, UG210.
Shocks
adding to forecasts, RM192.
responses to, RM240.
SHOCKS option
example, UG414.
SHORTANDLONG
procedure, UG359.
SHOW instruction, RM422.
Show Series Window wizard, RM246,
RM356, RM446, RM496.
Significance levels
computing, RM36, UG220.
definition, UG27.
interpreting, UG179.
Simplex method
non-linear optimization, UG262.
Sims, C. A., RM264, RM373, UG202,
UG232, UG251, UG343, UG347,
UG351, UG353, UG365, UG392,
UG393, UG571.
Sims exogeneity test, UG232.
SIMULADD.PRG example, UG414.
SIMULATE instruction, RM423,
UG478, UG482.
Simulations, RM423, UG47, UG482.
bookkeeping for, UG478.
computing a probability, UG479.
historical, UG413.
using FORECAST, RM188.
SIMULEST.PRG example, UG406.
SIMULFOR.PRG example, UG412.
SIMULMUL.PRG example, UG413.
Simultaneous equation models
accuracy statistics, UG413.
add factors, UG414.
algorithm for solving, UG411.
constructing models, RM224.
convergence, UG412.
example, UG406.
formulas, UG409.
historical simulations, UG413.
identities, RM196.
multipliers, UG413.
out-of-sample forecasts, UG412.
preliminary
transformations, UG404.
three-stage least squares, UG197.
Sine
function, RM558.
hyperbolic, RM558.
inverse, RM521.
Single-Equation Forecasts
wizard, RM368, RM486, UG300.
Singleton, K.J., UG281.
Singular value
decomposition, RM560, UG162.
Skewness, RM439, UG113.
SKIPSAVE option, RM193.
Smith, A. F. M., UG505.
Smoothing
complex series, RM502.
exponential, RM153, UG314.
Kalman, UG437.
SMPL instruction, RM427, UG64,
UG65.
getting current setting, RM254.
SMPL option, UG180.
with Kalman filter, RM259.
using SMPL instruction, RM427.
Solving
linear equations, RM558.
models, UG411.
Sorting
arrays, RM558.
data series, RM344.
vectors, RM536.
SOURCE instruction, RM429, UG58,
UG584, UG598.
Spearman’s rho, RM344.
SPECFORE procedure, UG334,
UG574.
SPECFORE.PRG example, UG335.
Specification tests, UG232, UG242,
UG281. See also Tests.
Harvey-Collier, UG196.
SPECIFY instruction, RM430, UG380.
Spectral analysis, UG562.
  complex expressions, UG558.
  density, UG566, UG568.
  forecasting, UG314, UG334.
  window, RM502, UG566.
Spectral density, RM352.
Spectral density function, RM187.
SPECTRUM procedure, UG568.
Spencer filter, RM173.
SPGRAPH instruction, RM214, RM434, UG108, UG123.
Splines, UG202.
SPREAD option, UG186.
  with panel data, RM347.
Spreadsheets, UG73.
  adding dates to files, UG89.
  reading data from, UG85.
  troubleshooting, UG89.
Square root function, RM559.
  complex, RM526.
  elementwise, RM559.
SSTATS Instruction, UG153.
Stability tests, UG234, UG238.
  See also Tests.
  CUSUM, RM393.
STABTEST procedure, UG234.
STABTEST.SRC procedure, UG234, UG238.
Standard error
  of autocorrelations, RM61.
  of coefficients, vector of, RM276.
  of estimate, UG178.
  of forecast, RM150, UG307.
  of projection, RM358.
  of series, RM468.
STARTUP option, UG541.
State space model. See DLM.
Static forecasts, UG34, UG299.
Statistics
  forecast performance, RM476.
  instructions for computing, RM591.
  marginal significance levels, RM36.
  moving window, RM309.
  regression, RM276, UG176.
  for series, RM439, RM468, UG13.
STATISTICS instruction, RM437, RM439, UG13.
STEPS instruction, RM443, UG300.
  saving results, UG303.
Stepwise regressions, RM449, UG43.
Stereo projection, RM559.
Stochastic volatility model, UG448, UG489.
Stock, J., UG252.
STORE instruction, RM446, UG82.
STRING data type, UG143, UG146.
Strings
  comparison, RM559.
  concatenating, UG118.
  expressions with
    COMPUTE, RM52.
  extracting substring, RM539, RM542, RM555.
    functions, UG155.
    from integer, RM559.
    length of, RM559.
    line breaks in, UG118.
    to lower case, RM560.
    numerical value from, RM566.
    operations with, RM571.
    reading from files, UG118.
    from real value, RM560.
    selecting from list, RM418.
    to upper case, RM560.
  variables, UG154.
Structural break tests. See Stability tests.
Structural breaks in ARIMA models, UG336.
Structural VAR, RM75, UG353, UG362, UG370.
Stuart, A., RM441.
STWISE instruction, RM449.
Sub-FRML’s, RM199.
Subprograms, UG583.
Sum
  of an array, RM560.
  of coefficients, RM453.
  partial, RM2.
  of squared residuals, UG178.
SUMMARIZE instruction, RM453, UG201, UG220.
Supplementary cards, UG20, UG61.
  ENTRIES option, UG217.
  long, UG61.
  in procedures, RM138.
  sets of, RM278.
SUR instruction, RM457, UG197.
  example of, UG200.
  technical information, UG197.
  three stage least squares, UG405.
  with VAR’s, UG454.
SUR.PRG example, UG200.
SV.PRG example, UG448.
Swamy, P. A. V. B., UG554.
SWAMY.PRG example, UG554.
SWAMY.SRC procedure, UG554.
Swamy’s random coefficients, UG554.
SWARCH.PRG example, UG458.
Sweep functions, RM560.
SWEEP instruction, RM464, UG549.
Switch options, UG61.
SYMMETRIC data type
  aggregate, UG143, UG148.
  Symmetric matrix
    packing/unpacking
    functions, RM561.
  Syntax errors, UG66.
SYSTEM instruction, RM466, UG342.
  Systems of equations estimating, RM457.
  forecasting, RM188.
  linear estimation, UG197.
T
  distribution
    CDF instruction, RM36.
    density, RM563.
    inverse CDF, RM538.
    inverse tail probability, RM538.
    multivariate density, RM541.
    multivariate random draw, UG484.
    non-central
    CDF, RM563.
    density, RM563.
    random draws, RM553, UG483.
    tail probability, RM564.
  t variable for SET, RM421, UG17.
  t-statistics, RM439, UG27.
    vector of coefficient, RM276.
TABLE instruction, RM468, UG13.
  from Series Window, UG13.
Tangent
  function, RM563.
  hyperbolic, RM563.
  inverse, RM521.
TAPER instruction, RM471, UG567.
Tapers, RM471, UG567.
Taqqu, M. S., UG576.
TAR model, UG196.
Taylor, W. E., UG552.
TEST instruction, RM473, UG31, UG220.
Tests
  analysis of variance, RM366.
  applicability, UG222.
  ARCH, UG227.
  Bayesian odds ratio (unit root), UG251.
  Breusch-Pagan, UG226.
  causality, UG232.
  Chow, UG234, UG236.
  coefficient values, RM473.
  cumulated periodogram, RM26, UG231.
  Diebold-Mariano, UG309.
  Engle-Granger cointegration, UG253.
  exogeneity, UG232.
  Goldfeld-Quandt, UG228.
  Hansen specification, UG245.
  Harvey-Collier misspecification, UG196.
  Hausman specification, UG243, UG247.
  heteroscedasticity, UG226, UG229.
  Johansen cointegration, UG253.
  lag length, UG348.
  Lagrange multiplier, UG224.
  likelihood ratio, RM372.
  linear restrictions, RM307, RM385.
  MacKinnon-White-Davidson, UG241.
  marginal significance levels, UG220.
  Monte Carlo, RM423.
normality, RM439.
overidentifying restrictions. See
Overidentifying restrictions.
RESSET, UG240.
serial correlation, RM61, UG230.
specification, UG242.
structural stability, UG234.
unit roots, UG249.
White heteroscedasticity, UG227.
Teukolsky, S. A., RM60.
Text files, UG39, UG74, UG95.
Theil, H., UG200, UG208, UG285,
UG345, UG382.
THEIL instruction, RM476, UG327,
UG383, UG413.
Theil U statistic, RM476, UG308,
UG383.
Three-stage least squares, RM457,
UG405, UG406.
non-linear, RM320.
Threshold autoregression, UG196.
Time effects, RM366, UG546.
computing means of, RM346.
Time-varying coefficients, RM484,
UG391.
estimating with KALMAN, RM259,
more general, UG394.
Time/date stamp, RM527.
TO triples, UG42.
Tobit models, UG533, UG539.
TOBIT.PRG example, UG539.
Today
entry number of, RM564.
Trace function, RM564.
TRACE option, UG257.
Trading days, RM564.
Transfer function
models, RM17.
spectral, RM482.
Transformations wizard, RM2,
RM281, RM420, UG19.
Transpose function, RM564, UG159.
TRAPEZOID taper, RM471.
Trend
creating, RM420, UG17, UG48.
and forecasts, UG315.
removing, RM109.
removing with HP filter, RM173,
UG446.
removing with linear regression, RM173.
Trend/Seasonals/Dummies
wizard, RM413, UG19, UG49.
TRFUNC instruction, RM482, UG563.
Trigamma function, RM564.
Truncated samples, RM268, UG533.
TSAYP149.PRG example, RM488.
TVARSET procedure, UG391.
TVARYING instruction, RM484,
UG391.
TVARYING.PRG example, UG391.
Two-stage least squares, UG192,
UG276, UG406.

with AR1 errors, RM6.
instraments list, RM257.
non-linear, RM313.
TYPE instruction, RM485, UG587.

U
UFORECAST instruction, RM486,
UG34, UG300, UG482.
UFOREERRORS procedure, UG308,
UG327.
Unbalanced panel data sets, RM348.
Uncentered R-squared, UG177.
Underflow, UG256.
Uniform distribution
random draws, RM565, UG160,
UG478, UG483.
UNION.PRG example, UG532.
Unit circle
complex numbers on, RM565.
random numbers on, RM553.
Univariate Statistics wizard, RM58,
RM170, RM439, UG15.
UNIX version
graphs, UG109.
UNRAVEL option, RM135, UG202,
UG205.
UNTIL instruction, RM489, UG56,
UG581.
UPDATE instruction, RM490, UG82.
Upper case
converting string to, RM560.
USERMENU instruction, RM491,
UG598.

V
VADD instruction, RM306, RM494.
Valid data
testing for, RM566.
Value at Risk (VaR), UG518.
Value options, UG61.
VAR
Bayesian, RM430, UG377.
Blanchard-Quah, UG358, UG362.
block exogeneity tests, RM372.
Choleski factorization, UG352,
UG362.
conditional forecasts, UG397,
UG400.
decomposition of variance, RM147.
deterministic variables, RM108.
differencing variables, UG343.
dummy observation prior, UG396.
error correction, UG374.
estimating, RM158, UG345.
estimating with LINREG, UG345.
estimating with SUR, UG345.
exogenous variables, RM108.
forecasting, RM188, UG377.
historical decompositions, RM233,
UG372.
hypothesis tests, UG347.
impulse responses, RM148, RM240,
UG364, UG368.
instructions for, RM590.
Kalman filter, UG388.
lag length, UG344, UG348, UG379.
lag sums, RM544.
lags, setting lat of, RM267.
long-run restriction, UG358.
Markov switching, UG454.
moving average
representation, UG351.
neq, UG342, UG345.
orderings, UG363, UG367.
orthogonalization, UG351.
partial, UG386.
preliminary
transformations, UG343.
prior, UG378, UG380, UG383.
random draw for
coefficients, UG492.
seasonal data, UG387.
selection of variables, UG385.
setting up a, RM466, RM495,
UG342.
SPECIFY instruction, UG380.
structural, RM75, UG353, UG362,
UG370.
time-varying coefficients, RM259,
RM263, RM484, UG391.
trends, UG344.
variance decomposition, UG366.
VAR (Forecast/Analyze)
wizard, RM147, RM188, RM233,
RM240, UG346.
VAR (Setup/Estimate)
wizard, RM108, RM128, RM158,
RM267, RM430, RM466, RM495,
UG342.
VARCAUSE.PRG example, UG350.
Variables
accessible, UG142.
data types, UG143.
declaring, RM99, UG144.
defined by RATS
instructions, UG142.
displaying values of, RM113,
UG140.
global, RM280.
input/output of, UG145.
label of, RM539.
local in procedures, RM280, UG583,
UG588.
names (form of), UG144.
reading values into, RM251,
RM374.
reserved, RM573.
scalar, UG136.
setting with COMPUTE, RM50.
user input for, RM368.
VARIABLES instruction, RM495, UG342.

Variance
components model, RM366, UG552.
decomposition of, RM147, UG366.
for equation, RM12.
for FRML, RM197.
moving window, RM309.
of series, RM439.
VARIRF procedure, UG368.
VARLAG.PRG example, RM384, UG348.
VCV instruction, RM496.
VCV option output from, UG43, UG179.
Vector
rectangular from a, UG159.
symmetric from a, UG159.
Vector Autoregressions. See VAR.
VECTOR data type, RM100.
aggregate, UG143, UG148.
Vettering, W. T., RM60.
VREPLACE instruction, RM306, RM499, UG410.
Wald test, UG223.
for logit, probit, RM97.
Weekly data, RM28.
Weight matrix
instrumental variables estimator, UG279.
Weighted least squares, RM272, UG186.
with panel data, UG548.
Weiss, L., UG367.
West, K., UG185.
WHILE instruction, RM501, UG56, UG581.
White
covariance matrix, RM295, UG187.
heteroscedasticity test, UG227.
White, H., UG227.
White, K. J., UG182.
Window
data, RM471.
directing output to, RM339.
displaying data in, UG145.
displaying information in, UG598.
Window (spectral)
leakage, RM471.
smoothing, RM502.
WINDOW instruction, RM502, UG666.
MASK option, RM43.
WINDOW option
PRINT instruction, RM497.
saving forecasts, UG304.
Windows Metafile format, RM140.
Wishart distribution
random draws, RM554, UG485.
Wizards
ARCH/GARCH, RM206.
Box-Jenkins (ARIMA models), RM17.
Calendar, RM28, UG7.
Covariance Matrix, RM496.
Cross Correlations, RM66.
Data, RM80, UG12.
Data (RATS Format), UG81.
Differencing, RM109, UG19.
Equation/FRML, RM142, RM196.
Exponential Smoothing, RM153.
Functions, UG138.
Graph, RM214, UG23.
Limited/Discrete Dependent
Variables, RM92, RM268.
Moving Window Statistics, RM309.
overview, UG5.
Panel Data Regressions, RM353.
RATS Variables, UG142.
Regression Tests, RM164, RM307, RM385, RM473, UG33, UG222.
Regressions, RM6, RM272, RM396, RM449, UG28.
Show Series Window, RM246, RM356, RM446, RM496.
Single-Equation Forecasts, RM358, RM486, UG300.
Transformations, RM2, RM281, RM420, UG19.
Trend/Seasonals/Dummies, RM413, UG19.
VAR (Forecast/Analyze), RM147, RM188, RM233, RM240, UG436.
VAR (Setup/Estimate), RM108, RM128, RM158, RM267, RM430, RM466, RM495, UG342.
X11, RM507.
WKS files, RM80, UG39, UG73, UG85.
WLS. See Weighted least squares.
WMATRIX option, UG279.
Wooldridge, J. M., UG423.
WRITE instruction, RM505, UG145.
with binary-format files, UG103.
with complex series, RM401.
with FORTRAN format, UG96.
WYSIWYG, UG387.
X
X11 instruction, RM507.
X11 wizard, RM507.
Y
Yule-Walker equations, RM60, RM250.